

Technische-Hochschule Nürnberg
Fakultät Elektrotechnik Feinwerktechnik Informationstechnik

Studiengang Master Elektronische und Mechatronische Systeme (M-SY)

Masterarbeit

von

Firat Gök

**Programmierung eines mobilen Roboters für die autonome
Navigation in einem benutzerdefinierten Hindernisparcours und
das dafür entwickelte API**

Wintersemester 2021/22

Abgabedatum: 14.02.2022

Betreuer:

Prof. Dr. Stefan May

Dipl.-Ing. Karlheinz Ruff

Prof. Dr. Jörg Arndt

Kurt Hüttinger GmbH & Co. KG.

Schlagworte: mobile Robotik, autonomer mobiler Roboter, ROS, autonom, autonom Navigation, rosipy, Pledge-Algorithmus

Prüfungsrechtliche Erklärung der/des Studierenden

Angaben des bzw. der Studierenden:

Name: Gök

Vorname: Firat

Matrikel-Nr.: 3027676

Fakultät: Elektro-, Feinwerk-, Informationstechnik

Studiengang: Elektronische und Mechatronische Systeme

Semester: Wintersemester 2021/2022

Titel der Abschlussarbeit:

Programmierung eines mobilen Roboters für die autonome Navigation in einem benutzerdefinierten Hindernisparcours und das dafür entwickelte API

Ich versichere, dass ich die Arbeit selbständig verfasst, nicht anderweitig für Prüfungszwecke vorgelegt, alle benutzten Quellen und Hilfsmittel angegeben sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Röthenbach a. d. Pegnitz, 14.02.2022

Ort, Datum, Unterschrift Studierende/Studierender



Erklärung zur Veröffentlichung der vorstehend bezeichneten Abschlussarbeit

Die Entscheidung über die vollständige oder auszugsweise Veröffentlichung der Abschlussarbeit liegt grundsätzlich erst einmal allein in der Zuständigkeit der/des studentischen Verfasserin/Verfassers. Nach dem Urheberrechtsgesetz (UrhG) erwirbt die Verfasserin/der Verfasser einer Abschlussarbeit mit Anfertigung ihrer/seiner Arbeit das alleinige Urheberrecht und grundsätzlich auch die hieraus resultierenden Nutzungsrechte wie z.B. Erstveröffentlichung (§ 12 UrhG), Verbreitung (§ 17 UrhG), Vervielfältigung (§ 16 UrhG), Online-Nutzung usw., also alle Rechte, die die nicht-kommerzielle oder kommerzielle Verwertung betreffen.

Die Hochschule und deren Beschäftigte werden Abschlussarbeiten oder Teile davon nicht ohne Zustimmung der/des studentischen Verfasserin/Verfassers veröffentlichen, insbesondere nicht öffentlich zugänglich in die Bibliothek der Hochschule einstellen.

Hiermit genehmige ich, wenn und soweit keine entgegenstehenden Vereinbarungen mit Dritten getroffen worden sind,

genehmige ich nicht,

dass die oben genannte Abschlussarbeit durch die Technische Hochschule Nürnberg Georg Simon Ohm, ggf. nach Ablauf einer mittels eines auf der Abschlussarbeit aufgebrachten Sperrvermerks kenntlich gemachten Sperrfrist

von Jahren (0 - 5 Jahren ab Datum der Abgabe der Arbeit),

der Öffentlichkeit zugänglich gemacht wird. Im Falle der Genehmigung erfolgt diese unwiderruflich; hierzu wird der Abschlussarbeit ein Exemplar im digitalisierten PDF-Format auf einem Datenträger beigefügt. Bestimmungen der jeweils geltenden Studien- und Prüfungsordnung über Art und Umfang der im Rahmen der Arbeit abzugebenden Exemplare und Materialien werden hierdurch nicht berührt.

Röthenbach a. d. Pegnitz, 14.02.2022

Ort, Datum, Unterschrift Studierende/Studierender



Datenschutz: Die Antragstellung ist regelmäßig mit der Speicherung und Verarbeitung der von Ihnen mitgeteilten Daten durch die Technische Hochschule Nürnberg Georg Simon Ohm verbunden. Weitere Informationen zum Umgang der Technischen Hochschule Nürnberg mit Ihren personenbezogenen Daten sind unter nachfolgendem Link abrufbar: <https://www.th-nuernberg.de/datenschutz/>

Kurzfassung

Die vorliegende Arbeit soll die Umsetzung eines Exponates für Bildungseinrichtungen mit einem mobilen Roboter als Intelligenzträger untersuchen und entwickeln. Dazu wurde im Laufe der Arbeit eine API entwickelt, das als Bindeglied zwischen der Anwendung und der Roboterkontrollarchitektur dient. Hierbei wurde das System mit Intelligenz angereichert, um so die Fähigkeiten und Klugheiten der mobilen Roboter, die heute möglich sind, spielerisch zu transportieren. So wurde mithilfe von Sensoren eine Hinderniserkennung implementiert, um vorrangig sicherheitsrelevante Fähigkeiten hardwarenah abzuwickeln, aber es wurde auch ein Ansatz für einen Algorithmus entwickelt, der ermöglicht, den Roboter aus einem Labyrinth zu befreien. Durch die offene Gestaltung der API kann die grafische Benutzeroberfläche je nach Einrichtung und Zweck angepasst werden. Das System selbst, das in dieser Arbeit untersucht und experimentell entwickelt wurde, beinhaltet viele kleinere Subsysteme, die im Einzelnen verschiedene Funktionalitäten ermöglichen. Durch diese Arbeit konnte ein Wissenslevel erarbeitet werden, um so ein System für die Firma Kurt Hüttinger GmbH & Co. KG zu entwickeln und gegebenenfalls mit weiteren Intelligenzen zu erweitern.

Abstract

This thesis aims to investigate and develop the implementation of an exhibit for educational institutions using a mobile robot as an intelligence carrier. For this purpose, an API was developed during the course of the work, which serves as a link between the application and the robot control architecture. In this process, intelligence was added to the system in order to playfully convey the capabilities and smarts of mobile robots that are possible today. For example, obstacle detection was implemented using sensors to handle primarily safety-related capabilities in hardware, but an approach was also developed for an algorithm that allows the robot to escape from a maze. Due to the open design of the API, the graphical user interface can be customized depending on the setup and purpose. The system itself, which was studied and experimentally developed in this work, includes many smaller subsystems that individually enable different functionalities. Through this work a level of knowledge could be worked out to develop such a system for the company Kurt Hüttinger GmbH & Co. KG and to extend it with further intelligences if necessary.

Danksagung

An diese Stelle möchte ich die Gelegenheit nutzen, einigen Menschen zu danken, die mich in meinem Studium unterstützt und somit diese Masterarbeit erst möglich gemacht haben.

Mein erster Dank gilt Herrn Prof. Dr. Stefan May, der diese Abschlussarbeit betreut und unterstützt hat. Für die hilfreichen Anregungen und die konstruktive Kritik bei der Erstellung dieser Arbeit möchte ich mich herzlich bedanken. Ich bedanke mich auch für die Roboterlösung, die ich zur Verfügung gestellt bekommen habe. Des Weiteren möchte ich der Firma Hüttinger danken, für ihre Aufgeschlossenheit und ihre unbürokratische Hilfe. Außerdem möchte ich Frau Ammon für das Korrekturlesen meiner Masterarbeit danken. Ein besonderer Dank gilt meinen Eltern, die mir mit ihrer Unterstützung mein Studium erst ermöglicht haben, ohne dies an Bedingungen zu knüpfen.

Inhaltsverzeichnis

Kurzfassung	III
Abstract	III
Danksagung	IV
Abkürzungsverzeichnis	VII
Formelzeichen und Indizes	VIII
1 Einleitung	1
1.1 Problemumfeld	2
1.2 Zielsetzung	3
2 Grundlagen - Stand der Technik	4
2.1 Roboter	4
2.2 Roboterkontrollarchitekturen	5
2.2.1 ROS - Robot Operating System	6
2.2.1.1 ROS Einführung	6
2.2.1.2 ROS Konzept	8
2.2.1.2.1 ROS-Datensystem-Ebene	8
2.2.1.2.2 ROS-Berechnungsgraphen-Ebene	8
2.2.1.2.3 ROS-Community-Ebene	11
2.3 Sensorik	11
2.3.1 Entfernungsmessungen	12
2.3.1.1 Laufzeitmessung	12
2.3.1.2 Phasendifferenzmessung	14
2.3.1.3 Triangulation	14

2.3.1.4	Ultraschallsensor	15
2.3.1.5	LiDAR	15
2.3.2	Odometrie	16
2.3.3	Gyroskope	17
2.3.4	Kamera	18
2.4	Navigation	19
2.4.1	Relative Navigation	19
2.4.2	Pledge-Algorithmus	20
3	Projektdurchführung	23
3.1	Anforderungen	23
3.1.1	Ermittlung der Stakeholder	23
3.1.2	Funktionale Anforderungen	24
3.1.3	Nicht-funktionale Anforderungen	25
3.1.4	Architektur	25
3.2	Entwurf und Implementierung	28
3.2.1	Implementierung Ultraschall-Sensor	29
3.2.2	Implementierung QR-Code Scanner	32
3.2.3	Implementierung Netzwerkkommunikationsmodule	33
3.2.4	Implementierung Roboter Module	33
3.2.5	Implementierung Pledge-Algorithmus	34
3.2.6	Implementierung API	36
4	Ergebnisse und Diskussion	38
5	Tests	40
5.1	Test mit der ROS Umgebung	40
5.2	API Test mit einer grafischen Oberfläche	41
6	Zusammenfassung und Ausblick	43
Abbildungsverzeichnis		44

Abkürzungsverzeichnis

API	Application Programming Interface
ROS	Roboter Operating System
DNS	Domain-Name-System
IP	Internet Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
LiDAR	Light detection and ranging
IMU	Inertial Measurement Unit
MINT	Mathematik, Informatik, Naturwissenschaft und Technik

Formelzeichen und Indizes

s	Strecke
t	Zeit
D	Distanz oder Strecke
f	Frequenz oder Brennweite
λ	Wellenlänge
b	Parallaxe
x_{t+1}	x-Position zum Zeitpunkt t+1
y_{t+1}	y-Position zum Zeitpunkt t+1
θ_{t+1}	Orientierung zum Zeitpunkt t+1
x_t	x-Position zum Zeitpunkt t
y_t	y-Position zum Zeitpunkt t
θ_t	Orientierung zum Zeitpunkt t
ΔU_t	Positionsänderung zum Zeitpunkt t
$v_{current}$	Aktuelle Geschwindigkeit
v_{max}	Maximale Geschwindigkeit
$d_{current}$	Aktuelle Distanz

d_{max}	Maximale Distanz
U_{Pin}	Spannung am Pin vom Raspberry Pi
U_{Echo}	Spannung des Echo Signals vom Ultraschallsensor
$R_{1\dots 3}$	Spannungsteiler Widerstände

1 Einleitung

Die autonome mobile Robotik ist heutzutage nicht mehr wegzudenken und hat eine wichtige Rolle eingenommen. Die mobilen Roboter nehmen banale Aufgaben wie Staubsaugen bzw. Wischen (siehe Abbildung 1.2), aber auch komplexere Aufgaben in der Logistik und Industrie (siehe Abbildung 1.1) wahr. Der Einsatz der Roboter soll auch in den kommenden Jahren eine wichtige Rolle bei Katastrophen wie Brand übernehmen. [7]



Abbildung 1.1: Einsatz von mobilen Robotern in der Industrie (a) KMR QUANTEC [14]
 (b) Magnus der Firma Baumüller [2]

Ein weiter Katastrophenfall ist die Corona Pandemie, die seit 2020 die ganze Welt in ihrem Bann hält. Die Pandemie hat viele Opfer gefordert, allein in Deutschland über 100.000, Stand 25.11.2021 [23]. Durch die leichte Übertragung hat sich das Virus zu einer Pandemie entwickelt. Die Corona Pandemie hat deutlich gemacht, dass Keime und Viren auf Oberflächen an öffentlichen Orten und Arbeitsplätzen getötet werden müssen und dass es nicht nur ein Thema in Krankenhäusern ist. [15] So konnte die autonome mobile Robotik sogar in der Pandemie an Wichtigkeit gewinnen. Es werden Serviceroboter entwickelt, die durch verschiedene Reinigungs- und Desinfektionsverfahren auf Oberflächen die Infektionsketten stoppen. Die Aufgabe der Desinfektionsroboter ist es zum Beispiel Türgriffe abzuwischen und im Anschluss über UV-Licht die Keime zu neutralisieren, sodass auch



(a)



(b)

Abbildung 1.2: Einsatz von mobilen Robotern in privaten Haushalten (a) Roomba Combo Saug- & Wischroboter der Firma iRobot [12] (b) Indego S+ 500 Roboter-Rasenmäher der Firma Bosch [3]

an schwer zugänglichen Stellen die Viren bzw. Keime getötet werden. Der mobile Roboter erkennt selbstständig durch Sensoren alle Objekte, die gereinigt werden müssen. [15] In dieser Arbeit soll eine Schnittstelle entwickelt werden, um die mobilen Roboter vereinfacht über Netzwerknachrichten zu bedienen. Dieses System soll als Grundlage dienen, um vereinfacht mobile Roboter in Bildungseinrichtungen als Lehrmaterial einzusetzen. Durch die standardisierten Schnittstellen können gezielt Benutzeroberflächen für unterschiedliche Anforderungen geschaffen werden, ohne die Implementierung am Roboter zu ändern.

1.1 Problemumfeld

Die Abschlussarbeit entstand in Zusammenarbeit mit der Firma Kurt Hüttinger GmbH. Dabei war die Firma, während des Bearbeitungszeitraums, in einer wirtschaftlich schlechten Lage, da die Corona Pandemie das klein- und mittelständische Unternehmen belastete. Die Firma Hüttinger ist ein Unternehmen, das auf Messe- und Museumsbau spezialisiert ist. Da aber durch die Pandemie viele Ausstellungseinrichtungen geschlossen oder eingeschränkt sind, ist die Firma in eine wirtschaftliche Schieflage geraten, daher konnte der ursprüngliche mobile Roboter der Firma EduArt Robotik GmbH nicht bestellt werden. Die Arbeit wurde schließlich auf einen prototypischen Aufbau mithilfe vom Kobuki umgestellt. Infolgedessen wurde ein System angestrebt, das als Exponat in einem Museum bzw. einer Bildungseinrichtung ausgestellt werden kann. Die Abschlussarbeit soll eine offene Plattform bieten für die Bedienung von mobilen Robotern. Demnach können die

Benutzeroberflächen für verschiedene Einrichtungen und Anforderungen entwickelt und geändert werden, ohne die roboternahe Implementierung zu ändern.

1.2 Zielsetzung

In dieser Arbeit wird ein System ausgearbeitet, das in Bildungseinrichtungen zum Einsatz kommen könnte, um Kindern und Jugendlichen die Technik der allgegenwärtigen mobilen Roboter näher zu bringen. Dazu wird eine API entwickelt, die diese mobilen Roboter bedient und leichter zugänglich macht. So kann die Benutzeroberfläche zum Bedienen der mobilen Roboter offen gehalten werden, da das API über Netzwerknachrichten bedient werden kann. Weiterhin werden in das API wichtige Funktionalitäten wie Hinderniserkennung implementiert. Dabei wird ein Kobuki Roboter für die experimentellen Tests verwendet.

2 Grundlagen - Stand der Technik

In diesem Kapitel werden Grundlagen beschrieben, die relevant für das Verständnis dieser Arbeit sind. Die Arbeit über mobile Roboter ist ein aktuelles Thema und wird in vielen Arbeiten behandelt, deswegen entsteht hier eine Beschreibung, welche den Stand der heutigen Technik wiedergibt.

2.1 Roboter

In der DIN ISO 10218-1:2021 wurde der Begriff Industrieroboter Roboter folgendermaßen erläutert: „Industrieroboter Roboter [sind] automatisch gesteuerte(r), frei programmierbare(r) Mehrzweck-Manipulator(en) [...], der in drei oder mehr Achsen [...] programmierbar ist und zur Verwendung in Automatisierungsanwendungen [...] in einer Industrienumgebung [...] entweder an einem festen Ort oder fest an einer mobilen Plattform [...] angeordnet sein kann,“ [5]

Der Begriff mobiler Roboter ist nicht eindeutig dargelegt, aber wenn man die einzelnen Begriffe für sich selbst definiert, bedeutet mobiler Roboter, dass eine Maschine nicht fest, sondern frei in der Bewegung ist, was von ihrer aktuellen Umgebung abhängt. Die Aktion des mobilen Roboters ist erst zum Zeitpunkt der Ausführung bekannt, es läuft nicht nach einem bestimmten Ablaufplan ab, sondern entscheidet durch Sensorik die nächste Handlung. In dieser Arbeit nehmen wir an, dass ein mobiler Roboter eine frei programmierte Maschine ist, die durch Umgebungseinflüsse interagiert. Dabei ist zum Zeitpunkt der Programmierung die Beeinflussung auf den Roboter unbekannt. [8, S.1ff]

Die aktuellen Staubsaugroboter sind viel intelligenter als noch vor paar Jahren. Das Prinzip ist, frei in einem Raum zu saugen und wenn ein Hindernis leicht berührt wird, die Richtung zu ändern und weiterzumachen. Dieser Vorgang wird so lange wiederholt,

bis verschiedene Aspekte erreicht sind, wie zum Beispiel eine vom Benutzer*in definierte Zeitangabe wird erreicht, Batterie Kapazität erreicht einen minimalen Level oder Saugbehälter ist überfüllt. Die aktuellen Roboter können auch Karten von den Räumen erstellen (siehe Abbildung 2.1[a]), die dann über eine Benutzeroberfläche vom Benutzer*in definiert werden, anschließend kann der Benutzer*in auf der Karte bestimmte Bereiche als unzugänglich markieren, wodurch die Roboter diesen Bereich nicht betreten. Deshalb kann der Roboter auch neue Hindernisse erkennen und die Verbraucher warnen mit einem Foto(siehe Abbildung 2.1[b]). Dazu wird eine weiterverbreitete Technik, das SLAM Verfahren, verwendet.



Abbildung 2.1: Die intelligenten Funktionen eines Staubsaugroboters der Firma iRobot
 (a)[10] (b)[11]

2.2 Roboterkontrollarchitekturen

Die Roboterkontrollarchitektur bezieht sich auf den Aufbau der Software für die Robotersteuerung. Dabei ist für eine gute Softwarearchitektur entscheidend, dass diese klare Strukturen hat, Daten- und Kontrollfluss gegliedert sind um die Wartbarkeit eines Programms zu verbessern. Die Kontrollarchitektur eines Roboters ist noch mal komplizierter, denn es ist ein System, das in einem geschlossenen Regelkreis läuft, das heißt Verarbeitung von Sensordaten in Echtzeit. Es kann dabei nicht auf einen standardisierten Daten-

und Kontrollfluss gesetzt werden, schließlich sind die Datenarten jedes Roboters unterschiedlich. Dadurch kann festgehalten werden, dass es nicht die eine Architektur gibt, die besser als alle anderen ist. In dieser Arbeit wird das ROS (siehe Unterabschnitt 2.2.1) verwendet, es ist einer der derzeit aktuellen und immer weiter verbreiteten Vertreter von Software-Tools und Kontrollstrukturen, was dabei hilft, einfach und robust ein System aufzubauen, ohne komplett eine Roboterkontrollsoftware zu programmieren. [8, S.317ff]

2.2.1 ROS - Robot Operating System

2.2.1.1 ROS Einführung

Roboter Operating System oder kurz ROS ist ein quelloffener Software-Framework oder Meta-Betriebssystem für Roboter. Es stellt wie ein Betriebssystem Dienste zur Verfügung wie zum Beispiel Hardwareabstraktion, Gerätetreiber oder Paketmanagement. Dabei bietet ROS auch Softwaretools und Bibliotheken an. Das Framework setzt auf individuelle und entkoppelten Komponenten. [20] Diese vielen Prozesse können auf unterschiedlichen Rechnern laufen und werden bei ROS zur Laufzeit mit Peer-to-Peer-Topologie vernetzt (siehe Abbildung 2.2).

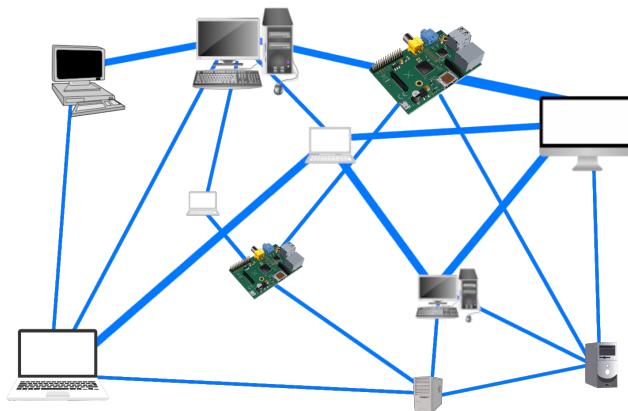


Abbildung 2.2: Netzstruktur einer Peer-to-Peer-Topologie

Dadurch wird das System dezentralisiert und braucht keine zentralen Server. Durch diese Topologie sind alle Rechner gleichwertig, somit wird unnötiger Netzverkehr vermieden

anders als bei Server-Client-Topologie (siehe Abbildung 2.3) da müssen alle Anfragen an den Server vermittelt werden.

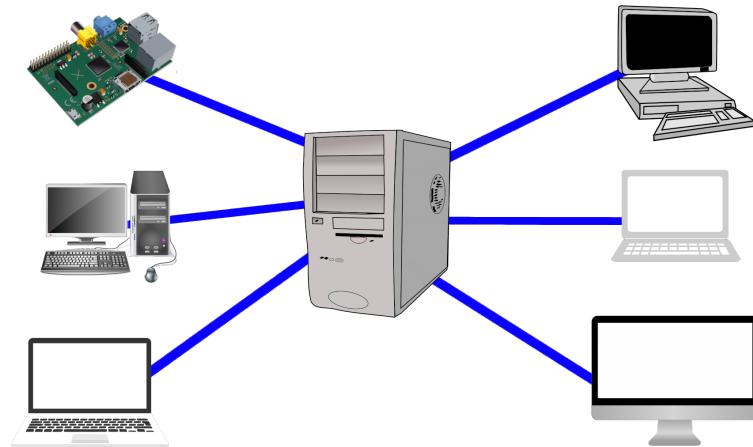


Abbildung 2.3: Netzstruktur einer Server-Client-Topologie

Bei ROS gibt es lediglich einen Master, der für die Namensdienste, also Verzeichnisse der Prozesse zuständig ist, damit die Rechner sich gegenseitig finden. [8] Aus den Zielen des ROS haben sich die folgende wichtigen Punkte deutlich gemacht.

Mehrsprachig, am Anfang wurden nur C++, Python, Octave und LISP unterstützt und jetzt werden alle wichtigen Sprachen unterstützt. Dieses Ziel soll dazu dienen, dass es keine Beschränkung für die Programmierer gibt, denn die Präferenz einzelner Entwickler sind unterschiedliche Programmiersprachen.

Schlank, denn ROS ist nur ein Framework und dient nur dazu, Daten in das Programm zu führen und Resultate abzufangen und zu übertragen. Dementsprechend setzt ROS voraus, dass alle Treiber und Algorithmen keine Abhängigkeit haben und eigenständig implementiert werden.

Werkzeugbasiert, ROS bietet viele verschiedene Tools an, die verwendet werden können, um z. B. Tests zu vereinfachen mit rostest, Verzeichniswechsel zu ROS Paketen mit roscd oder zur Visualisierung der Netztopologie mit rqt_graph. Diese ganzen Softwaretools vereinfachen und helfen bei der Entwicklung. [8, S.326ff]

2.2.1.2 ROS Konzept

2.2.1.2.1 ROS-Datensystem-Ebene

Die ROS Architektur hat drei Entwicklungsschichten, dazu zählt als erstes das ROS Datensystem. Die Datensystemebene umfasst grundsätzlich die ROS-Ressourcen, die wichtigsten werden im Folgenden näher beschrieben.

Als Erstes sind Pakete zu erwähnen, sie bilden die Grundeinheit für die Software in ROS. Sie beinhalten ROS-Prozesse wie Nodes, Bibliotheken, Konfigurationsdaten oder alles andere, was sinngemäß zusammengehört. Pakete sind die elementarste Einheit, die erzeugt und veröffentlicht werden können.

Metapakete sind verwandte Pakete, die in eine Gruppe zusammengefasst werden. Dabei kann beispielhaft das Paket vom Kobuki erwähnt werden, welches verschiedene Funktionalitäten in einem Metapaket zusammenfasst, wie Kobuki automatische andocken, Kobuki sichere Kontrollfahrt oder das Paket für normale Kobuki Fahrt. Diese funktionalen Pakete werden dann in einem Metapaket gesammelt.

Paket-Manifests, die Datei „package.xml“ liefert Metainformation über das Paket wie z. B. Name, Version, Lizenzinformation, Abhängigkeiten und andere Metadaten. [19, 4]

2.2.1.2.2 ROS-Berechnungsgraphen-Ebene

Als weitere Ebene in der ROS-Architektur ist die ROS-Berechnungsgraphen-Ebene zu erwähnen. Sie ist ein Peer-to-Peer Netzwerk aus Ros-Prozessen. Dabei sind die grundlegenden Konzepte der Ebene, *Nodes*, *Master*, *Parameter Server*, *Messages*, *Services*, *Topics* und *Bags*, die allesamt unterschiedliche Daten bereitstellen. Im Folgenden werden die wesentlichsten Konzepte der Ebene näher erläutert.

Nodes (Knoten) sind die Prozesse, die Aufgaben durchführen. Die ROS Architektur ist modular aufgebaut, so kann eine Roboterkontrollarchitektur aus mehreren Knoten bestehen. Die einzelnen Knoten können verschiedene Berechnungen steuern wie Laserentfernungsmessung, Lokalisierung, Manövrieren oder Pfadfindung. Nodes können mithilfe ROS-Bibliotheken implementiert werden und können auch untereinander kommunizieren, wenn sie parallel laufen.

Master stellt die Namensregistrierung und Namenssuche für die ROS-Berechnungsgraphen-Ebene zur Verfügung. Der Master ist ein zentraler Bestandteil, ohne ihn könnte kein Kommunikationsaustausch stattfinden. ROS arbeitet mit TCPROS das aber standardmäßig TCP/IP-Socket verwendet, wobei es XMLRPC-Nachrichten sind, die mit TCP verschickt werden. Der Standard-TCP-Port läuft auf 11311 und wartet auf Nachrichten, wobei die genaue Netzwerkadresse eingestellt werden kann, aber standardmäßig auf localhost, also auf die Netzwerkadresse des laufenden Rechners eingestellt ist. Das ROS über Netzwerknachrichten arbeitet, hat den Vorteil, dass leicht auf und mit entfernten Rechner kohäriert werden kann. Der ROS-Master kann mit *roscore* oder *roslaunch* gestartet werden, unterdessen müssen sich alle Prozesse am Anfang beim Master registrieren (siehe Abbildung 2.4) um die Bekanntmachung der Knoten mitzuteilen ob publiziert oder abonniert wird.

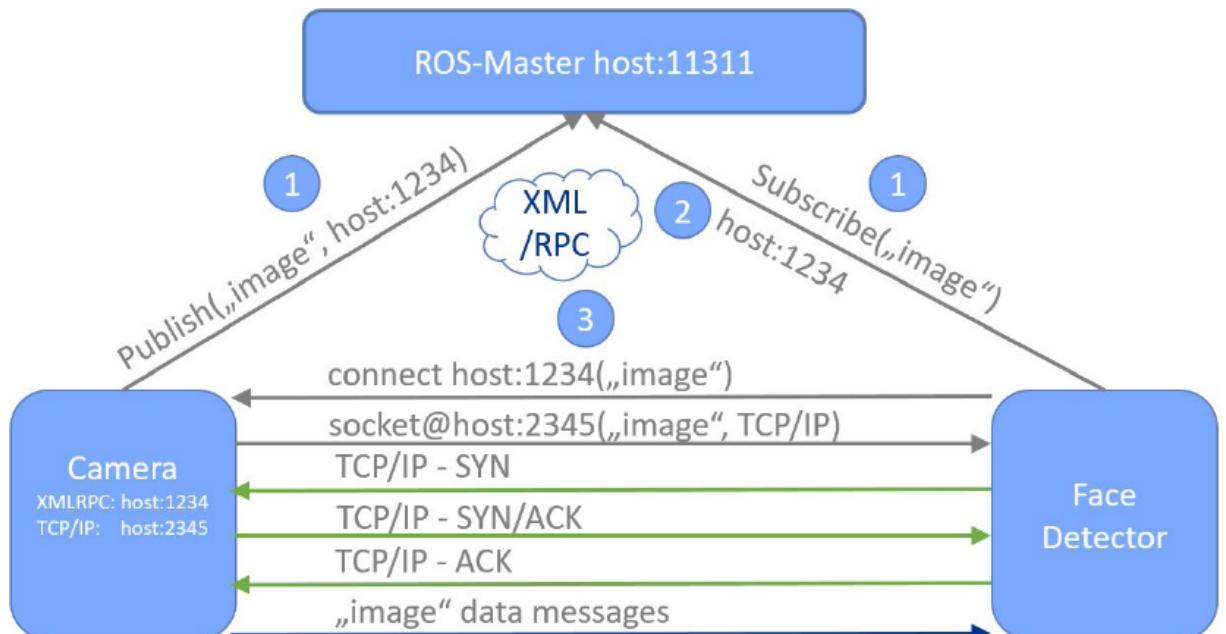


Abbildung 2.4: ROS-Kommunikationsmodes [4]

Die Abbildung 2.4 zeigt deutlich, dass der Master nur als ein Vermittler dient und nur die Verbindung der Nodes veranlasst, ähnlich wie ein DNS-Server. Die Daten werden nur noch zwischen den Nodes verschickt, ohne Bezug zum Master. So wird eine Stabilität des ganzen Systems garantiert, denn wenn ein Prozess ausfällt, hat es keinen Einfluss auf die anderen Prozesse oder Master. So kann auch bei einem bestehenden Datenfluss der

Ausfall durch Wartezeit überwunden werden ohne dass alle Prozesse aufgehalten werden.

Der Parameter-Server ermöglicht, die Konfiguration von Daten nach Schlüssel an einem zentralen Ort zu speichern, aber ausgelagert vom Programmcode. Der Parameter Server läuft innerhalb des ROS-Master und ermöglicht so Quellcodeabschnitte variabel zu halten. Durch Verwendung des Parameter-Server müssen bei Änderungen nicht die Nodes neu kompiliert werden.

Messages sind Datenstrukturen, die zur Kommunikation von Knoten miteinander verwendet werden. Diese Nachrichten werden in einer `.msg`-Datei definiert. Es werden einfache Datentypen und mehrdimensionale Arrays unterstützt. Nachrichten können beliebig verschachtelt werden so ähnlich wie bei *C-structs*.

Topics sind Namen für Nachrichten, die zur Identifikation der Nachricht dient. ROS arbeitet wie zuvor erwähnt mit Publish/Subscribe-Semantik so kann ein Knoten eine Nachricht zu einem bestimmten Topic veröffentlichen (Publish) und von einem oder mehreren Knoten kann dieser Topic abonniert (Subscribe) werden (siehe Abbildung 2.5), um die Nachricht zu erhalten. Dadurch müssen die Knoten sich nicht kennen, vereinfacht gesagt kann es als ein BUS-System angenommen werden. Jeder kann sich am BUS beteiligen, um Nachrichten zu verschicken oder zu empfangen.

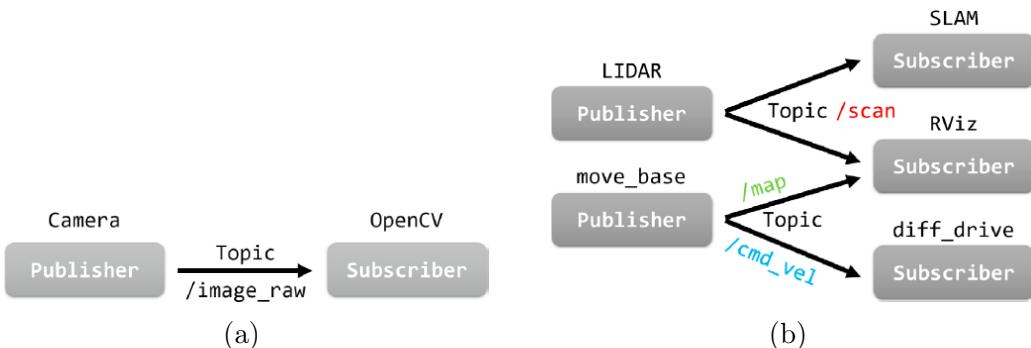


Abbildung 2.5: ROS-Topic Modell (a) Einfacher Publischer und Subscriber Modell (b) Multiple Publischer und Subscriber Modell [4]

Services ist ein Kommunikationsmodell, das bidirektional ist, also ein Datenfluss in beide Richtungen. Dabei arbeiten Services als Anfrage-Antwort-Modell und werden ähnlich wie Messages in `.srv`-Datei definiert.

Bags sind eine Art Format zur Speicherung und Wiedergabe von ROS-Nachrichten. Dies kann dazu verwendet werden, um Sensordaten, die schwer zu sammeln sind, zu speichern, um dies bei der Entwicklung abzuspielen. [19, 4]

2.2.1.2.3 ROS-Community-Ebene

Die dritte und letzte Schicht ist die Community Schicht, sie dient zum Austausch von Software und Wissen rund um ROS. Dabei gibt es verschiedene Plattformen und Lösungen, die weltweit zum Austausch zur Verfügung gestellt werden. Dazu zählt z. B. ROS-Distribution, die eine versionierte konsistente Softwaresammlung ist, Repositories, sie erweitern und ergänzen die Softwarepakete von ROS mit Softwarelösungen des weltweiten ROS-Netzwerks und Institutionen. Die ROS-Wiki Webseiten, sie sind die Hauptwissensvermittler für Dokumentationen und Tutorials. Das Bug-Ticket-System ist eine Zentrale Verwaltungsstelle, der Fehler gemeldet werden und vom ROS-Netzwerk bearbeitet werden. [19]

2.3 Sensorik

Die Handlung eines mobilen Roboters in der Umgebung wird mit Sensoren bestimmt. Die Sensorik widerspiegeln die Sinne eines Menschen (siehe Tabelle 2.1) für einen Roboter. Menschen können mit dem Auge die Umgebung wahrnehmen, wie Entfernung abschätzen, Helligkeit, Objekte und Hindernisse erkennen. Dabei können wir Menschen es auch nicht alles ab dem Zeitpunkt der Geburt, sondern es ist eine langwierige Lernphase, die unser ganzes Leben andauert. Schließlich können wir Objekte erkennen, die wir auch vorher kennengelernt und im Gehirn abgespeichert haben. Ein weiteres Beispiel ist die Entfernungsabschätzung, die wir erst durch Lernen der Messeinheiten erlernt haben. So wissen wir, dass auf der Autobahn die Leitpfosten mit 50 Meter Abstand aufgestellt sind und können daher den Abstand zum vorausfahrenden Fahrzeug einschätzen. Ebenso kann ein Roboter auch die Entfernung schätzen, indem er auf seine spezifischen Sensoren zugreift und die physikalischen Größen misst.

Mensch	Sinn	Organ	Sensorik	Erfassung von
Hören	Gehör	Ohr	Mikrofon	Schall
Sehen	Licht	Auge	Fotozelle	Licht, Konturen
			Kamera	Szenen
Fühlen	Temperatur	Haut	Thermometer	Wärme
	Schwere	Muskel	Waage	Masse
	Kraft		Dehnmessstreifen	Kraft, Drehmoment
	Tastsinn	Nerven	Fühler, Schalter	Form, Lage
Riechen	Geruch	Nase	Rauchmelder	Rauch, Gasen
Schmecken	Geschmack	Zunge/Gaumen	Künstliche Zunge	Inhaltsstoffen

Tabelle 2.1: Parallelen zwischen Mensch und Sensorik [9]

Die Sensorik misst physikalische Eigenschaften und wandelt es in ein elektrisches Signal um. Sensoren können in zwei Gruppen unterteilt werden, die interne oder externe Zustände erfassen. Beschaffenheiten, die im Inneren erfasst werden, sind z. B. Radgeschwindigkeit oder Radbeschleunigung, diese werden propriozeptive Sensorik genannt. Im Gegensatz dazu werden Zustände, die extern also umgebungsabhängig erfasst werden wie z. B. Ultraschallsensoren oder Kameras exterozeptive Sensorik genannt. In diesem Kapitel werden die Sensoren näher beschrieben, die für diese Arbeit verwendet wurden.

2.3.1 Entfernungsmessungen

Damit ein mobiler Roboter die Umgebung wahrnehmen kann, muss er seine Distanz zu Objekten messen. Dabei gibt es drei Methoden, um mit der Sensorik die Entfernung zu messen. Diese drei Arten werden im folgend kurz näher erläutert.

2.3.1.1 Laufzeitmessung

Diese Art von Messung ist eine zeitabhängige und somit hängt die Genauigkeit von der Zeit ab. Dabei sendet die Sensorik Signale aus und misst die Zeit, bis ein Echo vom ausgesendeten Signal empfangen wird. Durch die gemessene Zeit kann die Strecke berechnet werden mit der einfachen Geschwindigkeitsformel (siehe Gleichung 2.1).

$$s = v \cdot t \quad (2.1)$$

Solche Sensoren sind angewiesen vom externen Umfeld also kann es der Gruppe von exterozeptiver Sensorik zugeordnet werden. Die Formel Gleichung 2.2 hat zwei Unbekannte, t die Zeit, die das Signal für die Strecke Hin und Zurück braucht, diese wird gemessen. Die zweite Unbekannte ist die Geschwindigkeit v , diese ist durch die physikalischen Eigenchaften vorgegeben wie z. B. Geschwindigkeit von Schall bei Zimmertemperatur, diese ist 343 m/s oder z. B. die Lichtgeschwindigkeit ist 299792458 m/s. So kann die Entfernung gemessen und geschätzt werden.

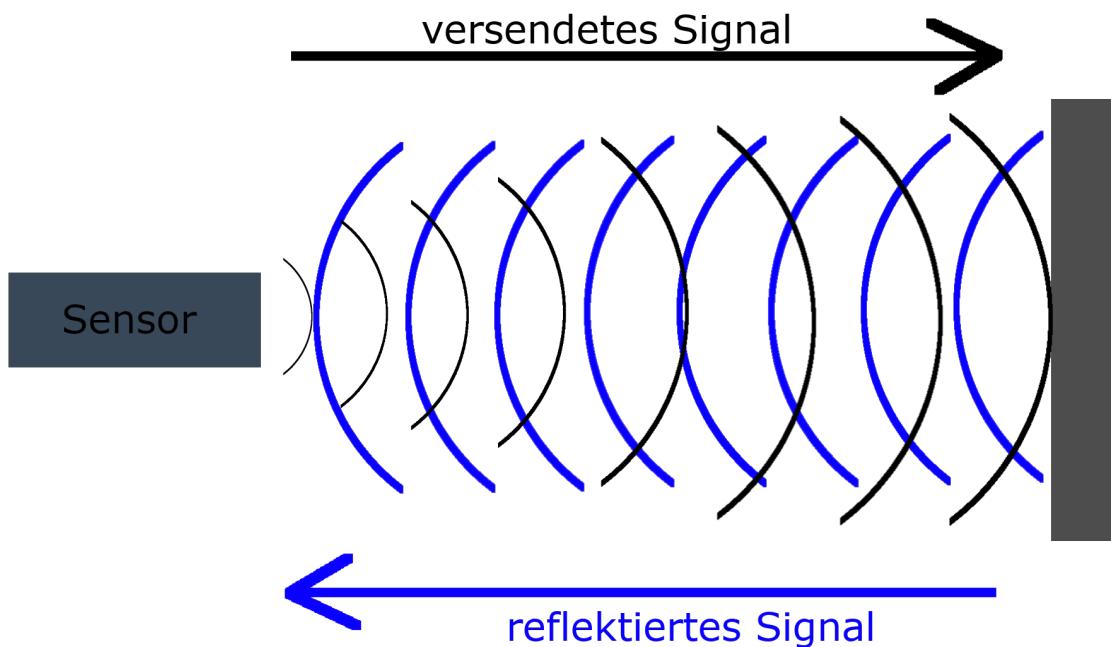


Abbildung 2.6: Grafische Darstellung von Laufzeitmesssystems

$$D = v \cdot \frac{t}{2} \quad (2.2)$$

Dabei ist noch zu beachten, dass die Zeit, die gemessen wurde für den Hin- und Rückweg ist (siehe Abbildung 2.6), die Entfernung zu Objekten ergibt sich somit aus der Hälfte der gemessenen Zeit (siehe Gleichung 2.2).[8, S.36f]

2.3.1.2 Phasendifferenzmessung

Diese Messmethode ist eine ähnliche Art wie die Laufzeitmessung (siehe Unterunterabschnitt 2.3.1.1) nur das hier die Phasenverschiebung von einer Welle gemessen wird. Die ermittelte Phasenverschiebung $\Delta\varphi$ ist proportional zur zurückgelegten Strecke D des Signals (siehe Gleichung 2.3).

$$D = \frac{\Delta\varphi \cdot \lambda}{4\pi} = \frac{\Delta\varphi \cdot v}{4\pi \cdot f} \quad (2.3)$$

Die Phasenverschiebung ist nur ein relatives Verhältnis des Signals zur Wellenlänge, deswegen werden verschiedene Signale modelliert, um mit hochfrequentem Anteil große Auflösung zu erreichen und mit niederfrequentem Anteil werden größere Entfernung bestimmt. [8, S.37f]

2.3.1.3 Triangulation

Bei der Triangulationsmessung wird ein Signal ausgesendet und am Empfänger wird die Auslenkung x zum Abstand b gemessen. Durch den Strahlensatz kann dann die Distanz errechnet werden (siehe Gleichung 2.4).

$$D = f \cdot \frac{b}{x} \quad (2.4)$$

Die folgende Abbildung 2.7 macht deutlich, dass der Bereich nahe am Sensor durch kleine Brennweite f und Parallaxe b möglich ist, aber anders ist es, wenn die Messung weitere Distanzen auflösen soll, denn dann muss die Brennweite f und Parallaxe b möglichst groß sein. Die Sensoren, die über diesen Messverfahren arbeiten, haben einen Totbereich, der keine Messwerte liefert.[8, S.38f]

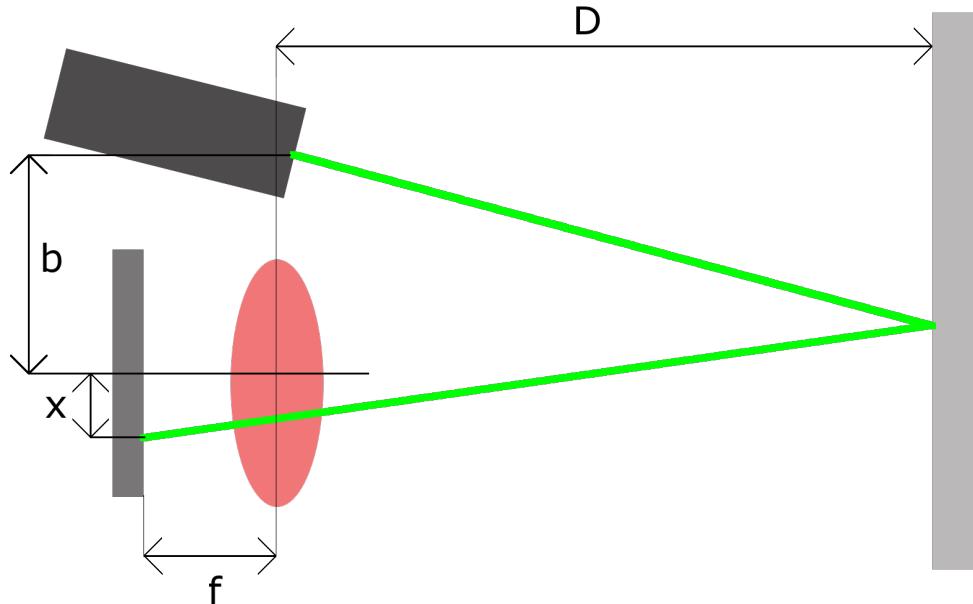


Abbildung 2.7: Triangulationsprinzip zur Distanzmessung

2.3.1.4 Ultraschallsensor

Dieser Typ von Sensorik zur Entfernungsmeßung arbeitet auf der Basis von Laufzeitmessung (siehe Unterabschnitt 2.3.1.1). Ultraschallsensoren haben einen Öffnungswinkel. Außerhalb dieses Winkels und im Blindbereich nahe am Sensor kann keine Messung erfolgen. In dieser Arbeit wurden die Ultraschallsensoren HC-SR04 verwendet, die im Embedded Umfeld ihre Verwendung finden. Der Sensor wird mit einem Trigger Impuls angesteuert, anschließend sendet der Sensor ein Ultraschallsignal. Nachdem das Signal beendet wurde, wird der Echo Ausgang vom Sensor auf Logikpegel High gezogen und die Zeitmessung beginnt. Die Messung wird beendet, wenn ein Echo Signal am Empfänger ankommt, somit wird dann das Echo Ausgang auf Logikpegel Low gesetzt. Das Ultraschallsignal breitet sich mit Schallgeschwindigkeit in der Luft aus.

2.3.1.5 LiDAR

Eine weitere Art von Sensorik, die in dieser Arbeit verwendet wird, ist ein LiDAR Scanner. Bei LiDAR Sensoren werden die zuvor beschriebenen Methoden verwendet, je nach

Hersteller und Modell. Die in dieser Arbeit verwendeten RPLiDAR A1 der Firma Slamtec messen mit der Triangulationsmethode (siehe Unterunterabschnitt 2.3.1.3). Der Scanbereich dieses 2D-Sensors beträgt 360° und hat eine Scanreichweite von $0,15\text{ m}$ und 12 m . Solche Sensoren liefern nur die Distanz die gemessen wurde und über die Nummer oder Index der Liste aller Werte kann der Winkel bestimmt werden. Das bedeutet das es Polarkoordinaten sind, die als Messdaten vorliegen. Diese können aber ins kartesische Koordinatensystem umgerechnet werden (siehe Gleichung 2.5), so kann auch eine Karte des Raumes erstellt werden.

$$\begin{aligned}x &= r \cdot \cos(\varphi) \\y &= r \cdot \sin(\varphi)\end{aligned}\tag{2.5}$$

2.3.2 Odometrie

Das Odometrie Verfahren ist für die Positionsbestimmung relevant. Dabei wird über Sensoren am Rad des Roboters die Eigenbewegung gemessen, die dann durch eine vektorielle Addition verrechnet wird. Dabei werden oft für die Drehwinkelmessung Sensoren wie Impulsgeber oder Inkrementalgeber verwendet.

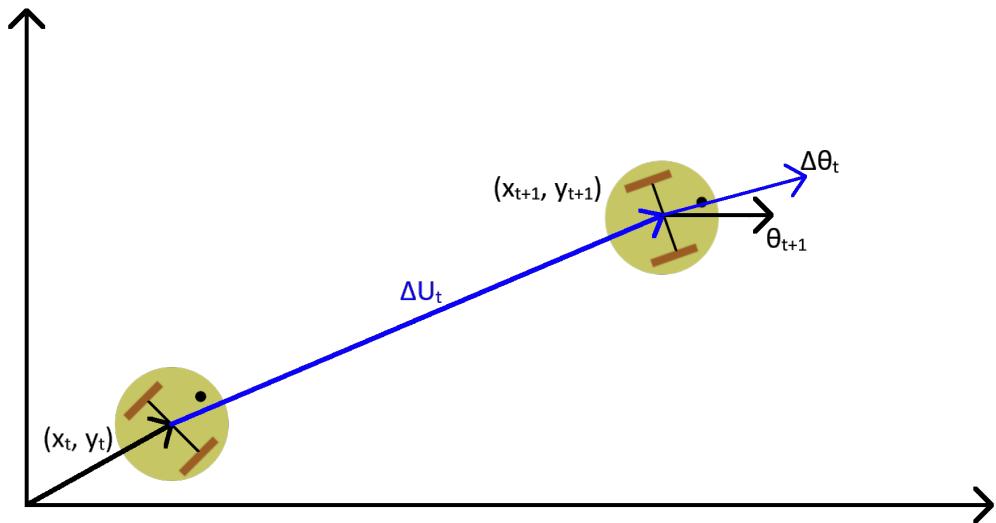


Abbildung 2.8: Bewegung eines mobilen Roboters

Die folgenden diskreten Zustandsgleichungen beschreiben die Bewegung für die Abbildung 2.8:

$$x_{t+1} = x_t + \Delta U_t \cdot \cos(\theta_t) \quad (2.6)$$

$$y_{t+1} = y_t + \Delta U_t \cdot \sin(\theta_t) \quad (2.7)$$

$$\theta_{t+1} = \theta_t + \Delta \theta_t \quad (2.8)$$

Dieses Verfahren hat oft einen Drift, der zustande kommt durch Schlupf an den Rädern oder Kollisionen mit Steinen. Die fehlerhaften Messungen werden jedoch weiter drauf addiert, was dazu führt, dass der Fehler immer größer wird. Um solche Fehler zu kompensieren, muss die Odometrie mit anderen Sensoren kombiniert werden.[9, S.171ff]

2.3.3 Gyroskope

Diese Art von Sensoren erfassen Drehbeschleunigungen bezüglich des Roll-, Nick- und Gierwinkels. In der natürlichen Form ist es ein schnell drehender Kreisel, der beweglich gelagert ist (siehe Abbildung 2.9). Da es ein geschlossenes System ist, führt das dazu, dass alle Impulse konstant bleiben. Versucht eine äußere Kraft das Gyroskop zu beeinflussen, so reagiert er drauf und kippt sich senkrecht zur angreifenden Kraft. Die dabei erzeugten Drehmomente werden erfasst und als Sensormessung ausgegeben.

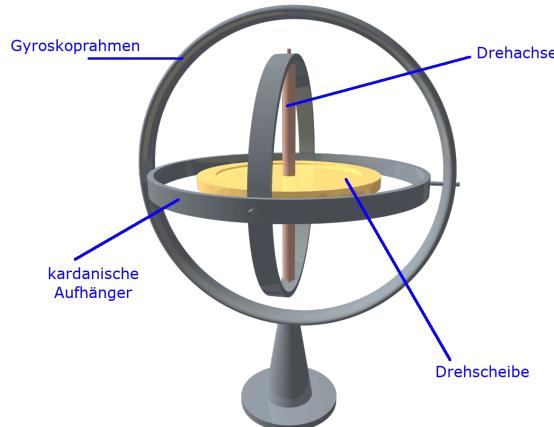


Abbildung 2.9: Natürliche form von Gyroskope [25]

Der Aufbau solcher Messinstrumente wird gegenwärtig als kleines Mikro Elektro Mecha-

nisches System ausgeführt und werden MEMS-Sensoren (engl. Micro Electro Mechanical Systems) genannt. Oft werden Gyroskope zusammen mit linearen Beschleunigungssensoren als eine Komponente vermarktet. Diese Sensoren werden IMU (engl. inertial measurement unit) genannt und werden als MEMS-Sensoren entworfen. [8, S.31f]

2.3.4 Kamera

Die Kamera ist in seiner atomaren Struktur ein Gebilde aus Sensoren. Dabei werden Momentaufnahmen als einzelne Bilder dargestellt. So besteht ein Bild aus mehreren Rastern, die ein Abbild der einzelnen Sensoren sind. Dementsprechend werden die einzelnen Elemente des Bildes, Pixel oder Bildpunkte genannt. Im Grunde werden bei der Bildaufnahme die Sensoren mit der bestimmten Belichtungszeit bestrahlt und wandeln diese Lichtintensität in ein elektrisches Signal um. Die einzelnen Bildpunktsensoren sind letztlich Halbleitersensoren aus Silizium wie CCD-Sensor oder CMOS-Sensor.

Prinzip	CCD	CMOS
Pixelsignal	Ladungen	Spannungen
Arbeitsweise	Integrierend	Proportional
Rauschpegel	Gering	Mittel
Gleichförmigkeit	Hoch	Gering
Geschwindigkeit	Mittel	Hoch
Kosten	Mittel	Gering

Tabelle 2.2: Gegenüberstellung der CCD- und CMOS-Sensoren [9, S.371]

Die CCD-Sensoren arbeiten anders als CMOS-Sensoren (siehe Tabelle 2.2), die einzelnen Ladungen der Sensoren werden in kleinen Zeitabständen ausgelesen, indem die Ladungen durch ein Schieber Register verschoben werden. Diese Methode ist auch der Namensgeber für CCD-Sensoren (engl. Charged Coupled Devices). Bei der Ladungsübertragung gibt es zwei Typen Frame-Transfer-Typen, die über Zwischenspeicher das gesamte Bild in einem Takt übertragen, und Interline-Transfer-Typen, die zeilenweise zum Ausgangsregister ausgelesen werden. Gegenübergestellt werden bei CMOS-Sensoren die Ladungen direkt am Bildpunkt abgeholt und in eine analoge Spannung gewandelt. So besteht beim CMOS-Sensor jeder einzelne Sensor aus Signalverstärkern aus mehreren Transistoren. Die in dieser Arbeit verwendete Kamera ist eine Raspberry Pi Kamera, die mit CMOS-Sensoren arbeitet.

2.4 Navigation

Eine entscheidende Aufgabe eines mobilen Roboters ist die Navigation. Dazu muss ein mobiler Roboter sich frei und kollisionsfrei in der Umgebung bewegen. Die Navigation beinhaltet Zusammensetzungen aus Selbstlokalisierung, Kartenerstellung und Pfadplanung. Die Selbstlokalisierung und Kartenerstellung ist in der Literatur mit dem Erkenntnis SLAM abgefasst. Dieses Verfahren ist nicht Teil dieser Arbeit und wird nicht näher erläutert. In den folgenden Kapiteln wird auf die relative Navigation und den Pledge-Algorithmus eingegangen.[8, S.261ff, 17, S.97f]

2.4.1 Relative Navigation

Als relative Navigation wird verstanden, dass sich das Vehikel im Hinblick zu seinem Umfeld fortbewegt. Die einfachste Form ist, dass der mobile Roboter sich dort hinbewegt, wo keine Hindernisse zu erkennen sind. In der Literatur wird diese Art der Navigation auch Freiraumnavigation genannt. Die Idee dieser Navigation ist es, dass sich das Vehikel im freien Raum bewegt und die Richtung stabil hält, wobei Hindernissen ausgewichen wird, ohne in Schwingung zu kommen. Solche Oszillationen kommen zustande, wenn die Programmierung direkt auf die Sensoren reagiert als Reflex, ohne eine Regelung. Dabei kann eine Fuzzy Regelung herangezogen werden wie die in der Listing 2.1, sie gibt an, in welche Richtung der Roboter fahren soll mit i-ten Abstandswert vom Laserscanner.

```
WENN(Winkel_i ist in Fahrtrichtung) UND  
      (Entfernung_i ist groß)  
DANN fahre in diese Richtung
```

Listing 2.1: Fuzzy Regelung [S.266, 8]

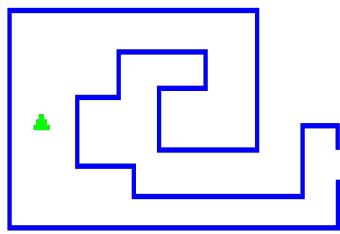
Eine beispielhafte Relative Navigation kann wie folgendermaßen beschrieben werden. Der Roboter fährt so lange geradeaus, bis er auf ein Hindernis trifft. Dabei kann die Vorwärts Geschwindigkeit sofort auf 0 gesetzt werden, wenn die Distanz einen Schwellwert unterschreitet und zugleich wird eine Drehung mit konstanter Geschwindigkeit gestartet, bis die Sicht frei wird. Das Tempo vom Roboter kann durch eine einfache Formel skaliert werden (siehe Gleichung 2.9), wenn die Distanz zum Objekt kleiner ist als die maximale Distanz, dann wird das Tempo verhältnismäßig geregelt, doch falls alles frei ist, wird mit dem größten Tempo gefahren.

$$v_{current} = (d_{current}/d_{max}) \cdot v_{max} \quad (2.9)$$

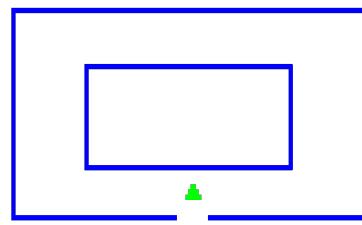
Die Idee hinter der Freiraumnavigation ist das Spurfahren im Straßenverkehr. Die Geschwindigkeit soll nicht beeinflusst werden, solange die Spur des Fahrzeuges frei ist. [8, S.266 ff]

2.4.2 Pledge-Algorithmus

Es gibt verschiedene Methoden, um aus einem Labyrinth zu entkommen wie zum Beispiel die Tiefensuche, Linke- bzw. Rechte-Hand-Regel oder Pledge Algorithmus, der hier näher dargelegt wird. Der Algorithmus wurde nach dem Erfinder John Pledge einem zwölfjährigen Jungen benannt. Pledge Algorithmus funktioniert in jedem Labyrinth das rechtwinklige Ecken hat und erweitert die Linke- bzw. Rechte-Hand-Regel.



(a)



(b)

Abbildung 2.10: Labyrinth

Die Grundidee ist bei der Hand-Regel, die Wand mit der Hand so lange zu verfolgen, bis ein Ausweg gefunden wird (siehe Abbildung 2.11[a]). Diese Idee funktioniert nicht, wenn es auf eine Säule trifft, denn dann dreht sich der Durchläufer im Kreis (siehe Abbildung 2.11[b]).

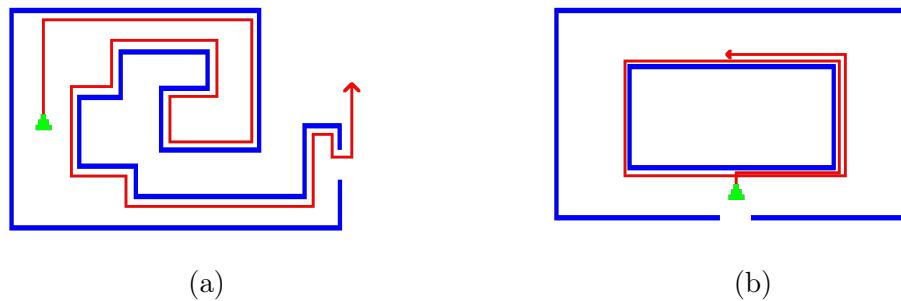


Abbildung 2.11: Grundidee Linke- bzw. Rechte-Hand-Regel

Eine weitere Funktionalität, die diesen Algorithmus erweitern kann, ist die, dass der Durchläufer seine Start Orientierung speichert und beim Laufen im Kreis die Wand verlässt, wenn die aktuelle Orientierung der Start Orientierung entspricht. Diese Funktion hilft beim Laufen im Kreis bei Säulen aber nicht immer bei allen Labyrinthen und führt zu Fehlfunktionalität.

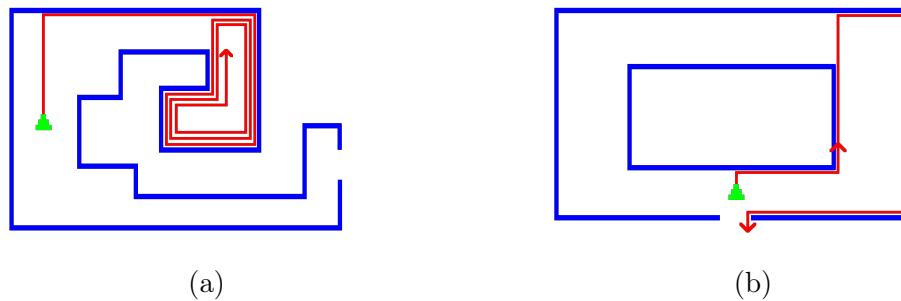


Abbildung 2.12: Linke- bzw. Rechte-Hand-Regel mit Erweiterung Start Orientierung Speicherung

Diese ganzen Funktionalitäten werden beim Pledge-Algorithmus verbessert und so durch alle Irrgärten, die einen Ausgang besitzen, herausfinden. Beim Pledge Verfahren gibt es zwei entscheidende Regeln.

1. Geradeaus fahren oder Laufen bis eine Wand erreicht wird.
2. Die Wand verfolgen, bis der Umdrehungszähler gleich null wird.

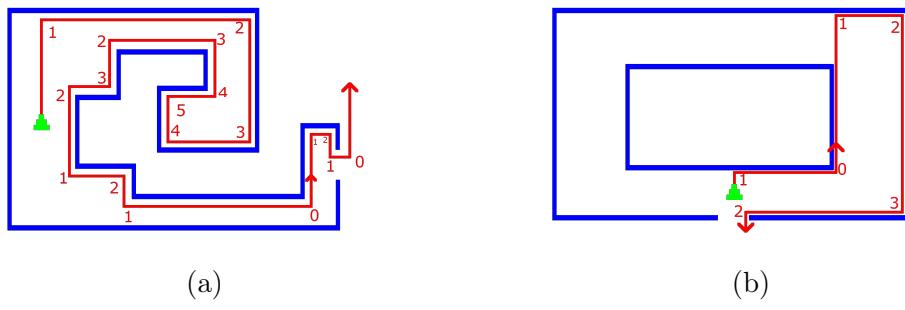


Abbildung 2.13: Ablauf Pledge-Algorithmus an zwei Labyrinth Typen

Diese Regeln werden beim Pledge-Algorithmus iteriert, bis ein Ausweg gefunden wird (siehe Abbildung 2.13). Falls es keinen Ausweg gibt, wird je nach Definition im Uhrzeigersinn oder gegen den Uhrzeigersinn gedreht und der Winkelzähler immer weiter erhöht bzw. verringert. [24, S.75 ff][16, S.250 ff][18, S.318 ff]

3 Projektdurchführung

In diesem Abschnitt der Arbeit wird die eigentliche Durchführung des Projektes näher beschrieben. Dabei wurde das ganze Projekt prototypisch entwickelt, um für spätere Entwicklungen Grundlagen und Wissensstand zu sammeln. Das System soll als Lehrobjekt in verschiedenen Bildungseinrichtungen vermarktet werden. Die Technologien sind im heutigen Zeitalter präsenter denn je, Kinder wachsen mit smarten Innovationen auf. So sollten die zukünftigen Generationen diese Technologien als Bildungsinhalt vermittelt bekommen. Das System in dieser Arbeit soll durch einen mobilen Roboter verschiedene Kompetenzen vermitteln, dazu zählen Programmierung oder logisches Denken. Das System soll eine Basis Applikation auf dem Roboter implementieren und durch verschiedene Mensch-Maschinen-Interfaces sollen unterschiedliche Fähigkeiten vermittelt und dargestellt werden.[1]

3.1 Anforderungen

Um ein System zu entwickeln, ist es wichtig, die Anforderung an das System klar zu definieren. Denn erst wenn alle Beteiligten des Systems die Funktionalitäten und Qualitäten bestimmen, kann die Entwicklung beginnen. So können SOLL- und IST-Zustände abgeglichen und gemessen werden, um später bei der Entwicklung und Herstellung des Systems Fehlinterpretation und im allgemeinen Zeitaufwand zu vermeiden. In den folgenden Kapiteln werden die Anforderungen und die beteiligten Personen des Systems dieser Arbeit aufgezeigt.[6, S.5 ff]

3.1.1 Ermittlung der Stakeholder

Stakeholder sind Personen oder Personengruppen, die das System durch ihre Anforderungen definieren. [6, S.18 ff] Die Stakeholder des Systems dieser Arbeit wurden intern bei

der Firma Hüttinger durch verschiedene Personen bestimmt, da das System eine Lösung bieten soll, die erst später vermarktet werden soll. Dazu haben sich Personen aus der Projektleitung, Softwareentwicklung, Vertrieb und Geschäftsführung zusammengesetzt. Im Folgenden werden die Stakeholdergruppen näher erläutert.

Die **Benutzer*in** sind in diesem System Kinder, Schüler*in oder Studierende in Bildungseinrichtungen. Dazu möchten die Benutzer*in ein Programm erstellen, das nicht nur textuell oder bildlich ist, sondern auch die Aktion sehen, die durch das Coden entstanden ist. Überwiegend werden die Benutzer*in aus den MINT-Fächern mit dem System konfrontiert sein. Dabei soll die Fähigkeit zum Programmieren erlernt werden und nicht nur Syntax von Sprachen gelehrt werden.

Die **Kunden** sind Bildungseinrichtungen wie Schulen, Hochschulen, Universitäten, Museen und Institutionen die Bildung vermitteln aber auch Besucherzentren. Die Kostenträger für das System sind bei Bildungseinrichtungen aber auch oft Kommunen, Städte, Regierungsbezirke oder Staaten.

3.1.2 Funktionale Anforderungen

Funktionale Anforderungen definieren das System hinsichtlich Funktionalität und Verhalten [6, S.37]. Im Folgenden werden die funktionalen Anforderungen tabellarisch dargestellt.

Anforderungs-ID	Anforderung
F1	Der Roboter muss Hindernisse erkennen.
F2	Der Roboter soll eine Ladestation haben.
F3	Der Roboter soll Abstandsmessungen mit einem LIDAR (ToF) durchführen.
F4	Der Roboter soll mit einer Kamera die unterschiedlichen Hindernisse erkennen.
F5	Der Roboter soll die Livekamerabilder Streamen.
F6	Der Roboter soll spezielle Grafiken erkennen.
F7	Der Roboter soll über Lautsprecher dem User Feedback geben.
F8	Der Roboter soll 2D Karten des Labyrinths erstellen.
F9	Der Roboter muss ein Ein/AUS Taster haben.
F10	Der Roboter soll eine Einheitliche Komponente sein.
F11	Das System soll Fernwartung ermöglichen.
F12	Das System muss ein offenes Kommunikationsprotokoll haben.

Tabelle 3.1: Funktionale Anforderungen

3.1.3 Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen erläutern Randbedingungen, die das System haben soll und keine Funktion darstellen [6, S.37]. In der folgenden Tabelle werden die nicht-funktionalen Anforderungen wiedergegeben, die das System dieser Arbeit beschreiben.

Anforderungs-ID	Anforderung
NF1	Der Roboter soll mit Mecanum Rädern ausgestattet sein.
NF2	Der Roboter soll so klein wie möglich entwickelt werden.
NF3	Der Roboter soll einen Batterie Status haben.
NF4	Der Roboter soll eine Batterielaufzeit von mindestens 20 min haben.

Tabelle 3.2: Nicht-funktionale Anforderungen

3.1.4 Architektur

Die Architektur eines Systems beschreibt die Korrelationen und Wechselwirkung mit einzelnen Komponenten. Die Anforderungen führen dazu, dass die Funktionalitäten definiert werden, die im Anschluss als Systemarchitektur dargestellt werden kann (siehe Abbil-

dung 3.1). Die Entwicklung der Systemarchitektur in dieser Arbeit wurde mithilfe der Software *Papyrus Eclipse* unternommen. Dabei konnte mit der Software *UML* und *SysML* als Modellierungssprache verwendet werden.

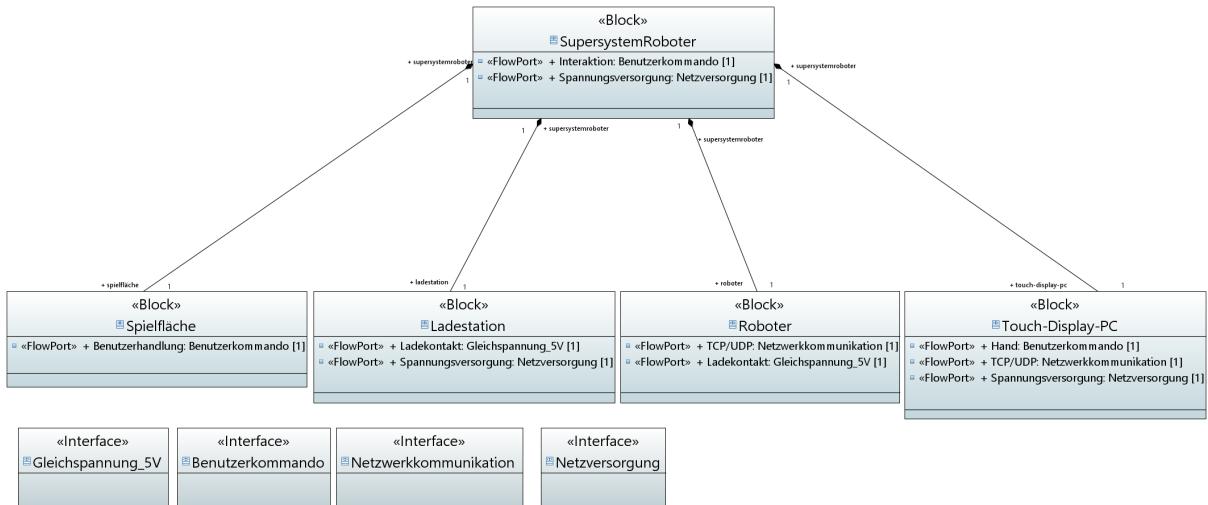


Abbildung 3.1: Supersystem Dekomposition dieser Arbeit

Das System in dieser Arbeit besitzt mehrere Komponenten, die verbunden sind. Dazu z\u00e4hlen die Benutzer*in, die durch ihre Aktionen und Aktivit\u00e4ten den mobilen Roboter beeinflussen, dass zu einer Reaktion der Maschine f\u00fchrt. Die Anwender*in k\u00f6nnen Quellcode \u00f6ber die grafische Oberfl\u00e4che erstellen oder Anweisungen dem mobilen Roboter \u00f6bermitteln aber Benutzer*in k\u00f6nnen als ein Nachbarsystem betrachtet werden. Die ganzen Aktionen der anwendenden Personen werden \u00f6ber die Komponente Touch-Display eingegeben, dadurch ist sie ein Bestandteil des Supersystems. Des Weiteren kann die Spielfl\u00e4che als ein St\u00fcck des Systems erw\u00e4hnt werden, da sie durch ihre Beschaffenheit die Bewegungsm\u00f6glichkeit des Roboters bestimmt. Das elementarste Teil des Systems ist der mobile Roboter, denn erst durch dieses Element k\u00f6nnen alle Teile des Systems interagieren. Die mobilen Roboter besitzen durch einen Akku eine Lebenszeit, die durch eine Ladestation verl\u00e4ngert werden kann, somit ist die Ladestation auch ein Bestandteil des ganzen Supersystems (siehe Abbildung 3.2).

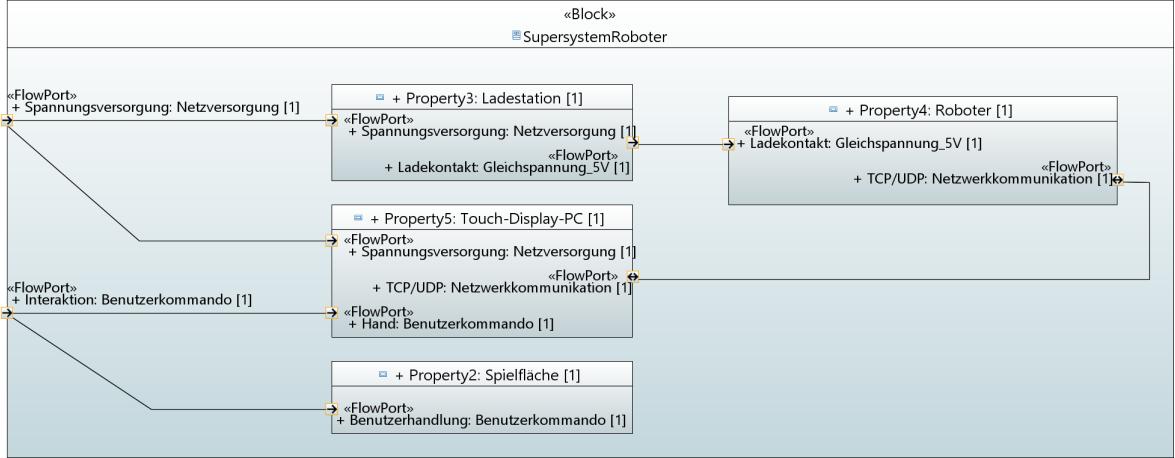


Abbildung 3.2: Supersystem Komposition dieser Arbeit

Die separaten Supersystemkomponenten sind wiederum Subsysteme, die auch als Systemarchitektur dargestellt werden können. Im weiteren Verlauf dieses Kapitels wird aber nur die Hauptkomponente der mobile Roboter atomarer dargestellt, da es der Hauptbestandteil dieser Arbeit ist (siehe Abbildung 3.3).

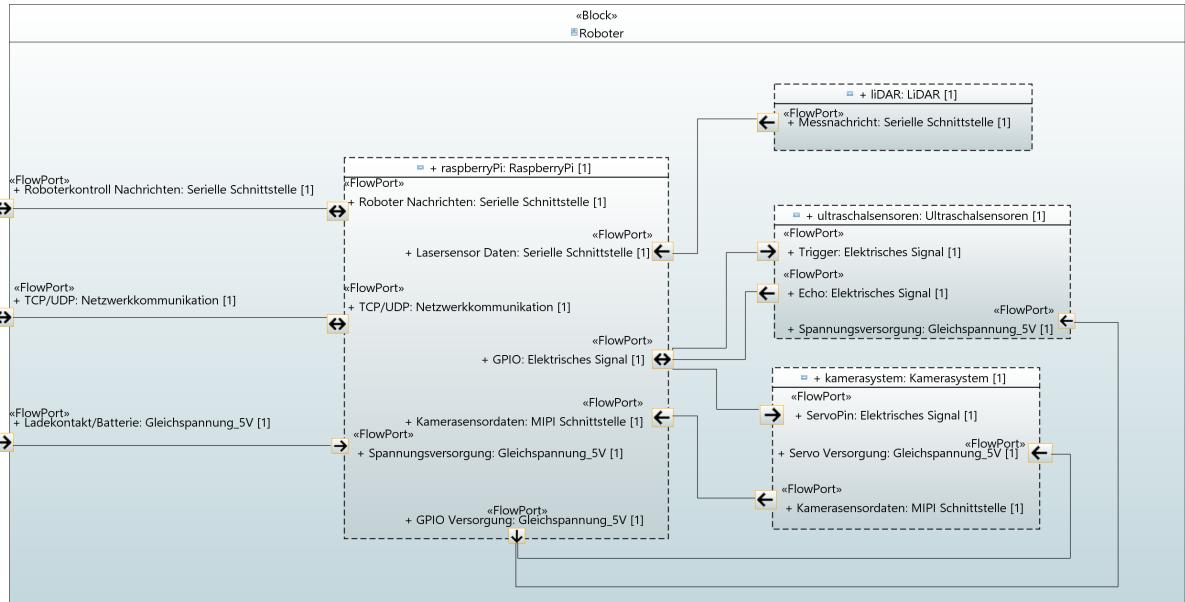


Abbildung 3.3: Mobiler Roboter System Komposition

Das System des mobilen Roboters besteht aus Sensoren, Aktoren und einem Rechner. Diese Oberbegriffe repräsentieren eigene System mit Hardware- und Softwarekomponenten.

3.2 Entwurf und Implementierung

Die Arbeit befasst sich hauptsächlich mit der Programmierung und einer Analyse der Umsetzungsmöglichkeit. Die Systemarchitektur wurde im vorherigen Kapitel umfangreich dargestellt und in diesem Abschnitt wird es um die Implementierung und den Entwurf der Softwarekomponenten gehen. Dabei wurde eine Software entwickelt, die nicht vollständig ist, da es hauptsächlich um die Machbarkeit und einen prototypischen Aufbau ging. Bevor die Softwareimplementierung und der Entwurf näher erläutert wird, werden einige Punkte erwähnt, die für die Umsetzung beschlossen wurden.

Der mobile Roboter, der verwendet wurde, ist der Kobuki Roboter (siehe Abbildung 3.4). Dabei handelt sich um einen Roboter mit Differenzialantrieb. Der Kobuki wurde vom Labor mobile Robotik der Technischen Hochschule Nürnberg zur Verfügung gestellt.

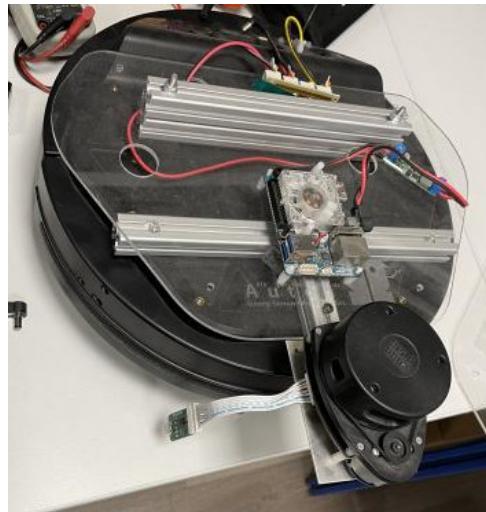


Abbildung 3.4: Mobiler Roboter Kobuki

Die ganze Software wurde in der Programmiersprache Python entwickelt. Dabei wurde die Python Version 3.7 verwendet. Die Programmiersprache wurde gewählt da sie mit der Roboterkontrollarchitektur ROS kompatibel ist. Python ist eine Sprache, die auf al-

len Rechnertypen funktioniert, da es sich um eine interpretierte Sprache handelt und somit nicht direkt mit der Computerhardware arbeitet. Die Programmiersprache ist so entwickelt das sie gut lesbar ist und kompakte Programme erstellt werden können.[22, S. 12 ff] Es können gut lesbare und kleine Programme entwickelt werden, die auch in der Firma, wo diese Arbeit entstanden ist, leichter interpretiert werden.

In dieser Arbeit wurde die Roboterkontrollarchitektur ROS verwendet (siehe Unterabschnitt 2.2.1). Dabei wurde die ROS-Noetic Distribution verwendet. Dabei wird eine Version von ROS-Paketen als ROS-Distribution bezeichnet. Die Pakete haben eine Basis, die verhältnismäßig stabile Software beinhaltet, auf der die Entwicklung zugreift [21]. Die Distribution wurde gewählt, da ein Kobuki Roboter verwendet wurde. Der Roboter läuft mit der ROS-noetic Version relativ stabil. Es werden ROS-Kobuki-Pakete benutzt, um den Roboter zu bedienen und die Sensoren des Roboters abzufragen. Diesbezüglich werden auch ROS-Pakete verwendet wie *rplidar*. *Rplidar* ist ein Paket für 2D Laserscanner wie ein LiDAR. Sie stellt ROS-Nodes zur Verfügung, um Messwerte vom Sensor zu erhalten.

3.2.1 Implementierung Ultraschall-Sensor

Der Kobuki der in dieser Arbeit verwendet wurde, ist ein älteres Modell und somit ist auch der LiDAR Scanner eines der ersten Modelle des Herstellers. Die altersbedingten Verschleisse des Scanners haben dazu geführt, dass viele Messpunkte nicht richtig gemessen werden konnten. So gab es kaum seitliche Messwerte vom Roboter. Um einen Pledge-Algorithmus (siehe Unterabschnitt 2.4.2) zu realisieren, braucht es seitliche Messwerte vom Vehikel, damit die Wand verfolgt werden kann. Diesbezüglich wurde die Maschine mit Ultraschall-Sensoren erweitert. Die verwendeten Ultraschall-Sensoren sind HC-SR04, die im Modellbau und Hobby-Embedded Umfeld ihre Anwendung finden.

Diese Ultraschall-Sensoren wurden am Roboter angebracht und mit dem Raspberry Pi GPIOs verbunden. Die HC-SR04 Sensoren arbeiten mit der Betriebsspannung von 5 V. Da die Raspberry Pi Pins mit 3,3 V arbeiten, musste ein Spannungsteiler bestimmt werden, der die Spannung von 5 V auf 3,3 V herabsetzt (siehe Abbildung 3.5).

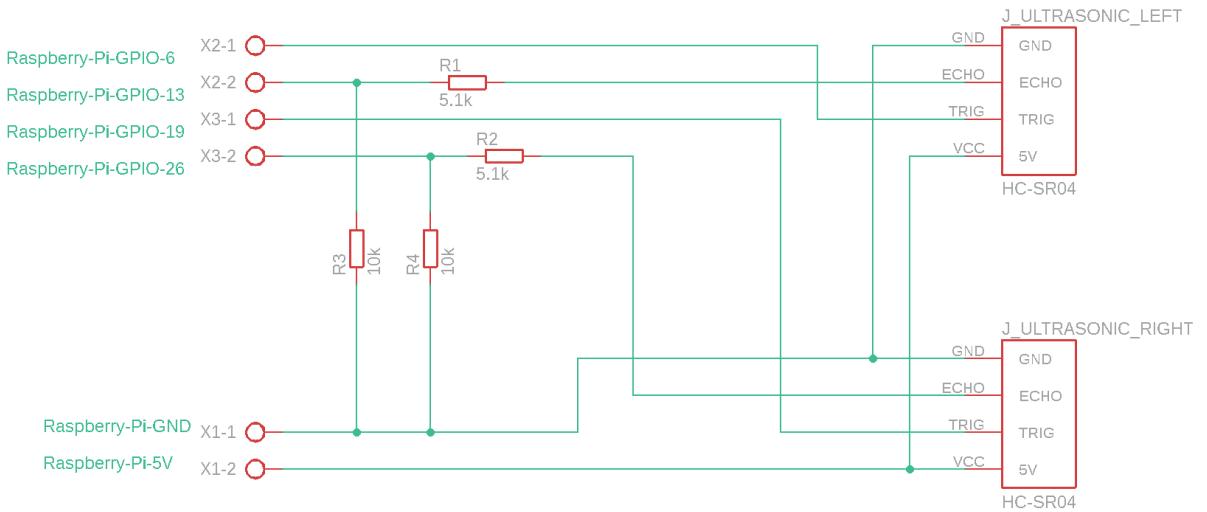


Abbildung 3.5: Schaltplan mit Spannungsteiler für Ultraschallsensor

$$\frac{U_{Pin}}{U_{Echo}} = \frac{R_3}{R_1 + R_3} \quad (3.1)$$

$$\frac{U_{Pin}}{U_{Echo}} = \frac{R_3}{R_1 + R_3} \left| \begin{array}{l} ()^{-1} \end{array} \right. \quad (3.2)$$

$$\frac{U_{Echo}}{U_{Pin}} = \frac{R_1 + R_3}{R_3} = \frac{R_1}{R_3} + \frac{R_3}{R_3} \left| \begin{array}{l} - \left(\frac{R_3}{R_3} = 1 \right) \end{array} \right. \quad (3.3)$$

$$\frac{U_{Echo}}{U_{Pin}} - 1 = \frac{R_1}{R_3} \left| \begin{array}{l} U_{Pin} = 3,3 \text{ V} \& U_{Echo} = 5 \text{ V} \& R_1 = 5,1 \text{ k}\Omega \end{array} \right. \quad (3.4)$$

$$R_3 = \frac{R_1}{\frac{U_{Echo}}{U_{Pin}} - 1} = \frac{5,1 \text{ k}\Omega}{\frac{5 \text{ V}}{3,3 \text{ V}} - 1} = 9,9 \text{ k}\Omega \xrightarrow[\text{Widerstandsreihe E24}]{\quad} R_3 = 10 \text{ k}\Omega \quad (3.5)$$

$$U_{Pin} = U_{Echo} \cdot \frac{R_3}{R_1 + R_3} = 5 \text{ V} \cdot \frac{10 \text{ k}\Omega}{5,1 \text{ k}\Omega + 10 \text{ k}\Omega} \approx 3,3 \text{ V} \quad (3.6)$$

Dabei muss nur das Echo Signal vom Sensor mit den Widerständen aus der Formel gesenkt werden (siehe Gleichung 3.1). Das Trigger Signal kann mit 3,3 V arbeiten da der Low Pegel des Signals bei < 1 V liegt. Der Spannungsteiler wurde für die Arbeit experimentell auf einer Lochrasterplatine gefertigt.

Nachdem die Hardware fertiggestellt wurde, konnte die Software für die Sensoren entwickelt werden. Dazu wurde eine Klasse entwickelt, die ein Ultraschall-Sensor ansteuert

und das Messergebnis liefert. Die Klasse wurde als Nebenläufigkeit implementiert, damit mehrere Instanzen der Klasse erstellt werden können und nicht geblockt sind (siehe Abbildung 3.6). So ist eine zeitgleiche Abfrage der einzelnen Sensoren möglich.

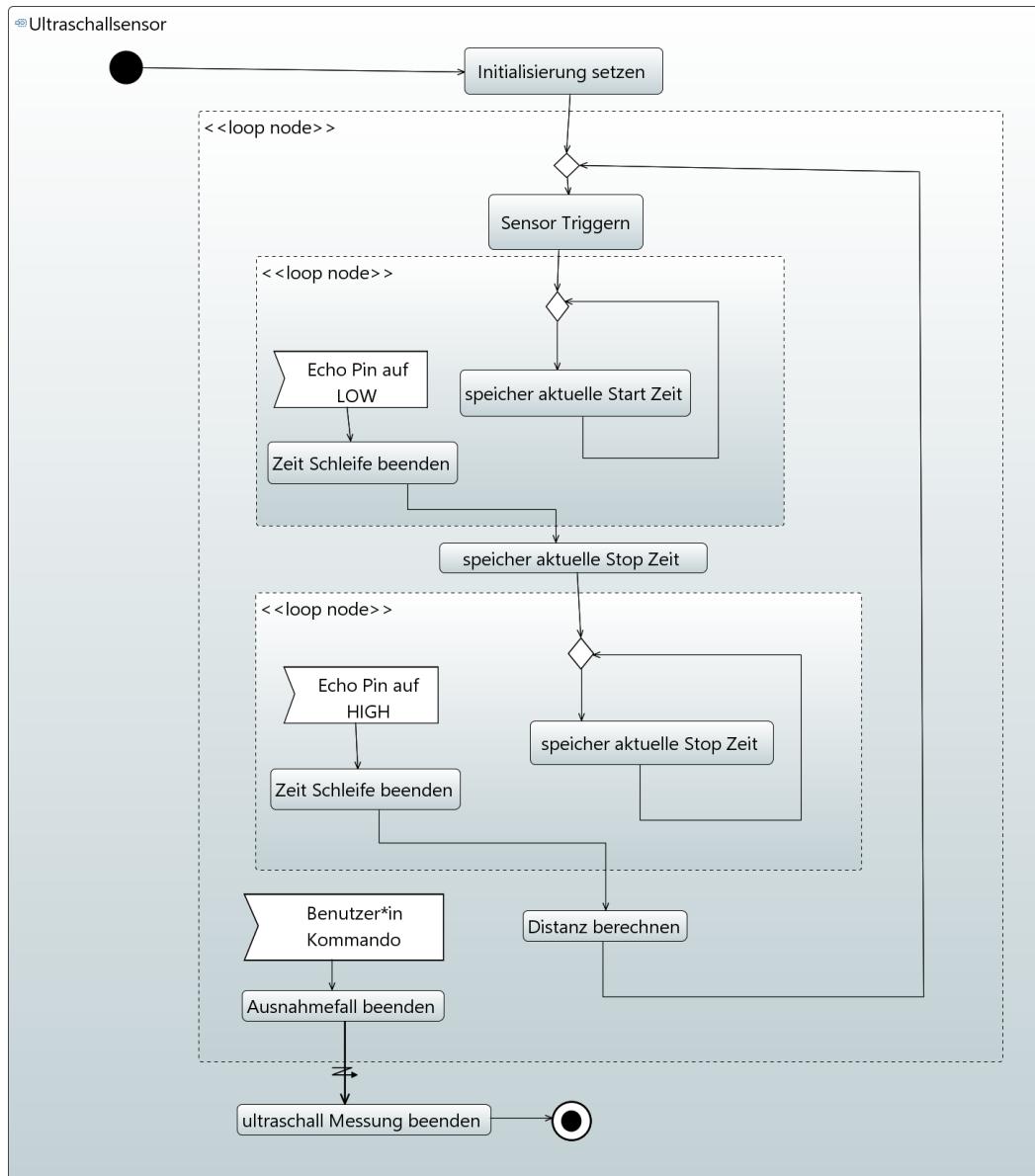


Abbildung 3.6: Aktivitätsdiagramm des Ultraschallsensors

Die zwei Ultraschall-Sensoren werden in einer *ultrasonicROS.py* als Instanzen der Klasse erstellt und in einer *while-schleife* mit ROS veröffentlicht. Dazu werden die Messwerte je

Sensor über einem Topic veröffentlicht. Zudem wurde eine Launch Datei erstellt, um die Sensoren komplett vom System abzukoppeln und als unabhängige Komponenten starten und stoppen zu können.

3.2.2 Implementierung QR-Code Scanner

Der Plege-Algorithmus (siehe Unterabschnitt 2.4.2) findet immer einen Ausweg aus einem Labyrinth, wenn ein Ausgang vorhanden ist. Wo es keinen Ausgang gibt, findet der Algorithmus auch keinen und kommt in eine Endlosschleife. Um diese Endlosschleife durchbrechen zu können, wurde eine Raspberry Pi Kamera installiert. Die Firma Kurt Hüttinger GmbH bietet oft Exponate an, die abgeschlossen sind und somit als eine Einheit vermarktet werden. So könnte die Spielfläche des mobilen Roboters geschlossen sein. Damit der Algorithmus auch in diesen Fällen funktioniert, wurde eine Kamera installiert, um mit QR-Codes das Ziel zu markieren. Des Weiteren kann der QR-Code Scanner auch andere Aufgaben übernehmen wie Positionsbestimmung mithilfe von Landmarken.

Da das Ganze experimentell aufgebaut ist, ist die dominante Seite auch Variable, was dazu führt, dass der Kamerawinkel auch Variable sein muss. Dies wurde durch ein Micro Servomotor SG90 9g gelöst. Die Kamera ist auf dem Servo montiert und je nach dominierender Seite kann der Winkel geschwenkt werden. Der Servomotor kann einen Winkel von 0° bis 180° einnehmen. Der Servomotor ist auch mit dem Raspberry Pi verbunden und wird durch definierte PWM Signale umgestellt. Dazu wurde auch eine Klasse entwickelt, die dann in einem ROS-Knoten untergebracht ist. Im ROS-Knoten wird ein Topic (siehe Absatz 2.2.1.2.2) abonniert, dies ermöglicht zur Laufzeit den Winkel einzustellen. Um diese Funktionalität auch vom System abzukoppeln, wurde dies auch in einem eigenständigen ROS-Paket mit Launch Datei erstellt.

Für den QR-Code Scanner wurde das Quelloffene Softwarepaket ZBar verwendet. Das ZBar Softwarepaket hat für die verschiedenen Programmiersprachen eigenständige Lösungen, so auch für Python mit *pyzbar*. Das Paket ermöglicht aus verschiedenen Quellen, wie Video-Streams, Bilddateien und Sensoren, Barcodetypen zu erkennen und zu decodieren. Dazu wendet das Softwaremodul einen Ansatz zum Erkennen und Decodieren, wie die handelsüblichen Strichcode Scanner und führt eine linearen Scann über die Quelle durch. Es wird jeder Pixel der Quelldatei als unabhängige Probe behandelt [13]. Der ganze QR-Code Scanner ist, wie die zuvor genannten Module, auch als eigenständige ROS-Nodes mit

Launch Datei realisiert, wobei hier der decodierte Code über einen ROS-Topic publiziert wird.

3.2.3 Implementierung Netzwerkkommunikationsmodule

Der Fokus dieser Arbeit ist das API. Die Applikationsschnittstelle soll eine Brücke schaffen zwischen der Anwendung und Roboterkontrollarchitektur, um eine einfache Bedienbarkeit des Roboters zu erhalten. Dabei soll die Schnittstelle über Netzwerk kommunizieren. Dazu wurden die Netzwerkprotokolle wie UDP und TCP auserwählt. Diese Protokolle werden über die IP-Adresse und Port adressiert.

Zu diesem Zweck wurde ein Modul für die Netzwerkkommunikation entwickelt. Das Python Modul ist nebenläufig entwickelt, da auf das Netzwerk Port aktiv gehört werden muss, um auch eine Nachricht erhalten zu können. Die Klasse definiert dazu alle nötigen Einstellungen. Zudem wurde die Klasse Variable hinsichtlich Protokolltyp aufgebaut. Das Netzwerkprotokoll wird über den Konstruktor bestimmt und je nach Art diesbezüglich definiert. So können auch mehrere Verbindungen aufgebaut werden mit verschiedenen Ports und IP-Adressen.

3.2.4 Implementierung Roboter Module

Neben der Schnittstelle ist das Roboter Modul eine wichtige Komponente. Es vereinfacht den Zugriff auf die Funktionalitäten des mobilen Roboters und der Roboterkontrollarchitektur.

Die Klasse ist als ein Paralleler Prozess implementiert auch genannt Thread, um eine kontinuierliche Verbindung zu der Roboterkontrollarchitektur ROS aufrecht zu erhalten. Dazu können wichtige Eigenschaften in einer Schleife durchgeführt werden, wie zum Beispiel ob der Weg des Roboters frei ist (siehe Listing 3.1).

```
1 def run(self):
2     self.running = True
3     self.rate = rospy.Rate(10) #10Hz
4     while (not rospy.is_shutdown() and self.running):
5         # self.printROSINFO("while Loop")
6
7         # ueberpruefung ob Roboter weg frei
8         self.checkCollision()
```

```

9
10     # setzt die Geschwindigkeiten der einzelnen Achsen
11     self.twist_pub.publish(self.currentMoveValue)
12
13     # Triggerung Hintergrund ROS-Prozesse, abwarten Zeit
14     self.rate.sleep()
15     print("End of LIBRobot Running")

```

Listing 3.1: Roboter Module Run Methode

Dabei kann die Klasse auch als ein Python Modul verwendet werden, um Roboter mit Hilfe von ROS zu bedienen. So kann sie nicht nur für die Schnittstelle verwendet werden, sondern auch universeller eingesetzt werden. Das Modul ermöglicht auch durch Umrechnungen Einzelschritt Bewegungen. Dies kann genutzt werden, um Pfade vorzuprogrammieren und sie abzuspielen. Die Funktionalitäten des Roboters sind so in einer Klasse ausgelagert und vereinfachen das API.

3.2.5 Implementierung Pledge-Algorithmus

Es wurde ein eigenständiges Objekt entwickelt für den Pledge-Algorithmus. Dazu wurde die Klasse als ein Zustandsautomat implementiert (siehe Abbildung 3.7). Der Ablauf von Zustandsautomaten ist es, dass Zustände eingenommen werden und nur durch eine Bedingung ein Zustandswechsel stattfindet. So ist ein Zustandsautomat ein geblocktes System. Um aber den Ablauf der API nicht zu gefährden, ist die Klasse parallel ausführbar. So kann jederzeit der Algorithmus beendet und erneut gestartet werden, ohne Auswirkungen auf das API zu haben.

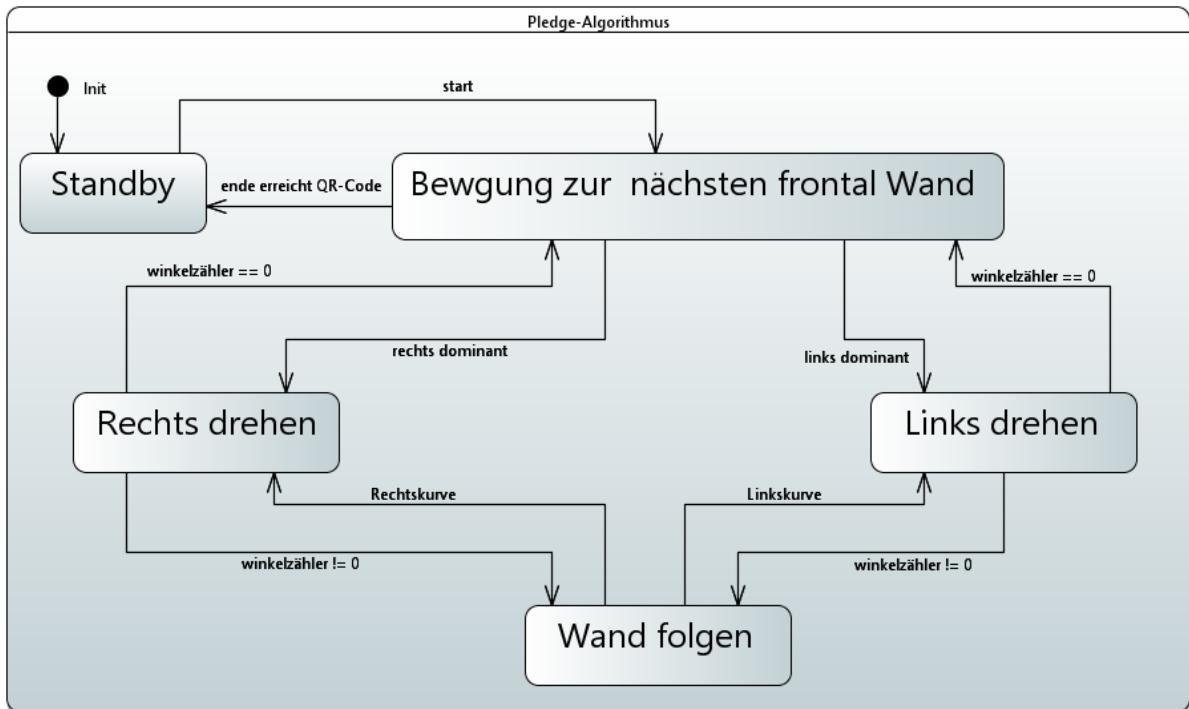


Abbildung 3.7: Zustandsautomat Pledge-Algorithmus

Die einzelnen Zustände sind eigenständige Methoden. So wird beim Zustand “*Bewgung zur nächsten frontalen Wand*” der mobile Roboter mit einer Vorwärts Geschwindigkeit angesteuert und in der Dauerschleife der Sensorwert vom LiDAR verglichen. Sobald der Wert des Sensors eine bestimmte Schwelle unterschreitet, wird dieser Zustand beendet und zum nächsten Zustand gewechselt (siehe Listing 3.2).

```

1 def moveToNextWall(self):
2     self.runningToNextWall = True
3     collidedToFrontWall = CollisionTyp.NOTHING # Kollisionsart
4         reset
5     self.robot.moveRobot(0.05, 0.0, 0.0) # setzt
6         vorwaertsgeschwindigkeit
7     while (self.runningToNextWall and not (collidedToFrontWall ==
8         CollisionTyp.LIDAR_FRONT)):
9         collidedToFrontWall = self.robot.checkCollision() #
10             Kollisionserkennung des Roboter Moduls
11         self.checkQRCode() # Abfrage ob Ziel erreicht wurde
12         if(collidedToFrontWall == CollisionTyp.LIDAR_FRONT): #

```

```

9         frontal eine Wand erreicht?
10        self.runningToNextWall = False
11        break
12        time.sleep(0.001) # kleine Pause um CPU entlastung
13        self.robot.moveRobot(0.0, 0.0, 0.0) # Schleife beendet ->
14          Roboter stoppen
15        self.currentStep = self.getNextStep() # setzt den naechsten
16          Zustand

```

Listing 3.2: Klassenmethode vom Zustand "Bewegung zur nächsten frontalen Wand"

Die Zustände Rechts bzw. Links Drehen sind $|90^\circ|$ Drehungen. Die werden über das Roboter Modul direkt angesteuert. Dies ist möglich durch die Schrittbewegungsfunktionalität des Moduls, wie bereits erwähnt im Unterabschnitt 3.2.4. Zusätzlich zur Drehung wird ein Winkelzähler mitgezählt. Ein weiterer Zustand ist „*Wand folgen*“, dass so ähnlich wie der Zustand „*Bewegung zur nächsten frontalen Wand*“ funktioniert. Bei dieser Klassenmethode werden die Ultraschallsensoren verwendet, um den Abstand zur Wand verfolgen zu können.

3.2.6 Implementierung API

Das ganze System wird durch das API verbunden und bindet somit alle Partikel miteinander. Die Schnittstelle selbst ist als ein Spiel-Algorithmus aufgebaut. Dabei werden alle nötigen Instanzen gebildet und im Anschluss wird eine Dauer-Schleife gestartet. Die Spiel-Schleife, auch oft *Game Loop* genannt, hat drei entscheidende Funktionalitäten. Dazu zählt als Erstes der *Input*, also die Benutzereingaben, diese sind in diesem Fall über Netzwerknachrichten abgedeckt. Als zweite Funktion dient das *Update* dazu, Berechnungen durchzuführen oder Positionen zu bestimmen. Die dritte Funktionalität der Schleife ist *Output* oder *Render*, dass für die Ausgabe von neu errechneten Werten fungiert. Diese drei Funktionen werden nicht nur in der Spieleprogrammierung verwendet, sondern auch in vielen verschiedenen Branchen wie in der Automatisierung oder auch allgemein bei Computern. Diese Methoden werden anders genannt, aber eines wird häufiger in diesem Zusammenhang verwendet, das *EVA-Prinzip*. Das *EVA-Prinzip* steht für Eingabe, Verarbeitung und Ausgabe. So wurde auch die Schnittstelle für diese Arbeit aufgebaut (siehe Listing 3.3). Als Eingabe fungieren die Netzwerknachrichten, der Verarbeitungsschritt ist die Verarbeitung der Nachrichten und als Ausgabe werden Roboter Steuerbefehle versen-

det. Durch diese Methode kann zu jeder Eingabe eine Reaktion folgen.

```
1 def main():
2     api = APIEngine()
3
4     while True:
5         try:
6             # api.input() # wurde ersetzt durch Nezwerk Callback
7             # funktion
8             api.update()
9             api.output()
10            time.sleep(0.001)
11        except KeyboardInterrupt:
12            print("Goodbye from ROS API")
13            sys.exit(0)
14            api.shutdown()
15 if __name__ == '__main__':
16     main()
```

Listing 3.3: Game Loop der Schnittstelle

4 Ergebnisse und Diskussion

Im folgenden Kapitel werden die Ergebnisse aus der Entwicklung zusammengefasst, im Gesamtzusammenhang dieser Arbeit interpretiert und mögliche Ansätze für weitere Forschung aufgezeigt.

Wie in der Zielsetzung Abschnitt 1.2 beschrieben wurde ging es in dieser Arbeit um eine Analyse und eine prototypische Entwicklung eines Systems das dazu dienen soll, eine Schnittstelle zu schaffen, die genutzt werden kann um mobile Roboter einfacher zu steuern ohne Kenntnisse mit Roboterkontrollarchitekturen. Es können verschiedene Szenarien als Benutzeroberflächen genutzt werden mit ein und derselben Anwendungsschnittstelle, so kann spielerisch die Funktionalität von Robotern näher gebracht werden. Die Intelligenz von mobilen Robotern kann je nach Anwendung so übermittelt werden. Dabei ergibt sich die Frage, warum wurde solch eine Arbeit durchgeführt da es schon Roboter Systeme gibt wie zum Beispiel von LEGO die genau dieses Vorhaben ermöglichen. Die Antwort dazu ist einfach, es gibt verschiedene Systeme die genau das Thema dieser Arbeit decken aber leider sind solche Systeme gebunden an Hersteller und es gibt nicht viel Spielraum für Erweiterung und Anwendungsgebiete. So ist es bei LEGO notwendig die eigenen Produkte zu verwenden sowohl Hardware oder Software. In erster Linie soll es als Exponat für die Firma Kurt Hüttinger GmbH dienen und Kindern aber auch erwachsenen Personen die heutzutage mögliche Intelligenz von mobilen Robotern vermitteln. Das API das hier entwickelt wurde kann durch die offene Gestaltung und durch die Standard Netzwerkprotokolle wie TCP und UDP eine beliebige Benutzeroberfläche nutzen (siehe Abbildung 4.1).

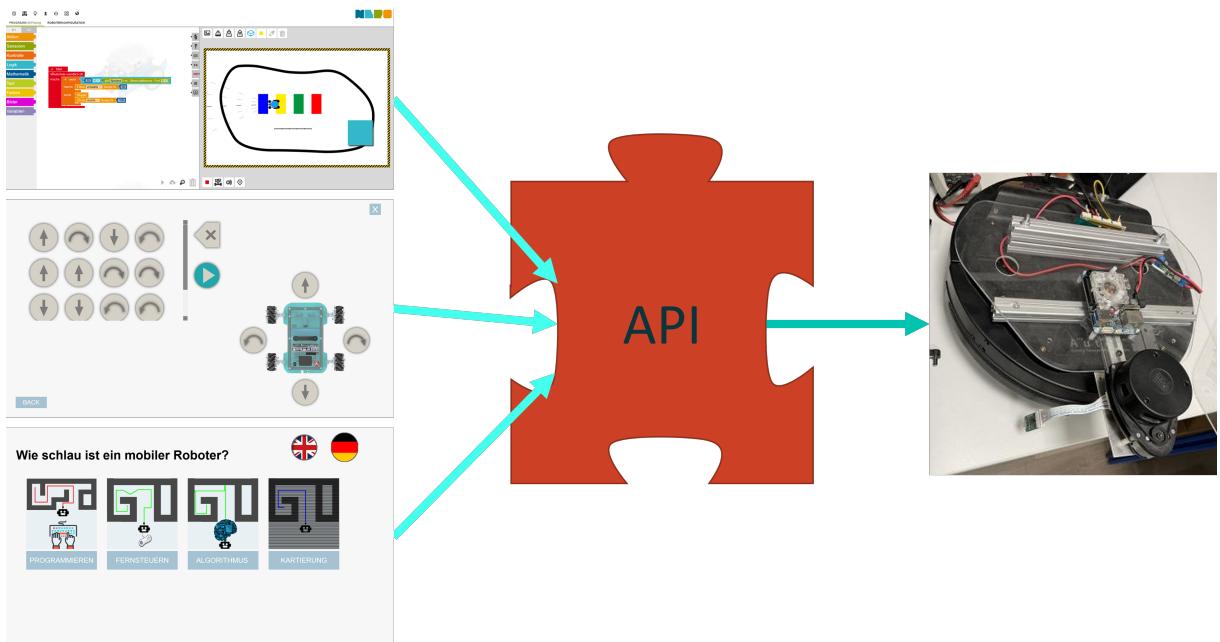


Abbildung 4.1: Vereinfachte Darstellung der Kommunikation mit der API

Des weiteren konnte ein Ansatz für einen Algorithmus entwickelt werden der dazu dient den mobilen Roboter aus einem benutzerdefinierten Hindernisparcours zu befreien. Durch solche intelligenten Funktionalitäten können die Möglichkeiten des Machbaren an die anwendenden Personen transportiert werden. Dabei schafft diese Arbeit eine Basis und einen Grund-Wissenslevel der für weitere Funktionalitäten genutzt werden kann.

Die Arbeit behandelte viele kleine Punkte, die zu diesem Ergebnis führten. So konnte eine Basis geschaffen werden, die zur Weiterentwicklung herangezogen werden kann. Es können weitere Intelligenzen des mobilen Roboters implementiert werden und so spielerisch der Wissensstand der Forschung auf dem Gebiet der mobilen Roboter übermittelt werden. Die weiteren Anknüpfungspunkte dieser Arbeit könnten Kartierung mithilfe von SLAM sein. So kann durch Kartenbildung eine schnellere Navigation zur Ladestation erfolgen. Die Kartenbildung könnte sich beim Plegde-Algorithmus im Hintergrund aufbauen und im Anschluss könnte durch einen anderen Algorithmus wie *A-Star* oder *Dijkstra* der Pfad zurück zum Start gefunden werden.

5 Tests

In diesem Kapitel werden die Tests näher erläutert, die während der Implementierung des Systems durchgeführt wurden.

5.1 Test mit der ROS Umgebung

Die Roboterkontrollarchitektur ROS hat für die Entwicklung verschiedene Tools zur Verfügung. Dazu können grafische und auch Befehlszeile Programme verwendet werden, um das laufende System zu Debuggen.

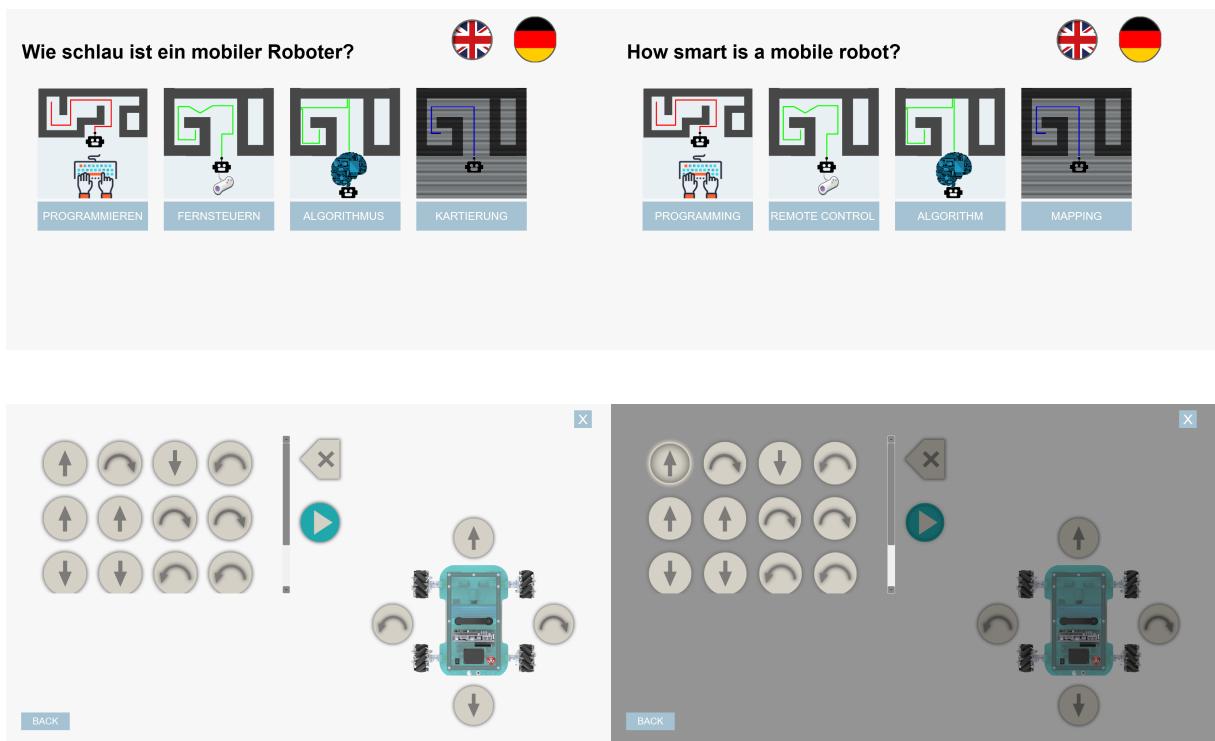
Ein grafisches Programm ist *RVIZ*, das 3-D-Visualisierung von Sensordaten, Robotermodellen und anderen 3-D-Daten ermöglicht. Durch diese Software konnten Sensordaten visualisiert und somit geholfen werden, die Implementierung greifbarer zu machen. So konnte schnell entdeckt werden, dass der LiDAR Scanner bei Messungen seitlich des Roboters Probleme aufzeigt.

Die *rqt_graph* ist ein Tool, um die aktiven Knoten, Topics und deren Verbindung grafisch als Diagramm darzustellen. Damit konnte in der Entwicklung zur Laufzeit aufgezeigt werden, wie die einzelnen Topics miteinander kommunizieren. Bei falscher Schreibweise der Topics führt es nicht in erster Linie zum Fehler, sondern zu neuen Topics die nicht existieren. Um solche Inkorrekttheit zu sehen, kann diese Software verwendet werden. So kann durch die Darstellung der Verbindungen auch auf den ersten Blick gesehen werden, welche Verbindungen nicht stimmen.

Weitere Werkzeuge sind die Befehlszeilen-Tools. Sie ermöglichen zur Laufzeit Informationen zu den einzelnen Knoten, Topics oder Nachrichten auszugeben. Es können Sensor-Topics ausgeben werden, die während der Laufzeit überwacht werden können. Sie bieten aber auch die Möglichkeit, Verbindungen oder die Anzahl davon abzudrucken und somit eine nicht grafische Art von *rqt_graph* darstellt. Diese Kommandozeilen Werkzeuge werden häufig verwendet, da sie eine schnelle und unkomplizierte Lösung sind.

5.2 API Test mit einer grafischen Oberfläche

Um die Schnittstelle zu testen, wurde mithilfe der firmeninternen Software eine grafische Oberfläche entwickelt. Die firmeninterne Software basiert auf *Windows Presentation Foundation*, kurz *WPF* Anwendung und heißt *Interactive Graphic Panel*, kurz genannt *IGP*. Die Software ist so aufgebaut, dass im Hintergrund eine *WPF* Anwendung läuft mit vielen *C#* Funktionalitäten wie TCP oder UDP Sockets, aber die grafische Benutzeroberfläche und die Interaktivität wird mit Javascript programmiert und mithilfe von *XML* werden die Layouts der einzelnen Seiten erstellt. So konnte schnell eine Mensch-Maschinen-Interaktion erstellt werden, die zu Testzwecken verwendet werden konnte (siehe Abbildung 5.2).



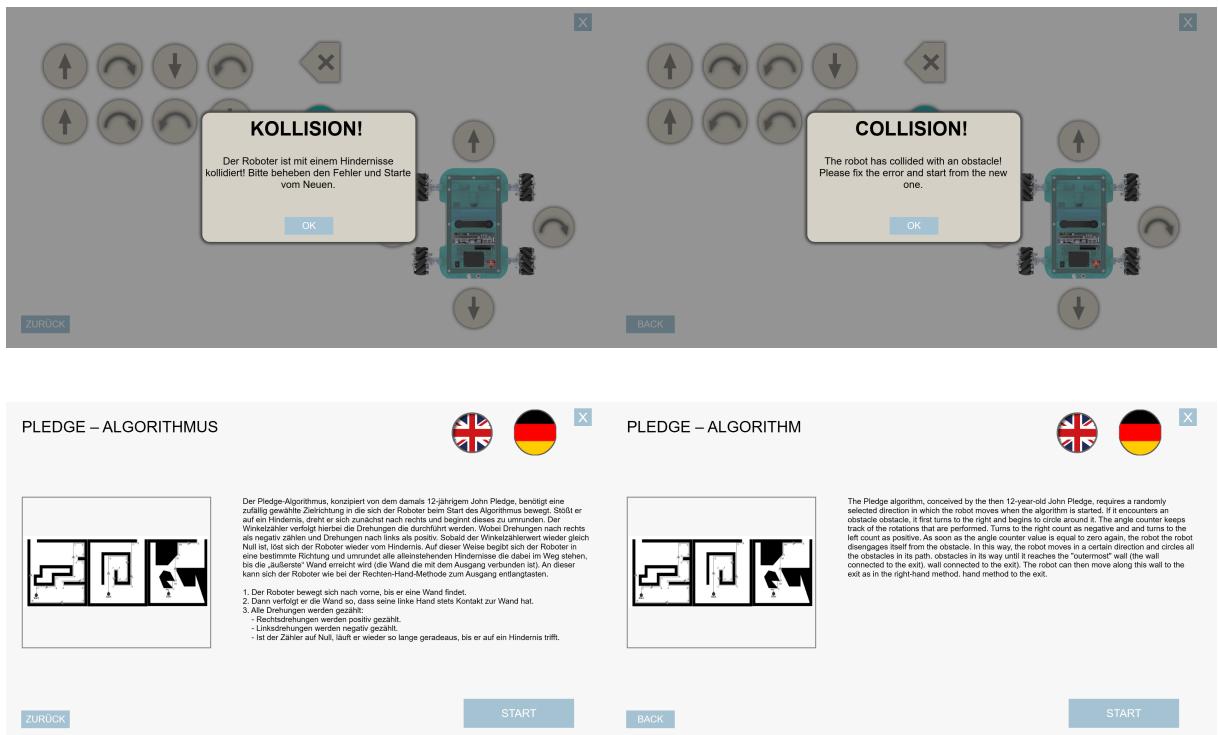


Abbildung 5.2: Test Benutzeroberfläche

6 Zusammenfassung und Ausblick

In dieser Arbeit sollte ein System entwickelt werden, das eine Schnittstelle zwischen der Anwendung und der Roboterkontrollarchitektur implementiert (siehe Abbildung 6.1).

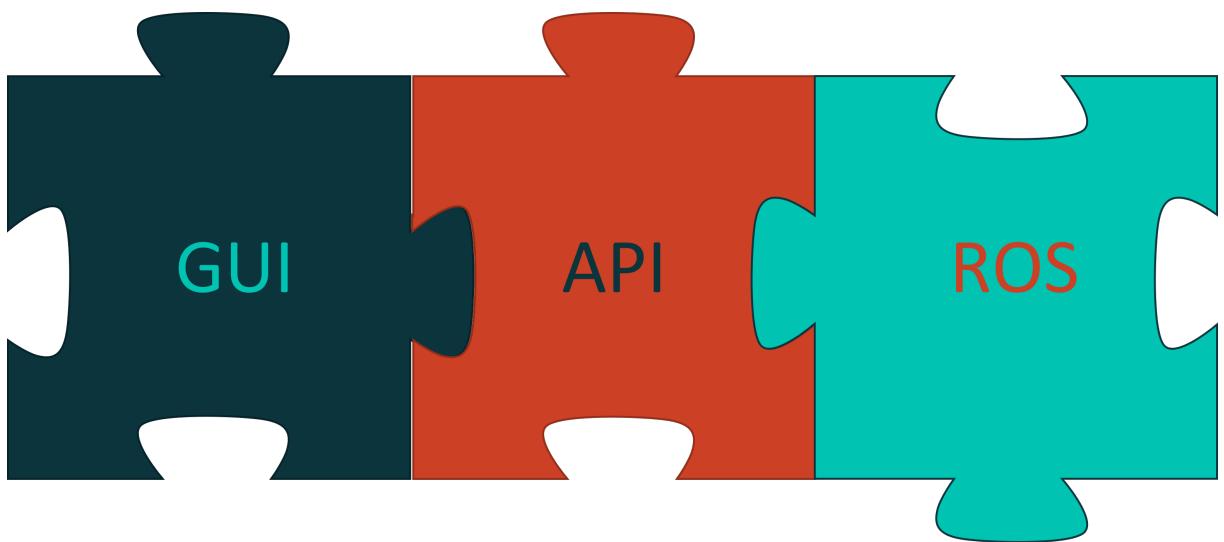


Abbildung 6.1: Darstellung der Schnittstelle

Dabei sollte das System als ein Exponat vermarktet werden. Der Markt dafür sind Bildungseinrichtungen. So soll spielerisch die Intelligenz von mobilen Robotern vermittelt werden. Ein weiterer Einsatzbereich ist die Vermittlung von Informationstechnologie. So kann die Benutzeroberfläche als Lehrmaterial gestaltet werden, um Schüler und Studierenden das Programmieren oder das technische Denken spielerisch zu transportieren. Dies würde somit die alteingesessenen Methoden mit Büchern oder Simulationen ergänzen. Das System in dieser Arbeit verwendet verschiedene Sensoren, die im Laufe des Projektes immer wieder ergänzt wurden, um mehrere Möglichkeiten abzudecken. So konnte mithilfe eines rotierenden Laserscanners der Abstand zu Hindernissen gemessen werden und im Programm die Hindernisse vermieden werden. Dies ermöglichte auch eine relative

Navigation des mobilen Roboters. Des Weiteren wurden Ultraschallsensoren seitlich des Roboters integriert, um dem Pledge- Algorithmus ausreichende Daten zu liefern. Dazu wurde eine separate Implementierung vorgenommen, um den Sensor abzufragen und über ROS die Werte zu veröffentlichen. Die Fähigkeiten von mobilen Robotern sollten mithilfe des Befreiungsalgorithmus im Labyrinth aufgezeigt werden. Dazu wurde der Pledge- Algorithmus verwendet, der auch schon bewiesen hatte, dass er immer einen Ausweg findet, wenn ein Ausweg aus dem Labyrinth existiert. Das Ganze wurde nur prototypisch als Zustandsautomat entwickelt und nur einzelne Teile wurden getestet. Im kompletten zusammenhängenden Ausmaß wurden dazu keine Tests durchgeführt. Die Arbeit entstand in der Zusammenarbeit mit der Firma Kurt Hüttinger GmbH und somit sollte das System als ein komplettes Exponat vermarktet werden. So war bei der Entwicklung klar, dass es keinen wirklichen Ausweg aus dem Labyrinth geben wird. Um trotzdem ein Ziel zu definieren, wurde das System mit einer Kamera erweitert, damit durch QR-Codes das Ziel markiert werden kann. Diesbezüglich wurde ein separates Modul entwickelt, das mithilfe von ZBar, QR-Codes aus Livebilddaten ausliest und über ROS veröffentlicht. Dadurch kann das Ende für den Algorithmus detektiert werden. Das Bindeglied der ganzen einzelnen Teile ist das API. Das API verbindet die ganzen Module miteinander und ermöglicht durch Standard-Netzwerkprotokolle die Bedienung des mobilen Roboters.

Das ganze System wurde an die Firma Kurt Hüttinger GmbH & Co. KG übergeben. Die Schnittstelle kann so eingesetzt werden. Die Firma kann das System mit weiteren Funktionalitäten erweitern, da es bausteinmäßig und objektorientiert aufgebaut ist.

Dadurch dass die Implementierung relativ modular und objektorientiert aufgebaut ist, können leicht weitere Intelligenzen eingebaut werden. So kann als Ausblick genannt werden die Kartierung vom Labyrinth mit SLAM und im Anschluss der Kartierung kann die Navigation mithilfe von weiteren Algorithmen für Pfadfindung erfolgen. Dazu kann der A-Star oder Dijkstra Algorithmus genannt werden, die als Suchalgorithmus dienen. Da auch künstliche Intelligenz immer mehr gefragt und eingesetzt wird, kann dieses System auch mit einer künstlichen Intelligenz erweitert werden, wie zum Beispiel mit einer Kamera, die Objekte und Gegenstände erkennt und je nachdem Entscheidungen trifft. So können auch Rampen eingebaut werden und der mobile Roboter erkennt diese, fährt über die Rampe, wird also diese nicht mehr als Hindernis betrachten. Im Allgemeinen kann das Projekt noch mit vielen Möglichkeiten und Intelligenzen erweitert werden.

Abbildungsverzeichnis

1.1	Einsatz von mobilen Robotern in der Industrie (a) KMR QUANTEC [14] (b) Magnus der Firma Baumüller [2]	1
1.2	Einsatz von mobilen Robotern in privaten Haushalten (a) Roomba Combo Saug- & Wischroboter der Firma iRobot [12] (b) Indego S+ 500 Roboter- Rasenmäher der Firma Bosch [3]	2
2.1	Die intelligenten Funktionen eines Staubsaugroboters der Firma iRobot (a)[10] (b)[11]	5
2.2	Netzstruktur einer Peer-to-Peer-Topologie	6
2.3	Netzstruktur einer Server-Client-Topologie	7
2.4	ROS-Kommunikationsmodes [4]	9
2.5	ROS-Topic Modell (a) Einfacher Publischer und Subscriber Modell (b) Multiple Publischer und Subscriber Modell [4]	10
2.6	Grafische Darstellung von Laufzeitmesssystems	13
2.7	Triangulationsprinzip zur Distanzmessung	15
2.8	Bewegung eines mobilen Roboters	16
2.9	Natürliche form von Gyroskope [25]	17
2.10	Labyrinth	20
2.11	Grundidee Linke- bzw. Rechte-Hand-Regel	21
2.12	Linke- bzw. Rechte-Hand-Regel mit Erweiterung Start Orientierung Spei- cherung	21
2.13	Ablauf Pledge-Algorithmus an zwei Labyrinth Typen	22
3.1	Supersystem Dekomposition dieser Arbeit	26
3.2	Supersystem Komposition dieser Arbeit	27
3.3	Mobiler Roboter System Komposition	27
3.4	Mobiler Roboter Kobuki	28

3.5	Schaltplan mit Spannungsteiler für Ultraschallsensor	30
3.6	Aktivitätsdiagramm des Ultraschallsensors	31
3.7	Zustandsautomat Pledge-Algorithmus	35
4.1	Vereinfachte Darstellung der Kommunikation mit der API	39
5.2	Test Benutzeroberfläche	42
6.1	Darstellung der Schnittstelle	43

Literatur

- [1] Fady Alnajjar u. a.
Roboter in der Bildung: Wie Robotik das Lernen im digitalen Zeitalter bereichern kann.
Hanser eLibrary. München: Carl Hanser Verlag GmbH & Co. KG, 2021.
ISBN: 9783446468023. DOI: 10.3139/9783446468023.
- [2] Baumueller. Autonome Mobile Roboter (AMR). 2021.
URL: <https://www.baumueller.com/de/systeme/amr> (besucht am 30.11.2021).
- [3] Bosch-diy. Indego S+ 500 Roboter-Rasenmäher | Bosch DIY. 2021. URL:
<https://www.bosch-diy.com/de/de/p/indego-s-500-06008b0302-v52902>
(besucht am 30.11.2021).
- [4] Murat Calis.
Roboter mit ROS: Bots konstruieren und mit Open Source programmieren.
1. Auflage. Heidelberg: dpunkt.verlag, 2020. ISBN: 9783960884675.
URL: <https://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=2358671>.
- [5] Deutsches Institut für Normung e.V.
Robotik - Sicherheitsanforderungen - Teil 1: Industrieroboter (ISO/DIS 10218-1.2:2021);
Berlin, 2021. (Besucht am 29.11.2021).
- [6] Marcus Grande. 100 Minuten für Anforderungsmanagement.
Wiesbaden: Springer Fachmedien Wiesbaden, 2014. ISBN: 978-3-658-06434-1.
DOI: 10.1007/978-3-658-06435-8.
- [7] Heinz Arnold. Hilfe bei Katastrophen: Rettungsroboter bis in drei Jahren.
Hrsg. von WEKA FACHMEDIEN GmbH. elektroniknet, 2019.
URL: <https://www.elektroniknet.de/automation/rettungsroboter-bis-in-drei-jahren.171694.html> (besucht am 26.11.2021).

-
- [8] Joachim Hertzberg, Kai Lingemann und Andreas Nüchter. Mobile Roboter: Eine Einführung aus Sicht der Informatik. eXamen.press. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. ISBN: 9783642017261. URL: <http://nbn-resolving.org/urn:nbn:de:bsz:31-epflicht-1624208>.
 - [9] Stefan Hesse. Sensoren Für Die Prozess- und Fabrikautomation: Funktion - Ausführung - Anwendung. 6th ed. Wiesbaden: Springer Fachmedien Wiesbaden GmbH, 2014. ISBN: 9783658058678. URL: <https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=1965742>.
 - [10] iRobot. Bundle aus iRobot® Roomba® s9+ Saugroboter und Braava jet® m6 Wischroboter. 2021. URL: https://shop.irobot.de/irobot-roomba-s9plus-braava-m6/EU_S9plus_m6.html?lang=de_DE (besucht am 30.11.2021).
 - [11] iRobot. iRobot® Roomba® j7 & iRobot Braava m6. 2021. URL: https://shop.irobot.de/irobot-roomba-j7-braava-m6/EMEA_j7_m6g.html?lang=de_DE (besucht am 30.11.2021).
 - [12] iRobot. Roomba Combo Saug- & Wischroboter | iRobot. 2021. URL: <https://www.irobot.de/combo> (besucht am 30.11.2021).
 - [13] Jeff Brown. ZBar bar code reader - About. 2011. URL: <http://zbar.sourceforge.net/about.html> (besucht am 24.12.2021).
 - [14] KUKA. “KMR QUANTEC”. In: (2016). URL: <https://www.kuka.com/de-de/produkte-leistungen/mobilit%C3%A4t/mobile-roboter/kmr-quantec> (besucht am 30.11.2021).
 - [15] Martin Ciupek/IPA. Roboter wischen Corona weg - VDI nachrichten. Hrsg. von VDI. 2021. URL: <https://www.vdi-nachrichten.com/technik/automation/roboter-wischen-corona-weg/> (besucht am 25.11.2021).
 - [16] Harald Nahrstedt. Algorithmen für Ingenieure: Technische Realisierung mit Excel und VBA. 3. Aufl. 2018. SpringerLink Bücher. Wiesbaden: Springer Vieweg, 2018. ISBN: 9783658192990. DOI: 10.1007/978-3-658-19299-0.

-
- [17] Ulrich Nehmzow. Mobile Robotik: Eine praktische Einführung. Springer eBook Collection Computer Science and Engineering. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. ISBN: 9783642559426. DOI: 10.1007/978-3-642-55942-6.
 - [18] Rolf Klein. Algorithmische Geometrie: Grundlagen, Methoden, Anwendungen. SpringerLink Bücher. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. ISBN: 9783540276197. DOI: 10.1007/3-540-27619-X. URL: <http://swbplus.bsz-bw.de/bsz264351746cov.htm>.
 - [19] ROS Community. de/ROS/Concepts - ROS Wiki. 2021. URL: <http://wiki.ros.org/de/ROS/Concepts> (besucht am 02.12.2021).
 - [20] ROS Community. de/ROS/Introduction - ROS Wiki. 2021. URL: <http://wiki.ros.org/de/ROS/Introduction> (besucht am 01.12.2021).
 - [21] ROS Community. Distributions - ROS Wiki. 2021. URL: <http://wiki.ros.org/Distributions> (besucht am 23.12.2021).
 - [22] Sarah Schmitt. Python Kompendium: Ein umfassender Einstieg in die Programmierung mit Python 3. Deggendorf: BMU Media Verlag, 2021. ISBN: 9783966450492.
 - [23] tagesschau de tagesschau. Liveblog: ++ RKI: Zahl der Corona-Ausbrüche in Schulen steigt. 2021. URL: <https://www.tagesschau.de/newsticker/liveblog-coronavirus-donnerstag-271.html> (besucht am 26.11.2021).
 - [24] Berthold Vöcking u. a., Hrsg. Taschenbuch der Algorithmen. eXamen.press. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. ISBN: 9783540763949. URL: <http://nbn-resolving.org/urn:nbn:de:bsz:31-epflicht-1587867>.
 - [25] Wikipedia, Hrsg. Gyroscope. 2021. URL: <https://en.wikipedia.org/w/index.php?title=Gyroscope&oldid=1050926143> (besucht am 08.12.2021).