

Entwicklung eines intelligenten lidarbasierten 3D Navigationssystems für den Unitree Go1

M a s t e r a r b e i t

an der
Hochschule für Angewandte Wissenschaften Hof
Fakultät: Informatik
Studiengang: Master Informatik M.Sc.

vorgelegt bei:
Prof. Dr. Christian Groth
Alfons-Goppel-Platz 1
95028 Hof

vorgelegt von:
Jonas Kemnitzer
Waldstraße 1
95183, Zedtwitz

Hof, 13. September 2023

Any sufficiently advanced technology is indistinguishable from magic.

— Arthur C. Clarke

Danksagung

Im Rahmen der Ausfertigung dieser Arbeit haben mich eine Reihe von Personen unterstützt. Mit großer Dankbarkeit möchte ich meine Anerkennung gegenüber diesen Personen zum Ausdruck bringen.

Ein besonderer Dank gilt meinem Betreuer Prof. Dr. Christian Groth, der mir die Ausarbeitung dieses Themas ermöglichte und jederzeit mit seinem Wissen sowie seiner persönlichen Einschätzung zur Fertigstellung dieser Arbeit beigetragen hat. Seine Anleitung während des Prozesses haben mir maßgeblich dabei geholfen, meine Gedanken und Ideen sinnvoll umzusetzen.

Weiterhin bedanke ich mich bei meinen Freunden und Kommilitonen Noah Lehmann und Ronja Gesell, die mir während des Korrekturprozesses beiseite standen und durch ihre Kenntnisse die Ausarbeitung unterstützten.

Zuletzt gilt mein Dank meiner Familie, die mich während der akademischen Laufbahn in jeglicher Hinsicht unterstützt haben. Ihre Liebe, Geduld und Glaube an mich, haben mir die Kraft gegeben, mich den Herausforderungen dieser Arbeit zu stellen und diese zu meistern.

Inhaltsverzeichnis

1 Einleitung	1
1.1 Hintergrund und Motivation	1
1.2 Zielsetzung der Arbeit	2
1.3 Anwendungsbereich und Einschränkungen	2
1.4 Inhaltlicher Aufbau der Arbeit	3
2 Grundlagen und Stand der Forschung	4
2.1 Grundlagen	4
2.1.1 Robotik	4
2.1.2 Komponenten von Robotern	7
2.1.3 Roboterprogrammierung	9
2.1.4 Künstliche Intelligenz	12
2.1.5 LiDAR-Technologie und Künstliche Intelligenz	15
2.2 Herausforderungen und Möglichkeiten von vierbeinigen Robotern	19
2.3 Verwandte Arbeiten	20
3 Inbetriebnahme des Unitree Go1 Roboters	22
3.1 Hardwarekomponenten	22
3.1.1 Überblick	22
3.1.2 Externe Anschlüsse	24
3.1.3 Interne Betrachtung	25
3.1.4 Überblick RS-Helios-16P LiDAR	29
3.2 Softwarekomponenten	31
3.2.1 Analyse des UnitreeCameraSDK	32
3.2.2 Analyse der Ultraschallsensoren	35
3.2.3 Simulation des Unitree Go1	38
3.2.4 Zusätzliche Ressourcen für den Go1	40

4 Umsetzung	42
4.1 Beschreibung des Versuchsaufbaus	42
4.2 Versuchsdurchführung	44
5 Ergebnisanalyse	51
5.1 Auswertung der Szenarien	51
5.1.1 Betrachtung Indoor-Szenario	52
5.1.2 Betrachtung Outdoor-Szenario	54
5.1.3 Haupterkenntnisse	57
5.2 Einschränkungen des Systems	57
6 Fazit	59
6.1 Zusammenfassung	59
6.2 Zukünftige Forschungsansätze	61
A Anhang: Abbildungen	63
B Anhang: Code	71
C Anhang: Tabellen	78

Abbildungsverzeichnis

2.1	Darstellung eines ROS Graphen	10
2.2	Zeitliche Entwicklung der KI betrachtet von 1930 bis 2000 mit relevanten Meilensteinen	12
3.1	Übersicht über die Komponenten des <i>Go1</i> Roboters	24
3.2	Anschlüsse und Ports auf der Oberfläche des <i>Go1</i> Roboters	25
3.3	Draufsicht über den internen Aufbau des <i>Go1</i> Roboters und Zuordnung darin befindlicher Komponenten	26
3.4	Draufsicht über den internen Aufbau des <i>Go1</i> Kopf und Verordnung darin befindlicher Komponenten	27
3.5	Architekturdiagramm des <i>Go1</i>	27
3.6	Architekturdiagramm des <i>MIT Cheetahs</i>	28
3.7	Ausschnitt aus dem Programm <i>RSView</i>	31
3.8	Aufteilung der UDP-Pakete des <i>RS-Helios-16P</i> Sensors	31
3.9	Ausschnitt aus dem Programm <i>RViz</i> zur Visualisierung der Gelenkwinkel des <i>Go1</i>	38
3.10	Ausschnitt aus dem Programm <i>Gazebo</i> zur Visualisierung des <i>Go1</i>	39
4.1	Montage des LiDAR Sensors auf dem <i>Go1</i>	43
4.2	Diagramm der <i>hdl_graph_slam Nodes</i>	46
4.3	<i>Nodes</i> und <i>Topics</i> des <i>rqt</i> Graphen für den <i>hdl_graph_slam</i> Algorithmus	49
5.1	Foto mit dazugehöriger ermittelter 3D-Punktwolkenumgebung des Foyers und SLAM Graphen des IISYS	52
5.2	Verschiedene Ansichten der gemappten Umgebung des <i>Indoor</i> -Szenarios (<i>FAST_GICP</i>)	53
5.3	Fehlerhafte Umgebungskarte erzeugt durch <i>HDT_OMP</i>	54

5.4	Verschiedene Ansichten der gemappten Umgebung des <i>Outdoor</i> -Szenario (<i>FAST_GICP</i>)	55
5.5	Vergleich der 3D-Karte für Steigungen und Treppen ohne <i>Floor-Detection</i> (links) und mit (rechts)	56
5.6	Darstellung der fehlerhaften 3D-Karte aufgrund geneigter Referenzmessungen	56
A.1	Ausgabe der installierten ROS Packages auf dem <i>NVIDIA Xavier NX</i>	63
A.2	Ausgabe der installierten ROS Packages auf dem <i>Raspberry Pi</i>	64
A.3	Ausschnitte aus der Web-Applikation und Anzeige der Personenerkennungssoftware	65
A.4	Ausschnitte aus der Web-Applikation und Anzeige der verschiedenen Kameras	66
A.5	Ausschnitte aus der <i>iOS</i> App	66
A.6	Ausgabe der Fehlermeldung bei dem Versuch die Sensordaten zu speichern.	67
A.7	Ausgabe der Fehlermeldung bei Verwendung des <i>CameraSDK</i>	67
A.8	Screenshot aus den Konfigurationseinstellungen des LiDAR	68
A.9	Ausgabe der Fehlermeldung bei der Konvertierung der LiDAR Formate .	68
A.10	Darstellung des <i>Unimate</i> -Roboterarm	69
A.11	Darstellung des <i>Spot</i> -Roboter	70

Tabellenverzeichnis

2.1	Tabelle über die Einteilung der Roboterarten nach verschiedenen Merkmalen nach [Mai16]	6
3.1	Übersicht über Bewegungsumfang der Gelenke	23
3.2	Hauptkomponenten und deren Aufgaben im <i>Go1</i>	26
3.3	Ermittelte IP-Adresse der Roboter Hardware	29
3.4	Tabelle über die technische Spezifikationen des <i>RS-Helios-16P</i> Sensors	30
4.1	Kenndaten der zwei Messszenarien	43
4.2	Beschreibung der Aufzeichnungsmöglichkeiten der Messpunkte	44
4.3	Kenndaten der erzeugten <i>bag</i> Dateien	49
C.1	Kenndaten des <i>Raspberry Pi</i> an Bord des <i>Go1</i>	78
C.2	Kenndaten des <i>NVIDIA Xavier NX</i> an Bord des <i>Go1</i>	78
C.3	Kenndaten der <i>NVIDIA Jetson Nanos</i> an Bord des <i>Go1</i>	79

Abkürzungsverzeichnis

KI	Künstliche Intelligenz
ML.....	Machine Learning
DL.....	Deep Learning
ANN.....	Artificial Neural Networks
IMU.....	Inertial Measurement Unit
CPU.....	Central Processing Unit
ROS.....	Robot Operating System
DARPA.....	Defense Advanced Research Projects Agency
DDS.....	Data Distribution Service
SPOF.....	Single Point of Failure
LiDAR.....	Light Detection and Ranging
SLAM.....	Simultaneous Localization and Mapping
RRT.....	Rapidly-exploring Random Tree
DOF.....	Degrees of Freedom
PMSM.....	Permanent-Magnet Synchronous Motor
FOC.....	Field Oriented Control
AVE.....	Absolute Value Encoder
MCU.....	Main Control Unit
RPM.....	Rounds Per Minute
OS.....	Operating System
BMS.....	Batterie Management System
SDK.....	Software Development Kit
WWAN.....	Wireless Wide Area Network
PCD.....	Point Cloud Data
UDP.....	User Datagram Protocol
MSOP.....	Main Data Stream Output Protocol
DIFOP.....	Device Information Output Protocol
I/O.....	Input and Output
GUI.....	Graphical User Interface

LCM	Lightweight Communications and Marshalling
URDF	Unified Robot Description Format
NDT	Normal Distribution Transform
VGICP	Voxelized Generalized Iterative Closest Point Algorithm
ICP	Iterative Closest Point
SVD	Singular Value Decomposition
RANSAC	Random Sample Consensus
XML	Extensible Markup Language
NaN	Not a Number
APE	Absolute Pose Error
RPE	Relative Pose Error

1 Einleitung

In diesem Kapitel werden die Hintergründe und Zielsetzung der Arbeit definiert. Dazu gehören der Anwendungsbereich, welcher die Rahmenbedingungen festlegt, sowie weitere Einschränkungen hinsichtlich des Umfangs der Ausarbeitung. Ein Umriss der inhaltlichen Struktur rundet das einleitende Kapitel ab.

1.1 Hintergrund und Motivation

Der Einsatz ebenso wie die Implementierung von Mensch-Roboter Systemen werden in der heutigen Gesellschaft immer relevanter, sei es durch den Gebrauch in privaten Haushalten mithilfe von Saugrobotern oder Hochrisiko-Umgebungen wie Industrieanlagen oder Katastrophenfällen, um dort bei der Rettung von Menschen zu unterstützen.

Light Detection and Ranging (LiDAR) Sensoren können dabei helfen, indem sie eine 3D-Karte des Umfeld mithilfe von sogenannten Punktwolken aufbauen. Das trägt dazu bei, dass Roboter ein besseres Verständnis über ihr Umfeld erlangen können. Vor allem in Betrachtung sicherheitsrelevanter Aspekte, ist eine zuverlässige Erkennung von Objekten und Menschen zur Vermeidung von Schäden unerlässlich. Im Gegensatz zu herkömmlichen Sensoren haben LiDAR Sensoren einen Vorteil gegen verschiedenste äußere Einflüsse wie Lichtverhältnisse, was sie – neben ihrem hohen Detailgrad sowie der Geschwindigkeit¹ – äußerst geeignet als Teil der Entwicklung von Navigationssystemen für Roboter macht.

Durch das Aufkommen autonomer Fahrzeuge und intelligenter Stadtanwendungen wird die Bedeutung derartiger Entwicklungen weiter bestärkt. So kann in dichten städtischen Umgebungen der Einsatz von lidarbasierten Robotersystemen eine erhöhte Sicherheit für Fußgänger sowie eine effizientere Verkehrs- und Stadtplanung ermöglichen. Dies spielt

¹Gemeint ist damit, dass es möglich ist, große Gebiete in relativ kurzer Zeit zu scannen und eine große Menge an Daten zu erhalten.

auch eine Rolle hinsichtlich von Datenschutzbelangen, da LiDAR-Sensoren in der Lage sind, räumliche 3D-Informationen ohne detaillierte Bilder von Personen aufzunehmen.

1.2 Zielsetzung der Arbeit

Im Vordergrund dieser Arbeit steht die Untersuchung zur Machbarkeit eines Navigationssystems mithilfe von vierbeinigen Robotern. Speziell soll ein *Unitree Go1 Edu* quadruped² Roboter in das Ökosystem der Hochschule für Angewandte Wissenschaften Hof integriert werden. Dies geschieht unter Zuhilfenahme des auf dem Rücken montierten LiDAR Sensor sowie gängigen Künstliche Intelligenz (KI)-Verfahren zur Lokalisierung und Erkennung der Umgebung im unmittelbaren Umfeld des Roboters. Im Zentrum steht dabei die Entwicklung eines funktionsfähigen Demonstrator mit dem anschließend einige Tests unter realen Bedingungen durchgeführt werden können.

1.3 Anwendungsbereich und Einschränkungen

Wie zuvor erwähnt, besteht eine Einschränkung bereits in der Wahl des Standorts. Als Umgebung für den Versuch wurde sich für das Hochschulgebäude und umliegende Gelände entschieden. Es bietet komplexe Szenarien wie Treppen und schmale Gänge aber auch größere Flächen um die Navigations- sowie Lokalisierungssoftware des Roboters zu testen.

Als Hardware stellt die Hochschule für Angewandte Wissenschaften Hof zwei Roboter vom Typ *Unitree Go1* zur Verfügung. Diese liefern eine Bandbreite an Funktionen, Bibliotheken und Sensoren mit, um den Prototypen zu entwickeln. Näheres zum Aufbau des Roboters in Kapitel 3.

²Quadruped ist ein Wort für die Form der Fortbewegung, bei der vier Gliedmaßen zum Tragen von Gewicht und zur Fortbewegung eingesetzt werden.

1.4 Inhaltlicher Aufbau der Arbeit

Grundlegend gliedert sich die Arbeit in sechs Kapitel. Zu Beginn werden mithilfe der Einleitung Hintergrund, Zielsetzung und Anwendungsbereich dieser Arbeit beschrieben und eingegrenzt.

Anschließend folgt ein Kapitel über die notwendigen Grundlagen der Robotik und KI, um Begriffe zu bezugnehmenden Technologien einzuführen und zu erklären. Nachdem ein Überblick über die zugrunde liegenden Technologien geschaffen wurde, werden Herausforderungen und Möglichkeiten des Vorhabens einen vierbeinigen Roboter auf dem Universitätsgelände einzusetzen, aufgeführt. Ein Bezug zu bereits vorhandenen Arbeiten, die für das Projekt relevante Problemstellungen betrachten, runden diesen Teil ab. Kapitel 3 beschäftigt sich mit der Inbetriebnahme des Roboters. Hier wird eine Übersicht über die Hardware- und Softwarekomponenten des *Unitree Go1* gegeben. Ergänzend dazu liegt das Hauptaugenmerk des darauf folgenden Kapitels auf der konkreten Umsetzung. Dazu werden die einzelnen Schritte für die Implementierung der Navigations- und Lokalisierungsalgorithmen aufgeführt.

Abschließend bilden die letzten beiden Kapitel eine Ergebnisanalyse und ein Fazit in der die zuvor definierten Versuche ausgewertet werden. Im Zuge dessen wird auf Einschränkungen des Gesamtsystems eingegangen. Ein kurzer Ausblick auf zukünftige Forschungsmöglichkeiten sowie die Zusammenfassung der Untersuchungsergebnisse bilden den Abschluss der Arbeit.

2 Grundlagen und Stand der Forschung

Das Ziel dieses Kapitels besteht darin, grundlegende Kenntnisse über Robotik, wie beispielsweise der Programmierung, sowie Ansätze im Bereich der KI, zu vermitteln. Darauf aufbauend wird ein Einstieg in die Thematik *Simultaneous Localization and Mapping (SLAM)* gegeben. Weiterhin werden Aspekte beleuchtet, welche die Möglichkeiten und Herausforderungen des Projektvorhabens beschreiben. Abschließend wird auf relevante Arbeiten im Bereich des Einsatzes von vierbeinigen Robotern eingegangen.

2.1 Grundlagen

Das Fachgebiet der Robotik ist aufgrund der Kombination mehrerer Teilgebiete als interdisziplinäre Wissenschaft anzusehen. So beinhaltet das Fachgebiet u. a. weitere Wissensbereiche wie Regelungs- und Steuerungstechnik, Informatik, Maschinenbau, Mathematik, Elektrotechnik, Sensorik sowie Teilbereiche der KI [Spa19].

2.1.1 Robotik

Die ersten modernen Roboter entstanden zu Beginn des 20. Jahrhundert [Fau22]. Der erste moderne programmierbare Roboter³ wurde 1954 von dem amerikanischen Erfinder George Devol patentiert. Er und Joseph Engelberger brachten in den 1960er Jahren den *Unimate* (siehe Anhang A.10) auf den Markt. Den ersten hydraulisch betriebenen Industrieroboterarm, der auf einer Magnettrommel gespeicherte Befehle ausführen konnte. Nur ein Jahr später wurde diese Erfindung für gefährliche Schweißarbeiten in einer Montagelinie von General Motors eingesetzt und ebnete den Weg für die vollautomatischen

³Eigentlich handelte es sich hier um einen Roboterarm/Manipulator

Produktionslinien, die heute aus vielen Branchen bekannt sind [ENG23].

Die ersten mobilen Roboter die sich in einer bestimmten Umgebung bewegen konnten, wenn auch ferngesteuert, wurden jedoch erst einige Jahre später – 1968 – entwickelt. Zwischen 1966 und 1972 entwickelte das Stanford Research Institute in den Vereinigten Staaten den Roboter *Shakey*, der nach einfacher Programmierung einen Befehl ausführte und somit als erster mobiler autonomer Roboter der Welt gilt [Fra19].

Die Entwicklung von vierbeinigen Robotern nahm mit der *TITAN* Serie von dem japanischen Entwickler Shigeo Hiros ihren größten Entwicklungssprung [Sha21]. Diese vierbeinigen Roboter waren die ersten der Welt, welche mit fußmontierten Drucksensoren und einem Lagesensor⁴ ausgestattet waren. Später, im Jahr 1986, wurde *TITAN IV* mit zusätzlichen Funktionen entwickelt, wie zum Beispiel der Umwandlung des Kriechens in einen Trab durch schrittweise Erhöhung der Geschwindigkeit. *TITAN V* und *TITAN VI* wurden für Zwecke wie Gewichtsmechanik, dynamisches Gehen und weiteren Eigenschaften entwickelt. Die Reihe setzt sich fort bis zur neusten Version dem *TITAN-XIII* [Kit+16].

Nicht zu vernachlässigen ist der Beitrag des Massachusetts Institute of Technology sowie Boston Dynamics im Bezug auf die Entwicklung vierbeiniger Roboter. Der *Little Dog* [Sta10], ein kleiner vierbeiniger Roboter, der 2010 auf den Markt kam, hatte vier Beine, die jeweils von drei Elektromotoren angetrieben wurden. Dieser Roboter wurde für die Defense Advanced Research Projects Agency (DARPA) entwickelt. Der Roboter war in der Lage zu klettern und sich dynamisch fortzubewegen. Im Jahr 2013 entwickelte das Massachusetts Institute of Technology einen hocheffizienten vierbeinigen Roboter namens *MIT Cheetah* [Ble+18]. Dieser 70 Pfund schwere Roboter verbrauchte nur sehr wenig Energie, da er bis zu anderthalb Stunden lang ununterbrochen mit einer Geschwindigkeit von 5 km/h trabte.

Eine Heutzutage bekannte vierbeinige Roboter ist ebenfalls von Boston Dynamics entwickelt. Die Baureihe trägt den Namen *Spot* (siehe Abbildung A.11). Sie wurde am 23. März 2016 vorgestellt. Im März 2021 wurde der *Spot* von SpaceX eingesetzt, um potenziell gefährliche Stellen rund um die Absturzstelle von Raketen zu inspizieren [Dyn23] [Ind21].

Die Zukunft vierbeiniger Laufroboter ist vielversprechend. Im Gegensatz zu ihren Pendants auf Rädern, Raupen oder in der Luft können sich Roboter mit Beinen hervorragend in unwegsamem Gelände und unstrukturierten Umgebungen fortbewegen. Durch ihre Fähigkeit zu klettern und kriechen eignen sie sich hervorragend für bestimmte An-

⁴ein Instrument zur Bestimmung der Winkelabweichungen der Achsen

wendungen und Einsatzbereiche (z. B. für Förderung der Barrierefreiheit), in denen Roboter auf Rädern nicht funktionieren würden. Sie können weiterhin mit einer Vielzahl von Sensoren ausgestattet werden, z. B. mit Kameras, Wärmebildkameras, LiDAR oder Gassensoren zur Messung der Luftverschmutzung. Diese Sensoren erleichtern die autonome Navigation und ermöglichen eine intelligente Lokalisierung und Objekterkennung. Weiterhin eröffnen sie ein breites Spektrum an Einsatzmöglichkeiten, wie die Inspektion von Maschinenanlagen oder der Rettung von Menschen aus Krisengebieten.

Roboterarten

Es gibt unterschiedliche Merkmale und Möglichkeiten Roboter in Kategorien einzufügen. Dennoch ist eine genaue Klassifizierung von Robotern oftmals unpräzise. Durch die beliebige Kombination von Eigenschaften oder Antriebsarten sind verschiedenste Kombinationen möglich. Die bekannteste Art der Roboter sind die der industriellen Automatisierung. Es gibt jedoch eine Reihe weiterer Arten von Robotern. Tabelle 2.1 zeigt eine Unterteilung unter Betrachtung verschiedener Merkmale.

nach Bauart	nach Verwendung	Sonstige
stationäre Roboter		
Industrieroboter	Service-Roboter	
Gelenkarm-Roboter	Haushaltshelfer	<i>BEAM</i> (Biologie, Elektronik, Ästhetik und Mechanik)
<i>SCARA</i> -Roboter		
Delta-Roboter		
autonome mobile Roboter	Erkundungsroboter	
Arbeitsroboter	Spielzeugroboter	Schwarmroboter
Laufroboter	Transportroboter	Nanoroboter
rollende Roboter	Militärroboter	bionische Roboter
fliegende Roboter	Rettungsroboter	
kognitive Roboter	medizinische Roboter	
humanoide Roboter (Androiden)		

Tabelle 2.1: Tabelle über die Einteilung der Roboterarten nach verschiedenen Merkmalen nach [Mai16]

2.1.2 Komponenten von Robotern

Moderne Robotersysteme können hinsichtlich ihrer Ausstattung und ihrem Einsatzzweck variieren. Grundlegend lassen sich jedoch fünf Kernkomponenten identifizieren, die folgend betrachtet werden sollen [Par20].

Aktoren

Ein Aktor ist der Teil eines Roboter, der dazu beiträgt, physikalische Bewegungen auszuführen, indem er Energie in mechanische Kraft umwandelt⁵. Einfach ausgedrückt, ist es die Komponente, welche Bewegung ermöglicht. Zu den bekanntesten Arten von Aktoren gehören elektrische, hydraulische und pneumatische Aktoren [Spa19].

Elektrische Aktoren verwenden eine externe Energiequelle wie eine Batterie um Bewegungsenergie zu erzeugen. Hydraulischer Aktoren verwenden eine nicht komprimierbare Flüssigkeit um mithilfe eines Kolben oder Drehschiebers Bewegung zu erzeugen. Zuletzt nutzen pneumatische Aktoren Druckluft.

Die verschiedenen Bauarten besitzen Vor- und Nachteile. Während hydraulische Systeme meist leistungsfähiger und präziser sind, erfolgt die Bewegung langsamer. Im Gegensatz dazu sind pneumatische Systeme schneller jedoch unpräzise⁶. Elektrische Antriebe ermöglichen eine äußerst präzise Steuerung und Positionierung. Ein weiteres Merkmal für elektrische Aktoren sind niedrige Betriebskosten [Ph.20].

Sensoren

Sensoren helfen Robotern ihre Umgebung wahrzunehmen oder Messwerte über interne Bauteile oder externe Umgebungsfaktoren abzugreifen. Ähnlich wie bei den Aktoren gibt es auch hier eine Reihe unterschiedlicher Typen, die sich je nach Aufgabe unterscheiden [ROB19]. Die hier erwähnten Sensoren werden später in Kapitel 3 anhand des *Go1* Roboters zugeordnet.

⁵Die weitere Unterscheidung zwischen translatorischen und rotatorischen Aktoren wird in dieser Arbeit nicht betrachtet, da diese von untergeordneter Bedeutung sind

⁶Beispielsweise kann sich die Position durch Änderung der Last ebenfalls ändern.

Umgebungssensoren

Hierzu gehören z. B. Lichtsensoren, Schallsensoren oder Temperatursensoren. Bei Lichtsensoren kommen überwiegend Photoresistoren sowie Photovoltaik-Zellen zum Einsatz. Mithilfe von Schallsensoren werden Schallwellen in elektrische Signale umgewandelt.

Näherungssensoren

Die meisten Näherungssensoren dienen als Abstandssensoren. Sie werden auch allgemein als Entfernungssensoren bezeichnet. Beispiele hierfür sind Infrarot- oder Ultraschallsensoren.

Lagesensoren

Lagesensoren dienen dazu, dem Roboter Informationen über seine Lage in der Umgebung oder die Lage der verbauten Komponenten, beispielsweise Gelenke, zu liefern. Eine Kombination von Geschwindigkeit, Orientierung und Schwerkraft liefert z. B. eine Inertial Measurement Unit (IMU).

Zusammengefasst liefern Sensoren wichtige Daten für den Roboter. Zu diesen Daten gehören u. a.: Position, Größe, Ausrichtung, Geschwindigkeit, Entfernung, Temperatur, Gewicht, Kraft und viele andere Faktoren, die Robotern helfen, ihre Umgebung wahrzunehmen und Aufgaben auszuführen. Mit dem steigenden Bedarf an Automatisierung und computergesteuerten Maschinen werden Sensoren immer wichtiger.

Kontrollsystem

Hierzu gehören u. a. die Central Processing Unit (CPU) sowie jegliches Programm, das den Roboter steuert. In der Anatomie von Robotern funktionieren CPUs ähnlich wie das menschliche Gehirn. Die Daten werden mithilfe von Sensoren eingespeist. In Zusammenarbeit mit der CPU sowie weiterer Hardware und einer Software-Komponente werden die Informationen verarbeitet und an die Steuerungssysteme als Befehle gesendet. Möglichkeiten Roboter zu programmieren behandelt Abschnitt 2.1.3.

Endeffektoren

Eine weiteres Bauteil, das die meisten Robotern gemeinsam haben, sind Endeffektoren. Der Begriff bezieht sich auf die Werkzeuge an Bord des Roboters, die Teile welche Arbeit ausführen und mit der Umgebung oder einem Werkstück interagieren. Moderne medizinische Roboter verwenden z. B. kleine Skalpelle und Kameras um filigrane Operationen zu ermöglichen während große Industrieroboter meist mit Schweißbrennern, Schraubdrehern oder Farbspritzgeräten ausgestattet sind.

Stromversorgung

Stromversorgungen können verschiedene Formen annehmen. Stationäre Roboter, z. B. in Fabriken, werden wie die meisten Geräte direkt mit Strom versorgt. Mobile Roboter verfügen in der Regel über Hochleistungsbatterien, während Robotersonden und Satelliten meist mit Solarzellen ausgestattet sind, um Energie aus der Sonne zu gewinnen. Nachdem ein Überblick über die gängigsten Komponenten gegeben wurde beschäftigt sich der folgende Abschnitt mit der Programmierung von Robotern.

2.1.3 Roboterprogrammierung

Unter Roboterprogrammierung wird die Erstellung von Anweisungen, die es einem Roboter ermöglichen, eine bestimmte Aufgabe zu erfüllen, verstanden. Diese Anweisungen werden i. d. R. in einer Programmiersprache verfasst, die vom Steuerungssystem des Roboters verstanden wird. Dabei kann das gewünschte Verhalten des Roboters definiert, in kleinere, leichter zu bewältigende Aufgaben zerlegt und diese dann mithilfe von z. B. Code umgesetzt werden. Heutzutage gibt es unterschiedliche Ansätze Roboter zu programmieren. Oft genannte Verfahren sind beispielsweise *Online*-, *Offlineprogrammierung* oder *Teach-Pendant* [Mai16].

Hinsichtlich der Programmierung von Robotern hat sich vor allem das *Robot Operating System (ROS)* als Standard etabliert. Es erleichtert die Entwicklung komplexerer Szenarien und bietet eine Reihe an Funktionen. Die neuste Version *ROS2* erweitert und verbessert dabei den Vorgänger in vielen Aspekten [JC21] [MF13]. Ziel von ROS ist es einen Standard für die Programmierung von Robotern zu schaffen und gleichzeitig Standard-Softwarekomponenten anzubieten, die leicht in benutzerdefinierte Roboteran-

wendungen integriert werden können [Rob21].

Entgegengesetzt dem Name ist ROS kein Betriebssystem. Es fungiert eher als *Middleware* oder eine Art Framework⁷. Dabei liefert es Funktionalitäten sowie eingebaute Bibliotheken zur Kommunikation zwischen Komponenten. Unterstützt werden dabei überwiegend *Python* (`rospy`) sowie *C++* (`roscpp`) als Programmiersprachen. Auch Anbindungen für *Java* (`rosjava`) oder *JavaScript* (`rosnodejs`) sind verfügbar.

Hauptakteure im ROS Ökosystem sind sog. *Nodes*⁸. Jede *Node* erfüllt dabei eine in sich geschlossene Aufgabe. ROS ermöglicht die Kommunikation zwischen diesen Knoten unter Verwendung von *Topics*⁹ [Rob16d], *Services*¹⁰ [Rob16c] und *Messages*¹¹ [Rob16b]. Zusammen ergeben diese Komponenten ein Netzwerk, welches als Graph dargestellt werden kann [Rob16a]. Abbildung 2.1 zeigt ein Beispiel eines Berechnungsgraphen.

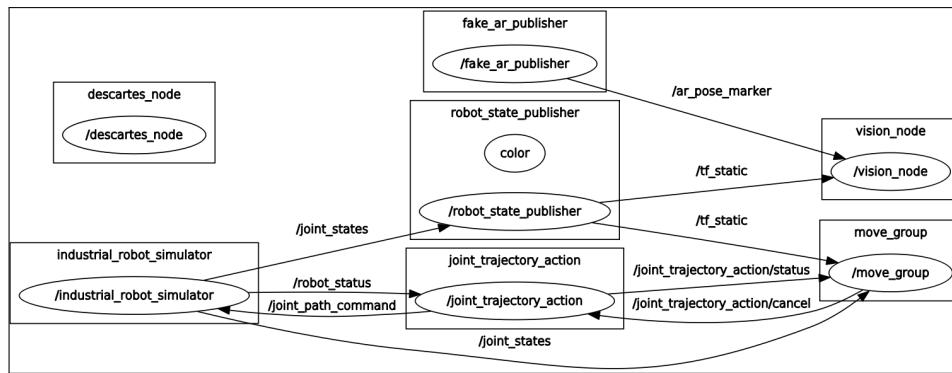


Abbildung 2.1: Darstellung eines ROS Graphen aus [Web17]

Ein Robotersystem umfasst dabei in der Regel verschiedene *Nodes*. Beispielsweise steuert ein Knoten einen Entfernungsmesser, ein weiterer führt die Lokalisierung durch und wieder ein anderer Knoten führt die Bahnplanung durch. Der ROS *Master* ermöglicht eine Namensregistrierung für die im Netz angemeldeten Knoten. Ohne den *Master* wären die Knoten nicht in der Lage, sich gegenseitig zu finden, Nachrichten auszutauschen oder Dienste aufzurufen.

Eine *Message* ist eine Datenstruktur, die aus typisierten Feldern besteht. Unterstützt werden primitive Standardtypen (Ganzzahl, Fließkomma, boolesche Werte usw.) sowie

⁷Ein Framework ist ein Rahmenwerk für Softwareentwicklung und Programmierung, das Grundstruktur und Programmiergerüst bereitstellt

⁸dt. Knoten

⁹dt. Themen

¹⁰dt. Dienste

¹¹dt. Nachrichten

Arrays primitiver Typen. Nachrichten können beliebig verschachtelte Strukturen und Arrays enthalten.

Die *Messages* werden auf Basis des *Publish/Subscribe Patterns* [Tar12] unter Zuhilfenahme von *Topics* zwischen den *Nodes* ausgetauscht. Das *Topic* ist eine Bezeichnung, die verwendet wird, um den Inhalt der Nachricht zu identifizieren. Ein Knoten, der an einer bestimmten Art von Daten interessiert ist, abonniert das entsprechende *Topic*.

Zuletzt helfen *Services* bei datenintensiver Kommunikation. Im Gegensatz zu *Topics*, die sich gegenseitig Informationen nach dem Publish-Subscribe-Modell senden, folgen Dienste einem *Request-Response*-Modell.

Die Vorteile von ROS für die Programmierung sind vielseitig. Die modulare Architektur ermöglicht eine gute Skalierung von Projekten. Durch die große Community ist außerdem eine Bandbreite an verschiedenen *Packages* [Rob23b] vorhanden. ROS liefert neben Community basierten *Packages* noch eine Reihe weiterer Entwicklungstools. Dazu gehören u. a. die Simulationssoftware *RViz* und *Gazebo* sowie Tools für das Debuggen von Anwendungen (*rqt_gui*). Hinzu kommt, dass ROS die Verwendung von Gerätetreibern und Schnittstellenpakete für Sensoren und Aktoren vereinfacht. Jedoch ist ROS im Einstieg komplex und kann daher für Neueinsteiger überwältigend sein.

Ein Grund dafür ist die Menge an notwendigen Wissen in Bereichen wie dem Linux-Betriebssystem, programmieren in *Python* oder *C++* und grundlegenden Ingenieurs- und Informatikbereichen. Aktuell wird mit der *ROS Noetic* Distribution ROS seinen *End-Of-Life* Zyklus im Mai 2025 erreichen. Gerade in den kommenden Jahren wird also ein Wechsel auf ROS2 unvermeidbar.

ROS2 versucht viele der Probleme von ROS zu vermeiden und setzt auf neue Ansätze [Rob23a]. Es erfolgt ein Wechsel der *primary/subordinate Middleware* auf *Data Distribution Service (DDS)*. DDS verspricht erhöhte Effizienz, Zuverlässigkeit, niedrigere Latenzzeiten, Skalierbarkeit sowie verbesserte konfigurierbare Parameter. ROS2 verfügt über eine in *C* geschriebene Basisbibliothek namens *rccl* auf der weitere Bibliotheken aufgebaut sind. Das ist einer der Hauptgründe dafür, dass ROS2 in der Lage ist, mehr Sprachen als nur *Python* und *C++* zu unterstützen, z. B. *Java* und *C#*. Weiterhin wird *Catkin* durch *Ament* als *Build-System* abgelöst. Durch das Wegfallen der *Master Node* wird außerdem ein *Single Point of Failure (SPOF)* beseitigt.

2.1.4 Künstliche Intelligenz

Der Begriff KI ist längst keine abstrakte Vorstellung mehr, die lange Zeit von Science-Fiction Romanen und Filmen geprägt wurde. Heutzutage hat der Begriff nur noch wenig mit den alten Vorstellungen zu tun. Es wird jedoch offensichtlich, dass KI aus der modernen Gesellschaft nicht mehr wegzudenken ist, sei es z. B. im Rahmen von Produkttempfehlungen auf *Amazon*, der automatischen Erkennung von Gesichtern auf Bildern, autonomen Fahrzeugen oder der Sprachassistentin *Siri*.

Zentraler Forschungsgegenstand von KI ist es, menschliche kognitive Fähigkeiten nachzuahmen, indem z. B. ein Algorithmus selbstständig ohne menschliches Zutun Strukturen und Zusammenhänge in Eingabeinformationen¹² entdeckt. [CSI23].

Geschichtlich kann die Entwicklung von KI in unterschiedliche Phasen einteilen [Xia22]. Abbildung 2.2 zeigt relevante Meilensteine der Entwicklung von KI über die letzten Jahrzehnte. Vorreiter der ersten Entwicklungen in diesem Gebiet, ist vor allem Alan Turing. In seinem Aufsatz [Tur09] beschäftigte er sich mit der Frage, ob Maschinen denken und Entscheidungen so rational und intelligent treffen könnten, wie ein Mensch.. Der daraus resultierte *Turing Test* wird heute noch verwendet.

Eine erste Begriffsdefinition entstand 1956 auf dem *Dartmouth Summer Research Project*. Hier wurden grundlegende Prinzipien von KI u. a. durch John McCarthy, Alan Newell, Arthur Samuel, Herbert Simon und Marvin Minsky konzipiert [McC+06].

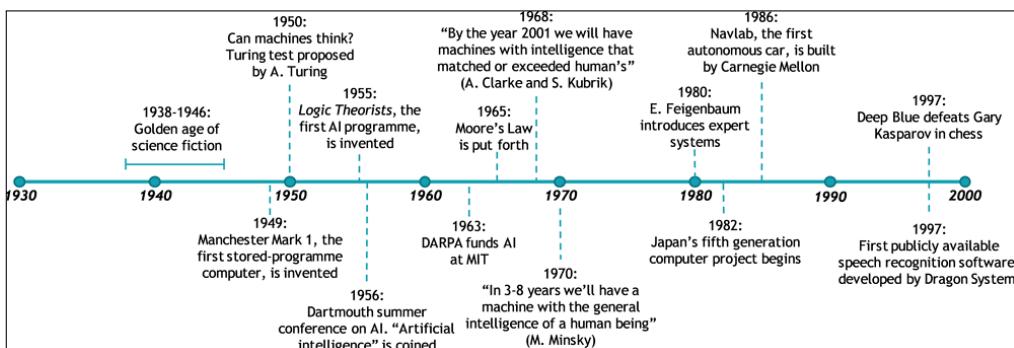


Abbildung 2.2: Zeitliche Entwicklung der KI betrachtet von 1930 bis 2000 mit relevanten Meilensteinen aus [OEC19]

Durch mangelnde Finanzierungen und sinkendes Interesse kam es in den 1970er Jahren zu einer Verlangsamung der Forschung. Durch die Zunahme an günstiger Rechenkapazität

¹²Dazu gehören u. a. Zahlen, Texte oder Bilder

zität und der damit einhergehenden Hardware, gewann KI wieder an Bedeutung.

Weitere Meilensteine bildeten in den folgenden Jahren der Chatbot *A.L.I.C.E* [Wal09] und *IBM's Deep Blue* Schach-Software [IBM23a], die den damaligen Weltmeister Gary Kasparov schlug.

Heutzutage ist die Weiterentwicklung von KI vor allem Gegenstand der Informatik. Durch den Anstieg von Rechen- und Speicherkapazitäten sowie die Weiterentwicklung von Technologien wie *Big-Data* und *Cloud-Computing* steigt die Leistungsfähigkeit sowie Verfügbarkeit von KI Applikationen an. Der technologische Fortschritt führt zu besseren und billigeren Sensoren, die zuverlässigere und größere Datenmengen erfassen, mit denen KI Software weiter verbessert wird [ENN23]. Begriffe die im Zusammenhang mit KI auftreten sind *Machine Learning (ML)* und *Deep Learning (DL)*. Diese bilden Unterkategorien des Überbegriff KI [RN21].

Machine Learning

ML verwendet statistische Lernalgorithmen, um intelligente Systeme zu entwickeln. ML Systeme können automatisch lernen und sich verbessern, ohne ausdrücklich¹³ programmiert zu werden. Empfehlungssysteme von Musik- und Video-Streaming-Diensten sind Beispiele für ML. Die Algorithmen des maschinellen Lernens werden in drei Kategorien unterteilt [Gér19]: *supervised*, *unsupervised* und *reinforcement learning*.

Im Fall von *supervised learning*¹⁴ wird mithilfe eines bereits *gelabelten*¹⁵ Datenset das Modell trainiert. Vergleichbar ist dies mit einem Lehrer, der seinen Schüler dauerhaft Feedback und Hinweise zu sowohl Lösungsweg als auch Korrektheit der Lösung liefert. Im *supervised learning* werden zwei Arten von Problemtypen betrachtet [RN21]. So genannte *classification problems*¹⁶ und *regression problems*¹⁷. In *classification problems* wird eine Vorhersage über die Zugehörigkeit diskreter Werte in eine Klasse oder Gruppe vorgenommen. Entgegengesetzt dazu beschäftigen sich *regression problems* mit kontinuierlichen Werten z. B. in der Vorhersage von Hauspreisen auf Grundlage der Größe des Hauses, der Lage und anderer Faktoren.

¹³Gemeint ist damit nicht, dass die Software ihren eigenen Quellcode schreibt. Jedoch eher der Algorithmus zur Erkennung von Strukturen und Zusammenhängen

¹⁴dt. überwachtes Lernen

¹⁵Das vorliegen von sog. *Ground-Truth* Daten

¹⁶dt. Klassifizierungsprobleme

¹⁷dt. Regressionsprobleme

Als Gegenstück zum *supervised learning* kann das *unsupervised learning*¹⁸[IBM23b] betrachtet werden. Algorithmen dieser Kategorie arbeiten teilweise ohne eine vollständige und sauber *gelabelte* Datenbasis. Ziel dieser Algorithmen ist es, zugrundeliegende Muster in Eingabedaten zu erkennen und diese in sinnvoller Weise zu gruppieren. Zu den wichtigsten Anwendungen des *unsupervised learning* gehören das *Clustering*, das Auffinden von Assoziationsregeln und die Erkennung von Anomalien.

Die letzte Kategorie bilden die Algorithmen des *reinforcement learning*¹⁹. Diese funktionieren so, indem Aktionen in einer geschlossenen Umgebung durchgeführt werden, um eine vordefinierte Gesamtbelohnung zu maximieren. Dabei werden verschiedene Aktionen ausgeführt und deren Einfluss auf die Gesamtbelohnung bewertet. Nach genügend Versuchen lernt der Algorithmus, welche Aktionen eine größere Belohnung bringen, und legt ein Verhaltensmuster fest [SB18]. Zu den Anwendungsbereichen gehören u. a. die Robotik oder Gamingbranche. Relevante Algorithmen für die Robotik, wie z. B. für die Navigation und Lokalisierung von Robotern, werden in Teil 2.1.5 genauer betrachtet.

Deep Learning

DL wird im Rahmen dieser Arbeit nicht ausführlich behandelt. Es folgt zur Vollständigkeit eine kurze Definition. DL ist der Teilbereich des ML, der sich mit Modellen beschäftigt, die auf vielschichtigen sog. *Artificial Neural Networks (ANN)* basieren. Diese nehmen sich das menschliche Gehirn und dessen Funktionsweise als Vorbild. Mithilfe mathematischer Modelle lassen sich somit ähnliche Strukturen abbilden. Das Wort *Deep* steht dafür, dass die Modelle viele Schichten von Knoten und Verbindungen besitzen. DL hat zu einer Vielfalt von Modellen, Techniken, Problemformulierungen und Anwendungen geführt [GBC16]. Hierzu gehören das automatisierte Fahren sowie die automatische Erkennung von Verkehrsschildern und Personen sowie die medizinische Forschung bei der Untersuchung von z. B. MRT-Scans. Letztendlich hat DL auch Einzug in den Smart-Home Bereich oder Endverbraucheranwendungen wie auf dem Handy erhalten.

¹⁸dt. unüberwachtes Lernen

¹⁹dt. bestärkendes Lernen

2.1.5 LiDAR-Technologie und Künstliche Intelligenz

Vor allem im Bereich des autonomen Fahrens ist die Verwendung von LiDAR Sensoren zum Standard geworden. Neben der Automobilbranche machen sich auch die Robotik sowie geographische Informationssysteme die Eigenschaften dieser Technologie zunutze. Dies liegt u. a. an der hohen Messpräzision sowie der Fähigkeit schnell große Areale zu mappen²⁰. Weiterhin haben derartige Systeme eine hohe Resilienz gegen äußere Einflüsse. Lediglich Reflexionen oder sehr schwere Witterungsbedingungen können den Messprozess beeinflussen [LIG20].

LiDAR Systeme senden Lichtimpulse aus und messen den Zeitraum des Lichtstrahls bis er wieder zurückstrahlt (siehe Gleichung 2.1). *ToF* steht für *Time of Flight*, also die Dauer die der Lichtstrahl unterwegs ist und c die Lichtgeschwindigkeit.

$$D = \frac{ToF \cdot c}{2} \quad (2.1)$$

Richtung und Entfernung dessen, worauf der Impuls trifft, werden als Datenpunkt aufgezeichnet. Die daraus resultierende Menge an Datenpunkten, bezeichnet man als *Point-Cloud*²¹. Jeder Punkt ist durch seine Position mit x, y, z Koordinaten und zusätzlichen Attributen wie z. B. einer Farbe für die Intensität, definiert. Die Punkte können dann gerendert werden, um ein detailliertes 3D-Modell der Umgebung zu erstellen. Die Verwendung dieser Repräsentationsform birgt jedoch Herausforderungen.

Beispielsweise die schnell größer werdende Menge an Datenpunkten²². Im Gegensatz zum herkömmlichen Ansatz der Bildverarbeitung, in der ein Bild üblicherweise die Form einer Pixelmatrix besitzt, hat die Darstellungsform der Punktwolken den Nachteil, dass es kein vordefiniertes *Grid* gibt. Somit ist also keinen systematischen Zusammenhang der einzelnen Punkte gegeben. Zuletzt bildet die Permutationsinvarianz einen weiteren relevanten Punkt. Diese sagt aus, dass in egal welcher Reihenfolge die Punkte der Punktwolke erscheinen, das gleiche Ergebnis zu erwarten ist. Das macht Sinn, da z. B. abhängig von der Richtung des Messgerätes die Darstellung der Punkte variieren kann [Gai+16] [Eng+17] [Hac+17].

Folgend wird auf das Zusammenspiel von KI, im speziellen ML, und dem Erstellen von Umgebungskarten sowie der Lokalisierung von Robotern eingegangen.

²⁰dt. aufzeichnen

²¹dt. Punktwolke

²²bis zu $\sim 10^8$ Punkte

Mapping und Navigation

ML bietet dem Bereich der Robotik eine Reihe von Werkzeugen für die Entwicklung komplexer Verhaltensweisen. Autonome mobile Systeme müssen gleichzeitig ihre Position in einer unbekannten Umgebung abschätzen und eine Karte erstellen. Diese Herausforderung wird auch als SLAM Problem bezeichnet.

Bis ungefähr Jahr 2010 wurden derartige Problemstellungen mithilfe von Filter-basierten Ansätzen gelöst [DWB06]. Aufgrund der breiten Verfügbarkeit und Weiterentwicklung nichtlinearer Optimierungsalgorithmen ist *Graph-SLAM* seit 2010 der beliebteste und effizienteste Ansatz für autonome mobile Systeme [Hen+20] [IPP21] [Wan+18]. Bei diesem Ansatz werden wichtige Punkte auf der Karte (sog. *Landmarks*) und Positionen eines Roboters durch einen Graphen dargestellt, wodurch das SLAM Problem durch nichtlineare Optimierungstechniken gelöst werden kann [Cad+16] [Hes+16]. Während SLAM Algorithmen bereits bei einfachen Anwendungen Fortschritte aufweisen, können sie bei anspruchsvollen dynamischen Roboterbewegungen oder Umgebungen an ihre Grenzen stoßen [Cad+16] [Spe+21].

Das grundlegende Problem von SLAM besteht darin, die Bewegungsbahn/Laufbahn des Roboters sowie die Position aller Umgebungsmerkmale abzuschätzen und zu ermitteln, ohne die wahre Position der Umgebungsmerkmale oder des Roboters selbst zu kennen [DWB06] [ZS14].

Seit dem Aufkommen und der raschen Verbesserung von Kosten und Effizienz in den letzten Jahren spielen LiDAR Sensoren eine Schlüsselrolle in der Erfassung der Umgebung und mobilen intelligenten Systemen [MSD18]. Das Einbinden von 3D-Punktwolken in bestehende Lokalisierungs-Systeme wirft jedoch weitere Probleme auf, vor allem hinsichtlich der Genauigkeit und dem Abgleich der Aktualisierungsrate anderer verwendeter Systeme wie einer IMU oder Kamerasystemen [Bur20].

Graph-SLAM und Scan-Matching Algorithmen

LiDAR Sensoren liefern 3D-Scans der Umgebung mit einer hohen Frequenz (siehe Kapitel 3.1.4), was – wie bereits erwähnt – zu großen Datenmenge führt. Die Änderungen zwischen einzelnen Scans sind oft minimal, mit nur kleinen Änderungen in der Translation und Rotation des Roboters. Um die Komplexität und Datenmenge zu reduzieren,

werden oft nur ausgewählte Scans, so genannte *Keyframes*, verwendet, um einen Graphen zu erstellen, der dann für das hinzufügen neuer Scans verwendet wird. So wird dem Graphen nur dann ein neuer Knoten hinzugefügt, wenn sich die Pose des Roboters translatorisch und rotatorisch signifikant verändert hat.

Mit jedem Scan eines LiDAR Sensors wird eine Punktwolke mit Messpunkten erstellt, wie im vorherigen Abschnitt erwähnt. Um diese Punktwolken für die Kartierung oder Lokalisierung zu verwenden, besteht die Herausforderung darin, den aktuellen Scan mit einem bereits bekannten Scan des umliegenden Gebiets abzulegen. Dieser Ansatz wird *Scan-Matching* bezeichnet. Zwei Algorithmen haben sich als gängige Optionen für diese Problematik herauskristallisiert. Der klassische *Iterative Closest Point (ICP)* Algorithmus und die generalisierte Version (*GICP*) [BM92]. Darüber hinaus wird die *Normal Distribution Transform (NDT)* in diesem Zusammenhang genannt [BS03]. Im folgenden werden NDT sowie der *Voxelized Generalized Iterative Closest Point Algorithm (VGICP)* genauer betrachtet. Dafür soll zunächst die grundlegende Idee hinter ICP erläutert werden, da die folgenden Algorithmen darauf aufbauen. Die Algorithmus-Definition 1 beschreibt die schrittweise Funktionsweise des ICP Algorithmus.

Ein alternativer Ansatz zu ICP ist der NDT Algorithmus. Bei diesem Ansatz ist im Gegensatz zur ICP Methode keine explizite Bestimmung der Punktzusammenhänge erforderlich. Stattdessen unterteilt NDT den gescannten Bereich in ein Gitter und berechnet die Normalverteilung der in jeder Zelle enthaltenen Punkte [BS03]. Der Hauptvorteil dieses Ansatzes besteht darin, dass keine explizite Berechnung zur Ermittlung der Zusammenhänge zwischen Punkten erforderlich ist. Jedoch ist die Effizienz von NDT abhängig von der Einstellung der Parameter, z. B. der Gittergröße (2D) oder der Voxelgröße (3D). Die optimale Voxelgröße ist im Wesentlichen von der Umgebung und den Sensoreigenschaften wie der Anzahl der Punkte abhängig. Wenn eine falsche Voxelgröße gewählt wird, führt dies zu Fehlern. Außerdem müssen genügend Punkte innerhalb eines Voxels liegen, sonst ist die Bestimmung der Kovarianzmatrix fehleranfällig [Koi+21] [HBMO22].

Da der in Kapitel 4 verwendete *hdl_graph_slam* Algorithmus VGICP verwendet, wird folgend dieser Ansatz kurz erläutert. VGICP kann als eine Kombination der oben genannten Methoden angesehen werden. Dafür wird der ICP Ansatz erweitert, indem die Verteilung der Punkte in Voxeln bestimmt wird, ähnlich der NDT Methode, die durch Aggregation der einzelnen Punktverteilungen innerhalb der Voxel erreicht wird. Die Kovarianzmatrix in VGICP wird dann aus den einzelnen Punktverteilungen errechnet. Dies führt zu korrekten Kovarianzmatrizen, auch bei einzelnen Punkten innerhalb eines Vo-

Algorithm 1: Schritte des Iterative Closest Point Algorithmus

Input: Punktwolken:

$$P_{Source} = \{x_1, \dots, x_n\}$$

$$Q_{Target} = \{p_1, \dots, p_n\}$$

Output: Transformationsmatrix T , die die Ausgangspunktwolke P so transformiert, dass sie der Zielpunktwolke Q möglichst gut entspricht.**1** Punktzuordnung: $\forall p_i \in P$ finde $q_{nearest}$ in Q durch z. B.:

$$q_{nearest} = \arg \min_{q_j \in Q} \|p_i - q_j\|$$

2 Berechne Schwerpunkte:(I) Schwerpunkt P :

$$P : \bar{p} = \frac{1}{n} \sum_{i=1}^n p_i$$

(II) Schwerpunkt Q

$$Q : \bar{q} = \frac{1}{n} \sum_{i=1}^n q_{nearest}$$

3 Berechnung Kovarianzmatrix H und Durchführung Singular Value Decomposition (SVD) [KL80] um Rotationsmatrix R zu erhalten:

$$\begin{aligned} H &= \sum_{i=1}^n (p_i - \bar{p})(q_{nearest} - \bar{q})^T \\ H &= U\Sigma V^T \\ R &= VU^T \end{aligned}$$

4 Berechnung Translation t basierend auf R :

$$t = \bar{q} - R\bar{p}$$

5 Anpassung der Punkte:

$$p'_i = Rp_i + t$$

6 Wiederhole 1 - 5 bis konvergent.

xels. Die Bestimmungen zusammengehöriger Punktpaare in VGICP erfolgt mithilfe eines *Single-to-multiple-distribution* Modell [Koi+21].

2.2 Herausforderungen und Möglichkeiten von vierbeinigen Robotern

Die Integration vierbeiniger Roboter in den Alltag birgt einige Herausforderungen und Möglichkeiten. Punkte die dabei zu berücksichtigen sind u. a. das physische Design sowie die Anpassungsfähigkeit an unterschiedliche Geländearten, die Navigation des Roboters, Sicherheitserwägungen um z. B. Kollisionen zu vermeiden, eine auf das Szenario abgestimmte Energieverwaltung sowie die Konnektivität zu möglicherweise weiteren Akteuren im Gesamtsystem und letztendlich Bildungsmöglichkeiten.

Hinsichtlich des physischen Designs, sollte für die Gliedmaßen sowie Korpus ein geeignetes Material gewählt werden. das sowohl robust als auch anpassungsfähig ist. Vor allem belastbare Materialien wie Aluminiumlegierungen oder kohlefaser verstärkte Verbundwerkstoffe haben sich über die Jahre aufgrund ihrer Eigenschaften bewährt [Mat19]. Wie bereits Eingangs erwähnt liegt der Vorteil vierbeiniger Roboter darin, dass sie, im Gegensatz zu Robotern auf z. B. Ketten, sich auch auf unebenem Terrain bewegen und leicht Hindernisse überwinden können. Durch den Einsatz von Simulationsumgebungen wie dem *Isaac Gym* [DEV23] von *NVIDIA* können mithilfe von ML Algorithmen optimale Bewegungsabläufe zuvor trainiert werden.

Bei der Entwicklung von Robotersystemen ist die Sicherheit nicht zu vernachlässigen. In Umgebungen in denen sich vorrangig Menschen und Tiere aufhalten sind entsprechende Schutzmaßnahmen zu berücksichtigen. Hierzu können Stoßdämpfer oder Schutzplatten eingesetzt werden, um sowohl Roboter als auch Außenstehende im Falle einer Kollision zu schützen. Die Implementierung von zusätzlichen Algorithmen zur Kollisionserkennung und Vermeidung [Mol+13] hilft dem Roboter dabei rechtzeitig auf Hindernisse zu reagieren. Dadurch können Kollisionen zum Teil vermieden werden.

Wie in Kapitel 3 noch genauer beschrieben, ist die Akkulaufzeit des Roboters begrenzt. Wird unwegsames Gelände, Treppen oder die Möglichkeit des Roboters zu rennen berücksichtigt, hat dies Auswirkungen auf die Laufzeit der Batterie. Eine effiziente Energieverwaltung führt zu einer längeren Einsatzdauer. Da der *Unitree Go1* Roboter ohne dedizierte Ladestation liefert wird, müssen für den zukünftigen Einsatz z. B.

Energierückgewinnungssysteme wie regenerative Bremssysteme eingesetzt werden. Allerdings tragen derartige Technologien nur begrenzt zu einer längeren Laufzeit bei.

Der Einsatz des Roboters im Universitätsumfeld führt auch zu Bildungsmöglichkeiten. Technikbegeisterten, Professoren, Studenten und Schülern kann mithilfe des Roboters ein Einblick in Konzepte wie Problemlösung, Programmierung und Robotik ermöglicht werden.

Zuletzt stellt vor allem die Navigation des Roboters ein komplexes Vorhaben dar. Die Fusion und Kombination von Technologien wie SLAM, Pfadplanungsalgorithmen wie *A** oder *Rapidly-exploring Random Tree (RRT)* sowie effizienter Hindernisvermeidung ist essentiell um ein robustes Navigationssysteme des Roboters zu ermöglichen. Weitere Ansätze beinhalten z. B. eine Terrainanalyse. Die Analyse von Geländemerkmalen hilft dem Roboter dabei seinen Pfad anzupassen, um Stabilität und eine effiziente Fortbewegungsart zu gewährleisten. Da der Vorteil von vierbeinigen Robotern die effiziente Bewegung in unwegsamem Terrain ist, sind vor allem Einsatzmöglichkeiten im militärischen oder humanitären Sektoren nennenswert. Verallgemeinert tritt hier der Begriff des *Service-Roboters* auf. Menschen aktiv bei ihren Aufgaben zu unterstützen, sei es als Blindenhund, Navigationshilfe in der Hochschulumgebung um Besucher zu ihrem Ziel zu führen oder für das Tragen von Gegenständen.

2.3 Verwandte Arbeiten

Die Bandbreite relevanter Forschungen ist vielseitig und erstreckt sich von grundlegenden Themen wie der Fortbewegung von Robotern bis hin zu komplexen Cloud-Systemen. In *A Study on Locomotions of Quadruped Robot* [TKDT14] wird das Gehverhalten vierbeiniger Roboter in flachem Gelände untersucht. Das Experiment analysiert die Lage des Schwerpunkts des Roboterkörpers in verschiedenen Bewegungsabläufen (Vorwärts, Rückwärts, Abbiegen, etc.). In *Trot Gait Design and CPG Method for a Quadruped Robot* [Zha+14] wird der Trab als Fortbewegung genauer untersucht. Die Arbeit folgt der Idee der sechs Determinanten des Gangs [HSL13] und entwirft einen Trab für vierbeinige Roboter.

Neben der Grundlagenforschung versucht *AQuRo: A Cat-like Adaptive Quadruped Robot With Novel Bio-Inspired Capabilities* [Sap+21] das Klettern und Überwinden von Hindernissen vierbeiniger Roboter durch Nachahmung einer Katzenstruktur effizienter zu gestalten. Hierfür wird eine neuartige Pfoten-Struktur vorgeschlagen, was das Erklim-

men steiler Leitern ermöglicht.

Hinsichtlich komplexerer Gesamtsysteme wird in *A Cloud-based Quadruped Service Robot with Multi-Scene Adaptability and various forms of Human-Robot Interaction* [Ma+20] ein Cloud-basiertes Mensch-Roboter System vorgeschlagen. Der Ansatz hierbei ist eine möglichst hohe Generalisierung der Einsatzmöglichkeiten eines vierbeinigen Robotern in verschiedenen Szenarien.

In *A1 SLAM: Quadruped SLAM using the A1's Onboard Sensors* [CD22] werden die SLAM-Funktionalitäten des *A1* Roboters (dem Vorgänger des *Go1*) betrachtet und gegen den *Cartographer* Algorithmus von *Google* ausgewertet. Ein robuster, schneller und performanter SLAM Ansatz spezielle für quadruped Roboter namens *FRL-SLAM* betrachtet die Arbeit *FRL-SLAM: A Fast, Robust and Lightweight SLAM System for Quadruped Robot Navigation* [Zha+21]. Vor allem Probleme wie beispielsweise ein Wackeln des Korpus wird durch diesen Ansatz entgegengewirkt. Zuletzt beschäftigt sich [MB+22] mit einer Betrachtung verschiedenster vorhandener SLAM Algorithmen und einer Auswertung dieser.

3 Inbetriebnahme des Unitree Go1 Roboters

Zentraler Inhalt dieses Kapitel ist die Vermittlung eines Überblickes von Hardware- und Softwarekomponenten des *Go1* Roboters im Werkszustand. Die zuvor in Kapitel 2.1.2 angesprochenen Merkmale werden hier verordnet. Anschließend erfolgt eine Analyse der Software des Roboters sowie Simulationsmöglichkeiten.

3.1 Hardwarekomponenten

Abbildung 3.1 zeigt verschiedene Ansichten des *Go1* Roboters. Im gefalteten Zustand besitzt dieser eine Größe von $0.54 \times 0.29 \times 0.1$ (Länge, Breite, Höhe) Metern. Im stehenden Zustand $0.645 \times 0.28 \times 0.4$ Meter. Ist die Batterie eingesetzt, so beträgt das Gewicht 12 kg. Sie wird mit zusätzlichen 1,5 kg veranschlagt. Die maximale Traglast ist mit $\sim 5 - 10$ kg angegeben²³. Die drei wesentlichen Komponenten des Roboters sind der Korpus, Kopf sowie die vier Gliedmaßen die als Aktoren dienen.

3.1.1 Überblick

Der Korpus beinhaltet den Großteil der verbauten Hardware. Dazu gehören die internen Hardwarekomponenten wie Akku, Sensorik und Kameras sowie Hüftgelenke einschließlich der Motoren der Beinaktoren. Wie aus Abbildung 3.1 ersichtlich ist, sind die Gelenke sowie die Motoren zur Steuerung der Hüftgelenke jeweils am vorderen und hinteren Teil des Korpus verbaut. Die Motoren sind zuständig für die Drehung der Hüfte und des

²³Die hier eingesetzte *EDU* Variante besitzt eine höhere Traglast. Bei herkömmlichen Modellen liegt sie bei 3kg.

Oberschenkels. Vier weitere Gelenk/Motor Bauteile befinden sich am oberen Ende der Beine, diese sind für das Anwinkeln zuständig. Sie sind Bestandteil der Gliedmaßen und nicht fest am Korpus verbaut.

Bei den Motoren handelt es sich um das Model *GO-M8010-6* der *Permanent-Magnet Synchronous Motor (PMSM)* Bauart mit einem maximalen Drehmoment von $23,7 \text{ N}\cdot\text{m}$ (Newton/Meter). Wird das Produktblatt des Motors betrachtet, so werden Features wie eine eingebaute *Field Oriented Control (FOC)* Kontrolleiterinheit, Temperatursensoren sowie ein *Absolute Value Encoder (AVE)*²⁴ aufgeführt. Ein Motor wiegt 530 g und kommuniziert durch eine *RS-485* Schnittstelle [Robb]. Zusammen ergeben die jeweils drei Motor/Gelenk Komponenten als Einheit drei *Degrees of Freedom (DOF)*²⁵. Somit erreicht der Roboter insgesamt 12 DOF, was für entsprechende Beweglichkeit sorgt. Tabelle 3.1 liefert einen Überblick über den Bewegungsumfang der Gelenke.

Gelenkbezeichnung	Bewegungsumfang
Laterales Hüftgelenk	$-40^\circ \sim +40^\circ$
Anteriores Hüftgelenk	$-218^\circ \sim +45^\circ$
Kniegelenk	$+24^\circ \sim +132^\circ$

Tabelle 3.1: Übersicht über Bewegungsumfang der Gelenke

Bei Betrachtung der Gliedmaßen, sind der obere und untere Teil (Schenkel und Knie) durch ein Kniegelenk miteinander verbunden, was für eine Streckung und Beugung der Beinkonstruktion sorgt. Am Ende der Aktoren befinden sich Drucksensoren²⁶. Diese dienen dazu um die Belastungswerte der Auftrittsstelle zu messen und an die *Main Control Unit (MCU)* weiterzureichen.

Insgesamt befinden sich fünf Stereo-Kamerasysteme am Roboter mit jeweils einem Blickwinkel von $150^\circ \times 170^\circ$. Zwei davon sind am Kopf verbaut. Weitere drei befinden sich an der linken und rechten Seite sowie der Bauchregion des Korpus. Ergänzend zur Kopfrektion sei angemerkt, dass an den Seiten zwei *LED*-Streifen verbaut sind, die verschiedene Farbtöne darstellen und ein Lautsprecher, der sich auf der Rückseite des Kopf befindet. Abbildung 3.4 zeigt eine Draufsicht des zerlegten Kopfteils.

²⁴Identifizieren die absolute Position sowie die Geschwindigkeit und Richtung des Drehgebers.

²⁵dt. Freiheitsgrade

²⁶Ebenfalls nur in der *EDU* Version vorhanden

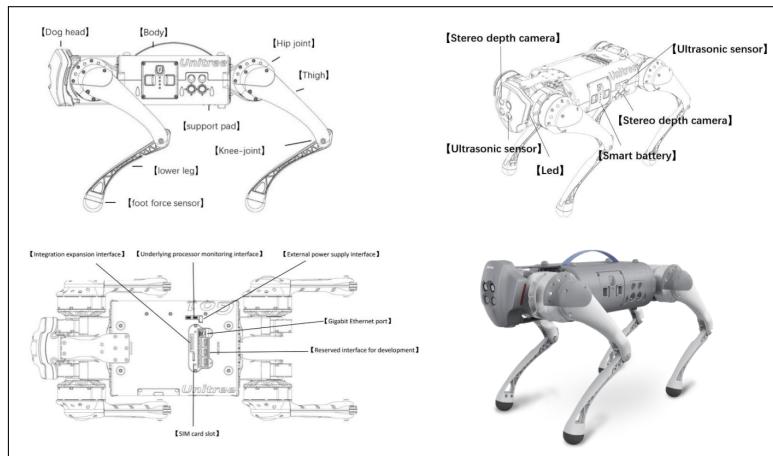


Abbildung 3.1: Übersicht über die Komponenten des *Go1* Roboters aus [Robe]

Laut Datenblatt [Robe] sind vier Ultraschallsensoren rund um den Kopf sowie den Rumpf verbaut. Nach genauer Betrachtung ließen sich jedoch nur drei feststellen. Dem Datenblatt nach ist ein am Hinterteil platziert Ultraschallsensor angedacht, jedoch nicht verbaut sondern nur für zukünftige Einbauten geplant. Das Erkennungsumfeld des Sensors liegt zwischen 5 – 200 cm.

Um den Roboter zu steuern, ist im Lieferumfang eine Fernbedienung enthalten, die sich per Funksignal mit dem Roboter verbindet. Auf der Fernbedienung sind vorprogrammierte Tricks und Funktionen wie Treppensteigen und Tänze gespeichert um die Möglichkeiten des Bewegungsumfangs zu zeigen. Durch einen am beispielsweise Gürtel platzierbaren Sender ist außerdem die *Follow-Me* Funktion aktivierbar. Dabei folgt der Roboter selbstständig dem Sender und weicht Hindernissen aus. Einige Versuche haben gezeigt, dass sich diese Funktion jedoch für Außenbereiche besser eignet als in geschlossenen Räumen. Nachdem wesentliche äußere Sensoren und größeren Bauteile betrachtet wurden, erfolgt ein Blick auf die am Rücken platzierten Anschlüsse.

3.1.2 Externe Anschlüsse

Nach außen werden auf der oberen Seite des Korpus verschiedene Anschlüsse zur Verfügung gestellt. Abbildung 3.2 zeigt die verfügbaren Anschlüsse in der Draufsicht. Zu den Anschlässen gehören jeweils 3 x *HDMI* und 3 x *USB-A* Anschlüsse. Diese ermöglichen den Zugriff auf die *NVIDIA Jetson's* sowie den *Raspberry Pi*. Weiterhin sind 1 x *RJ-45* und 1 x *SIM*-Karten Slot vorhanden. Neben einem *XT-30U* (24V, 30A) Anschluss, um den

Roboter über ein Netzteil mit Strom zu versorgen, gibt es außerdem noch 2 x *USB-C* Ports. Diese liefern eine Schnittstelle zu den *RS-485* Verbindung der Motoren. Bei dem letzten Anschluss handelt es sich um einen *40-pin Centronics Connector*, der für den Zugang zu verschiedenen Anschlüsse verwendet werden kann, die mit der Hardware im Inneren des Hundes verbunden sind. Denkbar wäre hier z. B. der Einbau eines *GPS*-Systems. Die äußere Betrachtung der am Roboter verbauten Hardware ist somit abgeschlossen. Folgend wird ein Einblick in den inneren Aufbau des Roboters und die darin enthaltene Hardware gegeben.

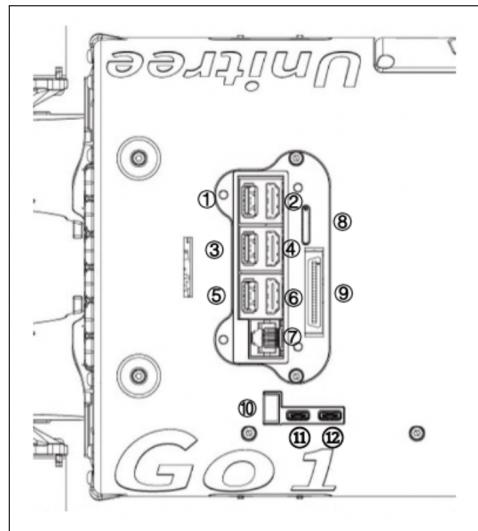


Abbildung 3.2: Anschlüsse und Ports auf der Oberfläche des *Go1* Roboters aus [Robe]

3.1.3 Interne Betrachtung

Abbildung 3.3 zeigt das Innere des Roboters nachdem der Korpus geöffnet wurde. Den größten Teil des Innenraums nimmt der Steckplatz für die Batterie ein. Dahinter befindet sich das Batterie Management System (BMS). Linksseitig des Batteriesteckplatzes befinden sich ein *NVIDIA Jetson Nano* sowie der *Raspberry Pi* und ein *NVIDIA Xavier NX*. Hauptaufgabe dieser ist das verwalten der zugeordneten Kameramodule und Ultraschallsensoren. Der *NVIDIA Jetson Nano* im Kopfbereich hat zusätzlich die Aufgabe, die Ultraschalldaten des nach vorne gerichteten Ultraschallsensors abzugreifen und weiterzuleiten. Tabelle 3.2 zeigt einen Überblick über die Hauptkomponenten des Roboters und deren Aufgaben. Die Tabellen C.1, C.2, C.3 im Anhang, geben eine Übersicht über

die wesentlichen Spezifikationen der einzelnen Komponenten. Eine erweiterte Betrachtung der Bauteile sowie deren Spezifikation liefert die Arbeit *Integration eines Unitree Go1 Quadruped Roboters in ein Hochschulökosystem* [Leh23].

Komponente Aufgabe

<i>Raspberry Pi</i>	Webhosting, App-Verbindung, Monitoring, Bibliotheken, Ultraschall, Wifi-Modul
<i>Nano 1 - 2</i>	LED-Steuerung, Audioausgabe, Ultraschall, Videodienste
<i>Xavier NX</i>	Videodienste, KI-Prozesse

Tabelle 3.2: Hauptkomponenten und deren Aufgaben im *Go1*

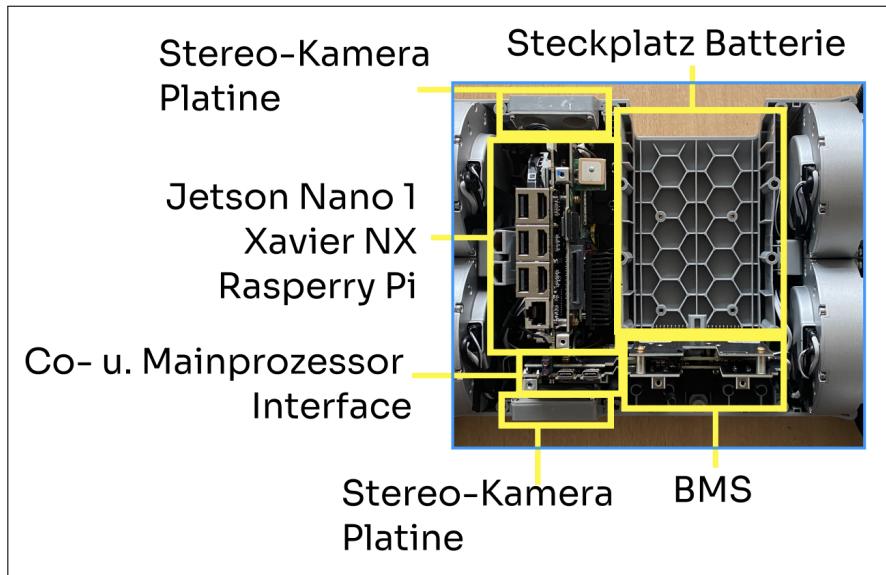


Abbildung 3.3: Draufsicht über den internen Aufbau des *Go1* Roboters und Zuordnung darin befindlicher Komponenten

Wird der Roboter nicht über eine externe Stromversorgung mithilfe des zuvor erwähnten *XT-30U* betrieben, so erfolgt dies mithilfe der Batterie. Ein Netzteil oder Adapter um den Roboter mit Strom zu versorgen ist nicht im Lieferumfang enthalten. Bei dem *GO1-BT03* Akku handelt es sich um einen Lithium-Ionen Akku, der in drei verschiedenen Ausführungen vorhanden ist. Mit 4500 mAh (Milliamperestunden), sowie der Standardausführung mit 6000 mAh und der *Long-Range* Variante mit 9000 mAh [Robc] [Ele]. Im Fall des verwendeten Roboters handelt es sich um die Standardausführung. Während der

Laborversuche haben Messungen gezeigt, dass ein Ladezyklus um den Akku vollständig aufzuladen $\sim 1,5$ Stunden beträgt. Je nach Auslastung des Roboters, also welche Funktionalität genutzt wird (Tricks, Treppensteigen, usw.), ist eine Neuaufladung nach ~ 1 Stunde notwendig.

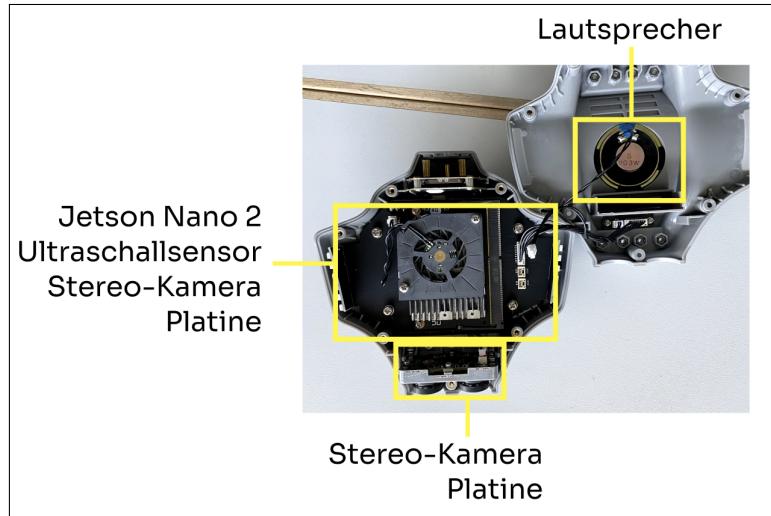


Abbildung 3.4: Draufsicht über den internen Aufbau des *Go1* Kopf und Verordnung darin befindlicher Komponenten

Um ein besseres Verständnis über das Zusammenspiel der einzelnen Komponenten zu erlangen ist es sinnvoll, diese in einem Diagramm darzustellen. Abbildung 3.5 zeigt ein Architekturdiagramm über die Bestandteile des *Go1* sowie deren Kommunikationskanäle. Zu erwähnen ist hier, dass der Aufbau sowie die Steuerungs- und Kontrollsysteme dem *MIT Cheetah* nachempfunden und zu Teilen sogar identisch sind. Ein Abgleich mit der Dokumentation des *MIT Cheetah* ist also ratsam.

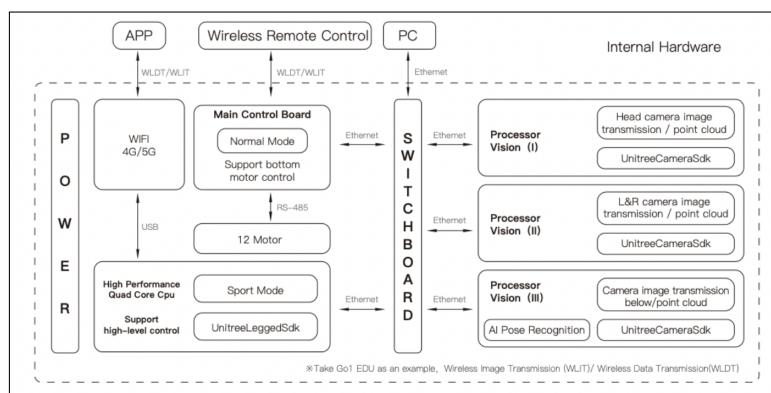


Abbildung 3.5: Architekturdiagramm des *Go1* aus [MAV]

Das in Abbildung 3.6 dargestellte Diagramm zeigt die Kommunikationsarchitektur des *MIT Cheetah* im Detail. Im *Unitree* Design ersetzen der *Raspberry Pi* sowie die *NVIDIA Jetson Nanos* und der *NVIDIA Xavier NX* die als *UP* gekennzeichnete Schnittstelle zur MCU. Das *RS-485* Netzwerk und die Bewegungsverarbeitung sind nahezu identisch. Im Falle des *Go1* handelt es sich bei der MCU um einen *STM32H743* mit MicroROS [ROS] [MAV]. Das BMS kann lediglich mittels Herstellerbibliotheken gelesen und manipuliert werden. Ein Direktzugriff auf die MCU ist nur lesend über Drittanbieter Bibliotheken möglich.

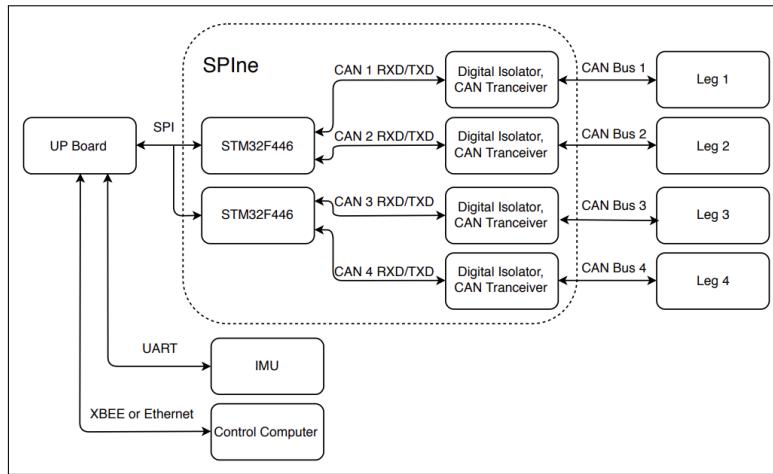


Abbildung 3.6: Architekturdiagramm des *MIT Cheetahs* aus [Kat18]

Auf weitere Details hinsichtlich der Implementierung einzelner Kommunikationswege und Steuerungssysteme wird folgend verzichtet, da diese eine untergeordnete Rolle für den Inhalt dieser Arbeit darstellen. Die Ausfertigung *A low cost modular actuator for dynamic robots* [Kat18] beschreibt die zugrundeliegende Kommunikationsarchitektur zwischen den Motoren sowie der MCU ausführlicher. Ebenfalls interessant sind in diesem Rahmen die Ausfertigungen des *Biometric Robotics Lab*²⁷. Weitere Ressourcen für den *Go1* werden in Kapitel 3.2.4 aufgeführt. Folgend wird das Thema Konnektivität kurz betrachtet und Merkmale des verwendete LiDAR Sensor erläutert, bevor anschließend die Softwarekomponenten betrachtet werden.

²⁷Zu finden unter: <https://biomimetics.mit.edu/publications>

Konnektivität

Eine ausführliche Betrachtung der Konnektivität des Roboters wird in *Integration eines Unitree Go1 Quadruped Roboters in ein Hochschulökosystem* [Leh23] gegeben, hier erfolgt auch die Betrachtung der 4G/5G Möglichkeiten des Roboters.

Alle direkt ansteuerbaren Komponenten sind per Ethernet – wie aus Abbildung 3.5 ersichtlich – mit einem Switch verbunden. Von außen kann auf den Switch des Roboters per Anschluss zugegriffen werden (siehe Abbildung 3.2). Die Registrierung aller Komponenten erfolgt dabei in dem Netzwerk mit der IP 192.168.123.0/24. In Tabelle 3.3 ist die Aufteilung der IP-Adressen aufgeführt.

Neben der verkabelten Verbindung, besitzt der *Raspberry Pi* weiterhin ein *Wireless Wide Area Network (WWAN)* Modul um das Netzwerk nach außen zu veröffentlichen. Diese drahtlose Verbindung kann als Zugriffspunkt auf das Robotersystem genutzt werden. Als *Gateway* dient dabei der *Raspberry Pi*.

Komponente	IP-Adresse
MCU	192.168.123.10
<i>Raspberry Pi</i>	192.168.123.161
<i>NVIDIA Jetson Nanos:</i>	192.168.123.13/14
<i>NVIDIA Xavier NX</i>	192.168.123.15

Tabelle 3.3: Ermittelte IP-Adresse der Roboter Hardware

3.1.4 Überblick RS-Helios-16P LiDAR

Bei dem auf dem *Go1*, mithilfe von Schienen, montierten LiDAR handelt es sich um den *RS-Helios-16P* der Firma *RoboSense*. Mithilfe von 16 Strahlen wird in einem 360° Umfeld die Umgebung ausgemessen. Die Reichweite beträgt dabei $\sim 150\text{ m}$, wobei die Präzision mit zunehmender Entfernung abnimmt²⁸. Die Abtastfrequenz beträgt 10 Hz (Hertz) oder 20 Hz bei einer Rotationsgeschwindigkeit von 600 oder 1200 *Rounds Per Minute (RPM)*²⁹. Tabelle 3.4 führt weitere technische Merkmale auf [Roba].

²⁸ $\pm 2\text{cm}$ (1m bis 100m); $\pm 3\text{cm}$ (10cm bis 1m); $\pm 3\text{cm}$ (100m bis 150m)

²⁹dt. Umdrehungen pro Minute

Eigenschaft	Wert
Laserwellenlänge	905nm (Nanometer)
Vertikales Sichtfeld	30°; ($-15^\circ \sim +15^\circ$)
Horizontale Auflösung	0,2° / 0,4°
Vertikale Auflösung	2°
Datendurchsatz	~ 288.000 Punkte / ~ 576.000 Punkte (Double Return)
Output	UDP-Pakete über Ethernet
Gewicht	~ 900g

Tabelle 3.4: Tabelle über die technische Spezifikationen des *RS-Helios-16P* Sensors aus [Roba]

Hilfreich bei der Verwendung des LiDAR ist das *rslidar_sdk* [RL]. Die Bibliothek stellt Funktionen zur Interaktion mit dem LiDAR bereit. Dazu gehört u. a. das Senden von *Point-Cloud* Daten mithilfe von ROS sowie das Aufzeichnen der *Point-Cloud* im *rosbag* Format. Zur Visualisierung der *Point Cloud Data (PCD)* kann das Tool *RSView* verwendet werden. Es bietet dem Benutzer eine Reihe von Möglichkeiten. Dazu gehört das Übertragen von Live-Daten, das Aufzeichnen sowie Abspielen von aufgezeichneten Dateien und einige Werkzeuge für das Bearbeiten (Crop, Distanzmessung, usw.) der Punktfolke. Während einiger Versuche konnten mithilfe des Programms zwar die Daten gut visualisiert werden, jedoch war das Aufzeichnen und Abspeichern der Daten in Formaten wie *PCD*, *LAS*, *PCAP* nicht möglich, da jedes Mal eine Fehlermeldung (siehe Anhang Abbildung A.6) seitens des Programm ausgegeben wurde. Hinzu kommt, dass laut Benutzerhandbuch eine Aufnahme der Live-Daten zwar möglich sein soll, der erforderliche Button dafür jedoch nicht im Programm implementiert ist.

Wie zuvor erwähnt, nutzt der LiDAR zur Kommunikation *User Datagram Protocol (UDP)* Pakete. Spezifisch werden zwei Protokolle verwendet. Nämlich *Main Data Stream Output Protocol (MSOP)* und *Device Information Output Protocol (DIFOP)*. Das MSOP Protokoll versendet dabei die Sensordaten, also die Daten der Punktfolke (Entfernung, Winkel, usw.), während über den DIFOP Kanal verschiedene Konfigurationsinformationen des LiDAR ausgegeben werden. Die Paketgröße beider Protokolle beträgt 1248 Byte. Die Standardports sind 6699 für MSOP und 7788 für DIFOP. Eine genaue Struktur der Pakete ist Abbildung 3.8 zu entnehmen. Das detaillierte Vorgehen zur Konfiguration und Implementierung des LiDAR erfolgt in Kapitel 4.

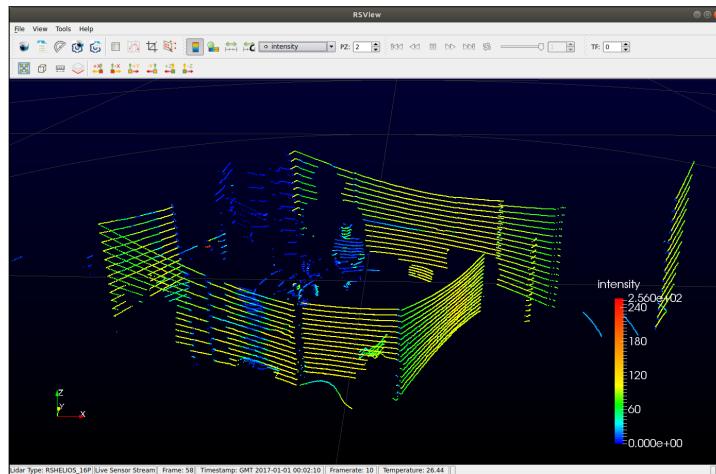


Abbildung 3.7: Ausschnitt aus dem Programm *RSView*

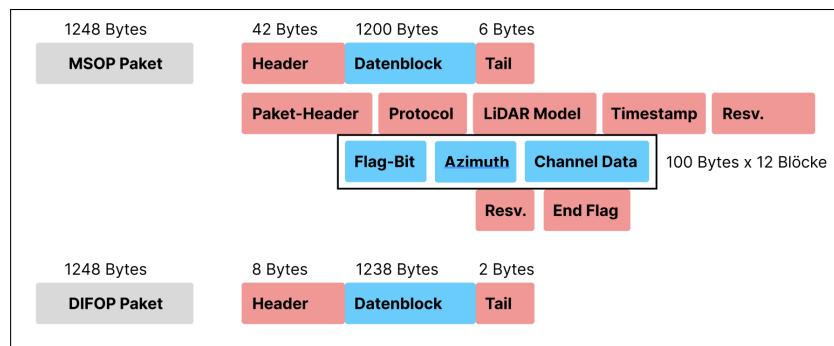


Abbildung 3.8: Aufteilung der UDP-Pakete des *RS-Helios-16P* Sensors

3.2 Softwarekomponenten

Die Untersuchung der *NVIDIA Jetson Nanos* und *NVIDIA Xavier NX* mittels eines *Dump* ergab, dass es sich bei dem Betriebssystem um eine *Ubuntu* Distribution der Version *18.04.5 LTS (Bionic Beaver)* handelt. Beim *Operating System (OS)*³⁰ des *Raspberry Pi* handelt es sich um *Debian GNU/Linux 10 (buster)*. Auf den *NVIDIA Jetson Nanos*, *NVIDIA Xavier NX* und *Raspberry Pi* befinden sich verschiedene ROS Pakete. Die Abbildungen A.1 und A.2 im Anhang zeigen eine Auflistung der installierten ROS Pakete. Die ROS Pakete auf den *NVIDIA Jetson Nanos* und dem *NVIDIA Xavier NX* sind identisch.

Für den Roboter ist weiterhin eine App für *iOS* sowie *Android* vorhanden. Die App kann über *iOS* mithilfe der *TestFlight* Anwendung installiert werden. Ist eine drahtlose Ver-

³⁰dt. Betriebssystem

bindung mit dem Netzwerk des Roboter vorhanden, so stellt diese einige Funktionen zur Verfügung. Die Hauptfunktionen der App sind u. a. ein Simulator und eine Statusanzeige für das BMS, die Gelenkpositionen und den aktuellen Status des Roboter (Versionen von Firmware, App, etc.).

Mithilfe des Simulator lässt sich ein digitaler Klon des *Go1* in einer eingeschränkten Umgebung bewegen. Die Steuerung erfolgt über das Handydisplay. Auch ein direktes Ansteuern eines echten Roboter ist mithilfe der App möglich. Eine Schaltfläche liefert die Möglichkeit, den Videostream der fünf Kamerasysteme anzuzeigen. Eine Oberfläche für grafische Programmierung mithilfe von Blöcken ist ebenfalls vorhanden. Die Möglichkeiten sind hier jedoch eingeschränkt. Einfache Befehle wie z. B. das Einstellen der *LED's* in einer bestimmten Farbe nach Ablauf eines Timers sind möglich. Für das Testen der Lautsprecher wird ebenfalls die App verwendet. Sofern ein LiDAR auf dem Roboter platziert und angeschlossen ist, besteht die Möglichkeit *Unitrees SLAM* Verfahren zu starten. Versuche auf Werkseinstellung dahingehend brachten keinerlei Erfolge die Anwendung zu starten. Abbildung A.5 im Anhang zeigt einen Ausschnitt aus der *iOS*-App. Die Web-Applikation liefert dabei die gleiche Funktionalität und besitzt nur wenig Unterschiede. Ein Update der Firmware ist mithilfe der Web-Applikation möglich. Außerdem lässt sich die im Werbevideo versprochene KI Software, durch die Web-Applikation anzeigen. Eine begrenzte Auflösung der Kamera und ein unzuverlässiger Erkennungsalgorithmus machen diese Anwendung jedoch nahezu unbrauchbar. Eine Abbildung zur eingebauten KI Erkennung ist in Anhang A.3 und A.4 zu finden.

3.2.1 Analyse des UnitreeCameraSDK

Mithilfe des *UnitreeCameraSDK* lassen sich diverse Funktionen der eingebauten Kameras abgreifen. Dieser Abschnitt dient dazu, die Bibliothek genauer zu betrachten.

Mithilfe des *Software Development Kit (SDK)* lassen sich Tiefen- und Farbbilder abgreifen. Weiterhin ist es möglich, die Kalibrierungsparameter der Kamera einzuholen. Als *Dependencies*³¹ sind *CMake* in der Version 2.8 oder höher, *OpenGL*, *GLUT* sowie *X11* angegeben. *OpenGL* ist eine sprach- und plattformübergreifende Programmierschnittstelle für das Rendern von 2D und 3D-Grafiken. Das *OpenGL Utility Toolkit (GLUT)* ist eine Bibliothek, die hauptsächlich *Input and Output (I/O)* Aufgaben mit dem Host-Betriebssystem durchführen. *X11* unterstützt Bitmap-Anzeigen. Dabei liefert es den

³¹dt. Abhängigkeiten

grundlegenden Rahmen für eine Graphical User Interface (GUI) [Robd].

Eine Anleitung zur Nutzung des SDK ist auf der dazugehörigen GitHub-Seite <https://github.com/unitreerobotics/UnitreecameraSDK> nicht aufzufinden. Stattdessen liefert *Unitree* ein Video auf der Plattform *YouTube*, in dem die Nutzung erklärt wird. Anzumerken ist jedoch, dass dieses Tutorial unvollständig ist. Eine Verlinkung zu einem externen Dokument unterhalb des Videos liefert weitere Details. Diese sind ebenfalls unvollständig oder nur in der Sprache *Chinesisch* vorhanden. Folgend wird ein Einstieg in die Bibliothek gegeben.

Auf dem Roboter (Sender)

Zunächst müssen einige ROS *Nodes* abgeschaltet werden, da diese sonst mit dem SDK interferieren könnten. Eine Nutzung während die *Nodes* noch in Betrieb sind, ist nicht möglich.

```

1 ps -aux | grep point_cloud_node | awk '{print $2}' | xargs kill -9
2 ps -aux | grep mqttControlNode | awk '{print $2}' | xargs kill -9
3 ps -aux | grep live_human_pose | awk '{print $2}' | xargs kill -9
4 ps -aux | grep rosnode | awk '{print $2}' | xargs kill -9

```

Listing 3.1: Bash Code zum Abschalten der zuvor erwähnten ROS *Nodes*

Als nächstes müssen auf dem Roboter einige Änderungen in der Konfigurationsdatei für die Parameter vorgenommen werden. Dafür steht die Datei `trans_rect_config.yaml` zur Verfügung. Hier sind vor allem zwei Parameter von Bedeutung, nämlich `IpLastSegment` und `Transmode`. Ersteres definiert die IP-Adresse des Empfängers und letzteres gibt die Art der Übertragung³² an.

```

1 IpLastSegment: !!opencv-matrix
2 rows: 1
3 cols: 1
4 dt: d
5 data: [99.]
6 Transmode: !!opencv-matrix
7 rows: 1
8 cols: 1
9 dt: d
10 data: [2.]

```

Listing 3.2: Ausschnitt aus der Konfigurationsdatei für die Kamera-Parameter

³²0-Kamera Rechts, 1-Stereo, 2-Kamera Rechts (Rectified), 3-Stereo (Rectified)

Anschließend kann die Bibliothek kompiliert werden. Sofern noch kein `build` Verzeichnis vorhanden ist, muss dieses zusätzlich erstellt werden.

```

1 cd UnitreecameraSDK
2 mkdir build
3 cd build
4 cmake ..
5 make

```

Listing 3.3: Erzeugen des `build` Verzeichnis und kompilieren

Das im SDK vorhandene Beispiel `example_putImageTrans.cpp` für das Starten der Übertragung kann nun verwendet werden. Eine mit eigenen Kommentaren versehene Version ist im Anhang unter B.1 zu finden.

```

1 ./bins/example_putImageTrans
2 [UnitreeCameraSDK][INFO] Load camera flash parameters OK!
3 [StereoCamera][INFO] Initialize parameters OK!
4 [StereoCamera][INFO] Start capture ...
5 hostIp+portString:host=192.168.123.99 port=9201
6 Framerate set to : 30 at NvxVideoEncoderSetParameterNvMMLiteOpen
7 : Block : BlockType = 4
8 ===== NVMEDIA: NVENC =====
9 NvMMLiteBlockCreate : Block : BlockType = 4
10 H264: Profile = 66, Level = 40

```

Listing 3.4: Ausführen der Binary und Bestätigung des Kamera-Stream

Der Roboter sendet nun UDP-Pakete an die zuvor angegebene IP-Adresse im Netzwerk des Roboters. Der nächste Schritt beinhaltet das Empfangen der Dateien sowie das Abspielen dieser.

Auf dem Empfänger-System

Sollte im `UnitreecameraSDK` Verzeichnis noch kein `build` Order vorhanden sein, so muss der Schritt aus Listing 3.3 wiederholt werden. Für das Empfangen des Videostreams wird die mitgelieferte Datei `example_getimagetrans.cpp` verwendet. Eine genauere Betrachtung der Datei mit Kommentaren findet sich ebenfalls im Anhang unter B.2.

Je nachdem, ob es sich bei dem Empfängergerät um eine *64-bit AMD Architektur* oder *ARM* basierten Prozessor handelt, müssen die *Decoder*-Einstellungen in der Datei `example_getimagetrans.cpp` modifiziert werden.

Für AMD-64 basierte Prozessoren:

```

1 std::string udpstrBehindData = " ! application/x-rtp,media=video,
2 encoding-name=H264 !
3 rtpH264Depay ! h264parse ! avdec_h264 ! videoconvert ! appsink";

```

Listing 3.5: Decoder Einstellungen für AMD basierte Prozessoren

Für ARM basierte Prozessoren:

```

1 std::string udpstrBehindData = " ! application/x-rtp,media=video,
2 encoding-name=H264 !
3 rtpH264Depay ! h264parse ! omxh264dec ! videoconvert ! appsink";

```

Listing 3.6: Decoder Einstellungen für ARM basierte Prozessoren

Nach erfolgreichem Ausführen der `./bins/example_getimagetans` Datei, wird das Kamerabild angezeigt. Es ist zu erwähnen, dass aufgrund von Zeitgründen sowie einer defekten Kameraplatine, das erfolgreiche Übertragen der weiteren zur Verfügung gestellten Möglichkeiten, wie z. B. Tiefenbild, Punktfolke, nicht möglich war. Während der Analyse wurden diese zwar getestet, jedoch kam es immer zu einer Fehlermeldung. Diese Fehlermeldung ist im Anhang unter A.7 dokumentiert. Eine erste Vermutung ist, dass es eine Diskrepanz zwischen der Konfiguration der Kamera und dem *OpenGL Decoder* gibt.

Nachdem das Kamera-SDK untersucht wurde, folgt eine Anleitung für die Ultraschallsensoren des *Go1*.

3.2.2 Analyse der Ultraschallsensoren

Eine genaue Anleitung zur Nutzung der Ultraschallsensoren ist seitens *Unitree* nicht vorhanden. Lediglich eine *Readme*³³ Datei wurde gefunden, welche Rückschlüsse auf die Verwendung zulässt. Die folgende Anleitung basiert auf eigens auf dem Roboter durchgeführten Tests und ist daher möglicherweise unvollständig. Folgend werden die einzelnen Schritte zur Auswertung der Ultraschall-Daten dargelegt.

Zunächst wird eine Liste der auf dem *Raspberry Pi* laufenden ROS Topics mithilfe von `rostopic list` abgefragt. Die vollständige Ausgabe ist im Anhang unter B.4 zu finden.

³³Eine Datei, die meist Anleitungen zur Verwendung der zugrundeliegenden Software enthält.

Interessant sind die in Listing 3.7 ermittelten Topics.

```

1 pi@raspberrypi: rostopic list
2 ...
3 /range_ultrasonic_face
4 /range_ultrasonic_left
5 /range_ultrasonic_right
6 ...

```

Listing 3.7: Ausschnitt der auf dem *Raspberry Pi* laufenden ROS *Topics*

Eine genauere Betrachtung der Struktur der *Topics* mithilfe des Befehl `rostopic echo /range_ultrasonic_right` ergab (Listing 3.8):

```

1 pi@raspberrypi: rostopic echo /range_ultrasonic_right
2 ---
3 header:
4   seq: 14996
5   stamp:
6     secs: 1636154800
7     nsecs: 45547136
8   frame_id: "camera_right"
9 radiation_type: 0
10 field_of_view: 1.57079637051
11 min_range: 0.050000007451
12 max_range: 2.0
13 range: 0.363781124353
14 ---

```

Listing 3.8: Struktur des ROS *Topics* für die vermuteten Ultraschallsensoren

Bei den Angaben `range`, `min_range` sowie `max_range` handelt es sich um Meter. Diese Annahme resultiert daraus, dass während des Tests, an der rechten Seite des Roboters eine Wand in ~ 30 cm Entfernung stand. Untersuchungen des *Raspberry Pi* Verzeichnis lassen weitere Rückschlüsse zu. So finden sich unter dem Pfad `pi@raspberrypi: home/Unitree/autostart/utrack` folgende Verzeichnisse (Listing 3.9).

```

1 pi@raspberrypi:~/Unitree/autostart/utrack  ls -la
2 total 36
3 drwxr-xr-x  6 pi pi 4096 Feb 16 2022 .
4 drwxr-xr-x 14 pi pi 4096 Aug 18 05:11 ..
5 drwxr-xr-x  4 pi pi 4096 Feb 16 2022 catkin_utrack

```

```

6 drwxr-xr-x 4 pi pi 4096 Feb 16 2022 g1_ukd2udp
7 drwxr-xr-x 4 pi pi 4096 Feb 16 2022 g1_ultrasonic_lcm
8 -rwxr-xr-x 1 pi pi 968 Feb 16 2022 run.sh
9 drwxr-xr-x 6 pi pi 4096 Feb 16 2022 ultrasonic_listener_example
10 -rwxr-xr-x 1 pi pi 138 Feb 16 2022 utrack.sh
11 -rw-r--r-- 1 pi pi 5 Feb 16 2022 version.txt

```

Listing 3.9: Ausgabe des Verzeichnis utrack im autostart Ordner des *Go1*

Für die Ultraschallsensoren relevant scheinen dabei die Ordner `g1_ultrasonic_lcm` und `ultrasonic_listener_example` zu sein. In letzterem Verzeichnis findet sich auch die erwähnte *Readme* Datei. Die darin beschriebenen Schritte brachten allerdings keinen Erfolg die Ultraschallsensoren auszulesen. Die Daten werden zwar erzeugt, jedoch nicht empfangen. Es ist festzuhalten, dass die Sensorwerte mithilfe von *Lightweight Communications and Marshalling (LCM)* veröffentlicht und abonniert werden können. Als alternativer Ansatz ist es möglich, mittels des `unitree_legged_sdk` die Daten der Ultraschallsensoren auszulesen. Dazu wird z. B. die Datei `example_walk` so modifiziert, dass anstatt der vorprogrammierten Befehle, eine Ausgabe der `rangeObstacle` Methode erfolgt (siehe Anhang B.3). Nachdem die Datei kompiliert und ausgeführt wurde, erfolgt die Anzeige der Werte im Terminal (Listing 3.10).

```

1 rangeObstacle= [Head: 1.725300, Left: 0.273141, Right: 0.363781]

```

Listing 3.10: Beispielhafte Ausgabe der Ultraschallsensorwerte

Der Grund für die fehlerhafte Übertragung mithilfe von LCM konnte während der Tests nicht ermittelt werden. Es wurden verschiedene Situationen (z. B.: `triggerSport` an-/ausgeschaltet, Neukompilieren des Projekts, usw.) getestet, jedoch ohne Erfolg. Weiterhin rätselhaft sind die Binary-Dateien im Verzeichnis `g1_ultrasonic_lcm`. Ein Starten dieser brachte ebenfalls keine weiteren Erkenntnisse. Hierzu sind keine Quelldateien vorhanden, ebenso wenig wie eine Erklärung. Weitere Untersuchungen in diesem Bereich sind für zukünftige Arbeiten also denkbar.

3.2.3 Simulation des Unitree Go1

Um einige Funktionen des Roboter zu testen, wie z. B. das Ansteuern einzelner Motoren, ist es sinnvoll, eine Simulationsumgebung zu verwenden, um Schäden vorzubeugen. Inhalt dieses Abschnitt ist die Erläuterung der Schritte für das Simulieren des *Go1*. Dabei bietet das *unitree_ros* Paket alle wesentlichen Funktionen um den Roboter zu simulieren. Inzwischen sind *Go1* Roboter auch in allen wesentlichen Frameworks und Tools eingebunden. So z. B. im *CHAMP* Framework [cha] oder in *NVIDIA's Omniverse* Software [Nvi]. Tests der Simulation mithilfe der *Omniverse* Software wurden aufgrund der zeitlichen Einschränkung nicht durchgeführt, könnten in Zukunft jedoch in Betracht gezogen werden.

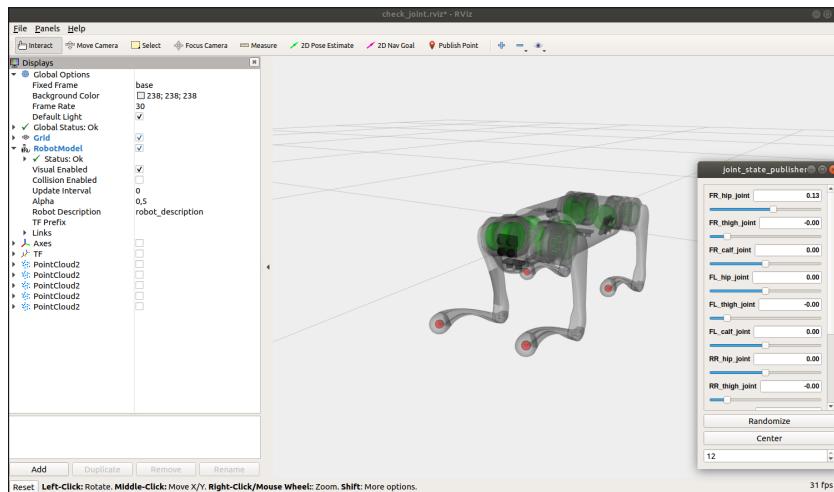


Abbildung 3.9: Ausschnitt aus dem Programm *RViz* zur Visualisierung der Gelenkwinkel des *Go1*

Die Simulation wurde auf einem Rechner mit *Ubuntu 18.04*, 62 GB Arbeitsspeicher, einem *Ryzen 9 3900X* 12-Kern-Prozessor, und einer *NVIDIA GeForce RTX 2090 Super* mit 8 GB Speicher durchgeführt. Weiterhin war *CUDA* zur Hardwarebeschleunigung auf dem System installiert. Die Simulation erfolgt dabei mithilfe von *Gazebo* sowie *RViz*. *Gazebo* und *RViz* sind Tools, die in Robotik- und Simulationsumgebungen eingesetzt werden, um die Entwicklung und Visualisierung von Robotersystemen zu unterstützen. *Gazebo* liefert eine physikbasierte Simulationsumgebung, in der Roboter, Sensoren und weitere Umgebungsobjekte dargestellt werden können. Dagegen ist *RViz* Bestandteil der ROS Umgebung und dient als Visualisierungswerkzeug von Sensordaten, Robotermode-

len oder Laserscans. Für die Modellierung des Roboters wird das *Unified Robot Description Format (URDF)* verwendet. Mithilfe einer *Extensible Markup Language (XML)* Datei werden physikalischen Eigenschaften wie Kinematik, Geometrie, Aufbau und Komponenten des Roboters definiert.

Sind alle Abhängigkeiten installiert, so startet der Befehl `rosrun urdf ros_urdf.py urdf_to_stl.py` und `roslaunch laikago_rviz.launch`. Das Visualisierungsprogramm wie Abbildung 3.9 zeigt. Es lassen sich nun mithilfe der Schieberegler die einzelnen Gelenkwinkel simulieren. Die Auswirkung der Winkelveränderungen wird unmittelbar dargestellt. Die Simulation des kompletten Roboters erfolgt mittels *Gazebo*. Dazu wird ein identisches Abbild eines realen *Go1* gerendert. Weiterhin lassen sich verschiedene Umgebungen (planare Ebene, Treppen, Häuser, usw.) darstellen. Für das Starten der Umgebung wird `roslaunch unitree_gazebo normal.launch rname:=go1 wname:=stairs` verwendet. Der Parameter `rname:` gibt den Modellnamen an und `wname:` den Dateinamen der Umgebung die geladen werden soll. Zunächst wird der Roboter im liegenden Zustand in die Umge-

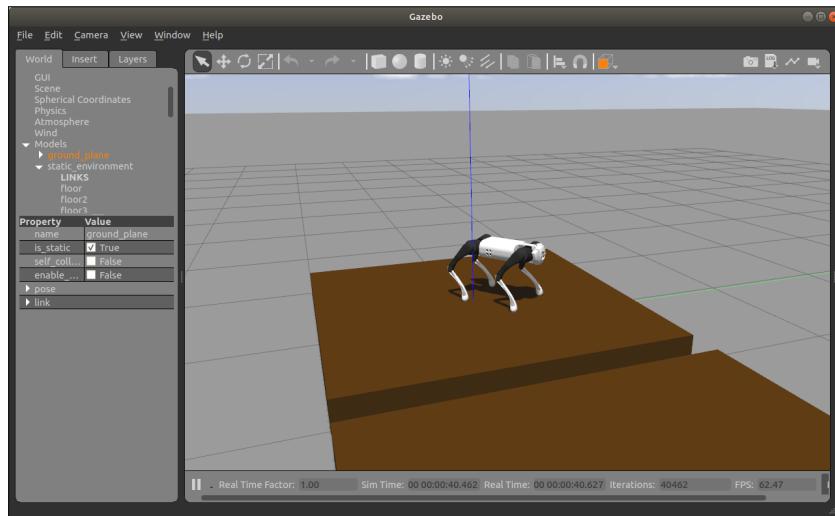


Abbildung 3.10: Ausschnitt aus dem Programm *Gazebo* zur Visualisierung des *Go1*

bung geladen. Um die Motoren zu aktivieren bedarf es einer weiteren ROS *Node*, die mittels `rosrun unitree_controller unitree_servo` gestartet wird. Der Roboter wird nun in den stehenden Zustand versetzt. Um Bewegungen des Roboters zu simulieren können nun weitere *Nodes* verwendet werden, so z. B. `rosrun unitree_controller unitree_external_force`. Es können nun mithilfe der Pfeiltasten äußere Einflüsse wie ein Stoßen erzeugt werden. Testfunktionen für z. B. SLAM in der Simulation sind seitens *Unitree* nicht vorhanden und müssen eigenständig entwickelt werden.

3.2.4 Zusätzliche Ressourcen für den Go1

Einen ersten Einstieg in die Handhabung des *Go1* liefert die Website <https://www.docs.quadruped.de/>. Hier finden sich neben der Dokumentation weitere Handbücher, beispielsweise für den Roboterarm *Z1*. Der Umfang erstreckt sich dabei von einem Überblick über den Roboter bis hin zu einzelnen Schritten für die Simulation des Roboters. Dazu findet sich unter der Adresse <https://www.docs.quadruped.de/projects/go1/html/operation.html> eine Übersicht zu verschiedenen Handbüchern, so beispielsweise zur Verwendung der Kamera SDK.

Das offizielle GitHub Repository von *Unitree* findet sich unter <https://github.com/unitreerobotics>. Dort befinden sich alle wesentlichen SDKs. Dazu gehört zunächst *unitree_ros*, ein Paket für die Simulation des Roboters im *Low-Level-Mode*, also Steuerung von Drehmoment, Position und Winkelgeschwindigkeit der Robotergelenke. Eine Simulation der *High-Level* Funktionalitäten dazu gehört u. a. das Laufen ist nicht enthalten. Die Simulation erfolgt durch *Gazebo* und benötigt *ROS-Melodic* oder *ROS-Kinetic*. Eine weitere Bibliothek ist das *unitree_legged_sdk*. Diese ermöglicht Funktionen um die Steuerung des Roboters zu beeinflussen. Dazu gehören sowohl *High-Level* als auch *Low-Level* Befehle. Das *UnitreecameraSDK* – wie zuvor bereits genauer analysiert – liefert verschiedene Möglichkeiten auf die Stereo-Kameras des Roboters zuzugreifen. Das SDK ermöglicht Streamen von Tiefen- und Farbbildern und stellt Informationen zur Kalibrierung zur Verfügung. Ergänzend zu den bereits erwähnten Bibliotheken gibt es noch eine Reihe weiterer. Dazu gehören das *unitree_actuator_sdk*, um die verbauten Motoren einzeln anzusteuern und die beiden Pakete *unitree_ros_to_real* und *unitree_ros2_to_real*, um mittels ROS Befehle an den Roboter zu senden.

Es ist an dieser Stelle wichtig zu erwähnen, dass die offizielle Dokumentation sowohl auf der Website als auch in den GitHub Repositories, zwar einen guten Einstieg in die Nutzung des Roboters liefert, jedoch zu großen Teilen unvollständig, unverständlich oder nur in der Originalsprache *Chinesisch* vorhanden ist. Ein Blick auf die folglich aufgeführten alternativen Quellen für den Bezug weiterer Informationen ist also ratsam, wenn auch gleichzeitig mit kritischem Blick zu betrachten, da vieles nicht seitens *Unitree* bestätigt noch geprüft wurde.

Erwähnenswert ist hier das Repository *Hangzhou Yushu Tech Unitree Go1 [MAV]*. Im Rahmen dessen wird der *Go1* Roboter anhand mehrerer Gesichtspunkte eingehend untersucht. Unter der *Slack* Gruppe *The Dog Pound animal control for Stray robot dogs* finden sich weitere Enthusiasten und Benutzer, die untereinander eine Rei-

he an verschiedenen Informationen bzgl. des Roboters austauschen, sei es hinsichtlich des internen Aufbaus von Motoren, Hardware und Empfehlungen zur Handhabung der SDK's oder eigenständigen Implementierungen zur Erweiterung der Funktionalität. Das Forum <https://community.droneblocks.io/> ist in dieser Hinsicht ebenfalls eine gute Anlaufstelle. Zuletzt ist der Youtube Kanal von *DroneBlocks* (zu finden unter <https://www.youtube.com/@droneblocks>) sowie das dazugehörigen GitHub Repository <https://github.com/dbaldwin> eine gute Anlaufstelle, um die Kenntnisse zu Möglichkeiten der Programmierung des *Go 1* zu vertiefen.

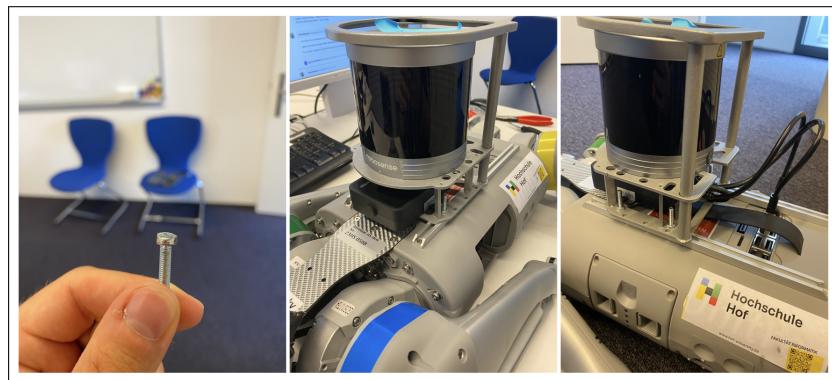
4 Umsetzung

Dieses Kapitel behandelt die Umsetzung des Vorhabens ein intelligentes Navigations- system auf dem *Unitree Go1* mithilfe des *RS-Helios-16P* LiDAR umzusetzen. Dabei werden die einzelnen Schritte wie die Vorbereitung des Roboters, die Auswahl der Ver- suchsumgebung sowie die Testdurchläufe aufbereitet und erläutert. Anschließend werden die verschiedenen Szenarien beschrieben, durch die dann eine Umgebungskarte mithilfe des LiDAR erstellt werden soll.

4.1 Beschreibung des Versuchsaufbaus

Um das Vorhaben umzusetzen wird der LiDAR auf dem Roboter montiert. Dafür dient ein Schienensystem auf dem Rücken. Erwähnenswert ist an dieser Stelle, dass seitens des Herstellers keinerlei Anleitung zur Befestigung des Sensors vorhanden ist. Die Kon- struktion basiert daher auf eigenen Ideen. Die dafür eingesetzten Schrauben werden mit dem Kopf in das Schienensystem geschoben. Um ein Drehen des Schrauben-Kopfes zu vermeiden, werden einige Millimeter seitlich abgeschliffen. Anschließend wird der LiDAR auf die Schrauben aufgesteckt und mit Schrauben-Muttern festgezogen. Zuletzt wird der Sensor mit den Schnittstellen des Roboters, dem *RJ-45* und *XT-30U* zur Stromausgabe, verbunden. Abbildung 4.1 zeigt den fertigen Aufbau.

Als nächstes werden die Grundeinstellungen des LiDAR konfiguriert. Dafür wird seitens des Herstellers eine Weboberfläche zur Verfügung gestellt, auf die mithilfe der IP-Adresse des Sensors zugegriffen werden kann. Für die Versuche wurde die IP auf 192.168.123.60 festgelegt. Anhang A.8 zeigt einen Ausschnitt aus den Einstellungen der Weboberfläche. Der Parameter *Destination IP Address* wurde auf die IP-Adresse des *NVIDIA Xavier NX* festgelegt. MSOP und DIFOP Ports bleiben unverändert.

Abbildung 4.1: Montage des LiDAR Sensors auf dem *Go1*

Um ein möglichst diverses Testumfeld zu erhalten, wurden zwei Szenarien für die Versuche gewählt, in einem Gebäude sowie im Außenbereich. Als *Indoor* Teststrecke dient das Foyer des Institut für Informationssysteme der Hochschule Hof. Der *Outdoor*-Test befindet sich unmittelbar um den Alfons-Goppel-Platz der Hochschule Hof. Beide Teststrecken bieten geeignete Umgebungen, wie Treppen oder unwegsames Gelände und Steigungen. Zu Beginn der Versuche wurde sichergestellt, dass der Akku des Roboters voll aufgeladen ist. Tabelle 4.1 nennt wesentliche Eckdaten der gewählten Versuche. Die Durchschnittsgeschwindigkeit berechnet sich dabei aus der geschätzten zurückgelegten Strecke und der Aufnahmemezeit.

Szenario	Zeit t (Sekunden)	\sim Distanz m (Meter)	Geschwindigkeit v^- (m/s)
Indoor (IISYS-Foyer)	249	124	0,497
Outdoor (HS-Hof)	615	585	0.951

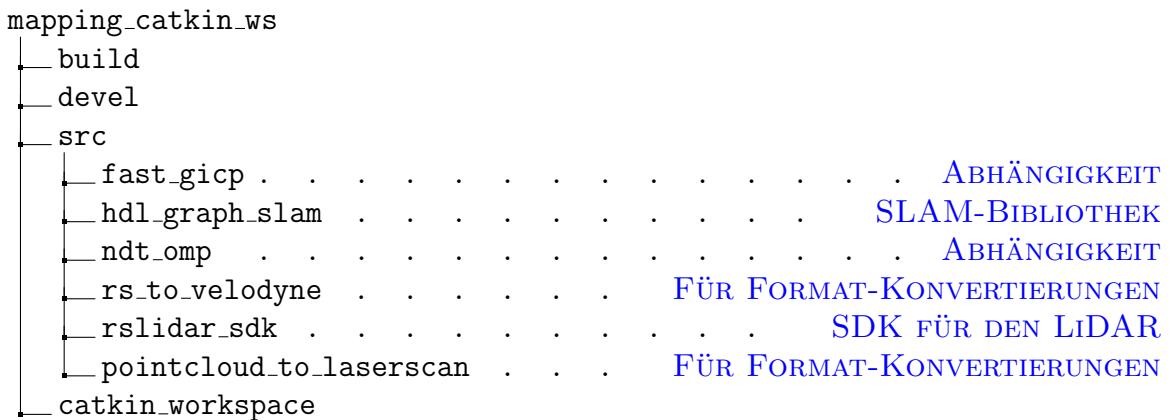
Tabelle 4.1: Kenndaten der zwei Messszenarien

Deutlich wird hier, dass der *Mapping*-Prozess im Innenbereich mit einer wesentlich niedrigeren Durchschnittsgeschwindigkeit v^- auffällt. Hingegen ist im Außenbereich die Durchschnittsgeschwindigkeit nahezu um mehr als das doppelte erhöht. Eine Auswertung der beiden Szenarien erfolgt im Anschluss in Kapitel 5. Ein Hinweis gilt der Distanz. Aufgrund dessen, dass keine genaue Ermittlung der zurückgelegte Strecke möglich war, handelt es sich hier um eine Schätzung.

4.2 Versuchsdurchführung

Für die Implementierung des Echtzeit-SLAM-Algorithmus auf dem Roboter wurde *hdl_graph_slam*³⁴ verwendet. Dabei handelt es sich um eine *Open-Source* ROS Bibliothek für 3D-LiDAR Anwendungen. Es basiert auf einem *3D-Graph-SLAM* mit NDT-ähnlichen *Scan-Matching*-Verfahren zur Bestimmung der Bewegungsbahn. Dabei ist *hdl_graph_slam* für sechs Freiheitsgrade geeignet. Neben dem *Scan-Matching* können auch andere Sensoreingänge wie IMU oder GPS als Randbedingungen für die Bewegungsbestimmung verwendet werden. Da *Unitree* keine direkte ROS Schnittstelle für das Abgreifen der IMU Daten zur Verfügung stellt, entfällt diese Option.

Für die Umsetzung auf dem Roboter wird zunächst ein eigener *catkin-Workspace*³⁵ auf dem *NVIDIA Xavier NX* des Roboters eingerichtet. Dieses hat folgende Struktur:



Weiterhin werden einige Änderungen im SDK des LiDAR Sensors vorgenommen. Als Format-Typ für die ausgesendeten Punkte, können zwei verschiedene Arten gewählt werden. **XYZI** und **XYZIRT**. Tabelle 4.2 fasst die Aufzeichnungsmöglichkeiten der Messpunkte zusammen.

Deskriptor	Beschreibung
X,Y,Z	Räumliche Koordinaten des LiDAR-Punktes
I	Rückstrahl-Intensität des Messpunktes
R	Ringnummer, aus die der Punkt stammt
T	Zeitstempel

Tabelle 4.2: Beschreibung der Aufzeichnungsmöglichkeiten der Messpunkte

³⁴https://github.com/koide3/hdl_graph_slam

³⁵catkin ist ein CMake basiertes Build-Tool für ROS

Da letztere Einstellung wesentlich mehr Informationen bereitstellt, wird das Dateiformat in der Datei unter `src/rslidar_sdk/CMakeLists.txt` angepasst. Weiterhin wird die Art des *Build-Tools* auf CATKIN gesetzt.

```
1 set(POINT_TYPE XYZIRT)
2 set(COMPILATION_METHOD CATKIN)
```

Listing 4.1: Änderungen im LiDAR-SDK

Unter `src/rslidar_sdk/config/config.yaml` sind weitere Einstellung vorzunehmen. Die Parameter `send_packet_ros` sowie `send_point_cloud_ros` werden auf `false` und `true` gesetzt. Der `lidar_type` bekommt das verwendete Modell, `RSHELIOS_16P` zugeordnet. Zuletzt kann der Name des ROS *Topic* festgelegt werden, über die der LiDAR die Daten der Punktwolke veröffentlichen soll. Das Aussenden der Messung erfolgt über das *Topic* `/rslidar_points`.

```
1 send_packet_ros: false
2 send_point_cloud_ros: true
3 ...
4 lidar:
5   - driver:
6     lidar_type: RSHELIOS_16P
7   ...
8   ros_send_point_cloud_topic: /rslidar_points
```

Listing 4.2: Änderungen der `config.yaml` im LiDAR-SDK

Da die aufgezeichneten Punktwolken des *RS-Helios-16P* Versuchen nach *Not a Number (NaN)* Werte enthalten und die meisten Bibliotheken aufgrund dessen Fehlermeldungen ausgeben und auf *Velodyne* Sensoren ausgerichtet sind, empfiehlt sich eine Konvertierung des Formats. Das Paket `rs_to_velodyne`³⁶ übernimmt diese Aufgabe und entfernt gleichzeitig NaN Werte. Während einiger Testläufe kam es bei der Konvertierung der Formate oft zu einer Fehlermeldung (siehe Anhang A.9). Bei Betrachtung des Codes zur Konvertierung in der Datei `rs_to_velodyne.cpp` ließ sich folgender Fehler finden (siehe Listing 4.3):

³⁶https://github.com/HViktorTsoi/rs_to_velodyne

```

1 POINT_CLOUD_REGISTER_POINT_STRUCT(RsPointXYZIRT,
2     (float, x, x)(float, y, y)(float, z, z)(float, intensity, intensity)
3     (uint16_t, ring, ring)(double, timestamp, timestamp))
4 // Änderung des Typ von uint8_t, intensity, intensity auf float

```

Listing 4.3: Änderungen der *rs_to_velodyne.cpp* Datei

Die Variable `intensity` war zuvor als Typ `uint16_t` definiert. Das Format des verwendeten LiDAR ist jedoch vom Typ `float`. Der Ursprung liegt vermutlich darin, dass die Bibliothek einige Zeit nicht weiterentwickelt wurde und inzwischen Änderungen im Format seitens des Herstellers geschehen sind. Die Änderung des VariablenTyps sorgt dafür, dass die Fehlermeldung behoben wird. Eine eigens angelegte *Launch* Datei vereinfacht den Startprozess, sodass die Befehle nicht jedes mal einzeln eingegeben werden müssen.

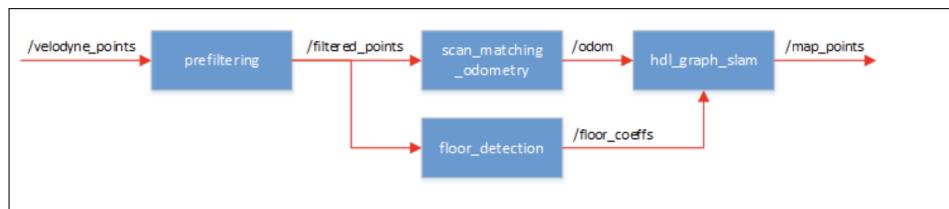
```

1 <launch>
2   <node pkg="rs_to_velodyne" name="rs_to_velodyne" type="rs_to_velodyne"
3     args="XYZIRT XYZIRT" output="screen">
4   </node>
5 </launch>

```

Listing 4.4: Launch Datei für die *rs_to_velodyne* Node

Als nächstes erfolgt die Konfiguration des *hdl_graph_slam* Algorithmus. Es folgt eine kurze Erläuterung der Funktionsweise. Relevant sind vier *Nodes*. Abbildung 4.2 zeigt den Ablauf des Algorithmus und die relevanten Knoten.

Abbildung 4.2: Diagramm der *hdl_graph_slam* Nodes aus [koi23]

Die eingehende Punktwolke unter dem *Topic* `/velodyne_points` wird zunächst gefiltert. Das geschieht in der *prefiltering Node*. Hier können mithilfe zweier Schwellwerte `distance_near_thresh` und `distance_far_thresh` Messpunkte die zu nah oder weit

entfernt sind, herausgefiltert werden. Weiterhin können sogenannte *Outlier*³⁷ mithilfe zweier Verfahren entfernt werden. Dafür gibt es ein Radius-basiertes Verfahren und ein statistisches Verfahren. Erstere Methode berücksichtigt Nachbarn eines Punktes innerhalb des in `radius_radius` festgelegten Radius. Liegt der Wert der Mindestanzahl der Nachbarn unterhalb des Wertes `radius_min_neighbours`, so wird der Punkt entfernt. Hingegen können bei der statistischen Methode mithilfe einer Standardabweichung `statistical_stddev` *Outlier* ermittelt werden.

Das in 2.1.5 beschriebene *Scan-Matching* geschieht in der `scan_matching_odometry Node`. Die Sensorposition wird durch Anwendung von *Scan-Matching* aus aufeinanderfolgenden Messungen geschätzt, woraus die Bewegungsbahn des Roboters bestimmt wird. Für das *Scan-Matching* können die in Abschnitt 2.1.5 beschriebenen Verfahren ICP, GICP, NDT und VGICP ausgewählt und die entsprechenden Parameter konfiguriert werden. Da in Gebäuden meist eine ebene Fläche als Boden anzunehmen ist, wird mithilfe der `floor_detection_node` eine weitere Begrenzung eingeführt. Diese optimiert den Graphen dahingehend, dass der erkannte Boden für alle erfassten *Scans* auf der gleichen Ebene liegt. Diese Berechnung der Ebene erfolgt mittels Random Sample Consensus (RANSAC) [FB81].

Folgende Einstellungen wurden für die Verwendung des Algorithmus im Innenraum vorgenommen. In der *Launch*-Datei:

```

1 <launch>
2 ...
3 <arg name="enable_floor_detection" default="true" />
4 <arg name="enable_gps" default="false" />
5 <arg name="enable_imu_acc" default="false" />
6 <arg name="enable_imu_ori" default="false" />
7 ...
8 <arg name="points_topic" default="/velodyne_points" />
9 ...
10 <param name="tilt_deg" value="0.0" />
11 <param name="sensor_height" value="0.5" />
12 <param name="height_clip_range" value="0.5" />
13 ...
14 </launch>
```

Listing 4.5: Ausschnitte der *Launch* Datei für den `hdl_graph_slam` Algorithmus

Funktionen hinsichtlich der Berücksichtigung von GPS sowie IMU wurden abgeschaltet.

³⁷dt. Ausreißer

ten, da diese nicht vorhanden sind. Die eingehende *Message*³⁸ der Punktwolke vom Typ `sensor_msgs/PointCloud2` wird auf `velodyne_points` festgelegt. Das entspricht dem Typ, den die zuvor eingestellte `rs_to_velodyne` *Node* veröffentlicht. Weiterhin ist anzunehmen, dass der LiDAR eben auf dem Roboter platziert ist. Der Neigungsgrad `tilt_deg` wird daher auf 0° gesetzt. Die gemessene Höhe des LiDAR liegt bei $\sim 0.5\text{m}$. Der letzte relevante Parameter `height_clip_range` gibt einen Schwellwert für die Erkennung des Boden an. Dabei werden nur Punkte im Bereich von `sensor_height + height_clip_range` bis `sensor_height - height_clip_range` für die Bodenerkennung berücksichtigt. Die Einrichtung der einzelnen *Nodes* ist damit abgeschlossen. Folgend wird der Prozess der Aufnahme und Auswertung geschildert.

Mithilfe von `roscore` wird die *Master Node* initialisiert und gestartet. Anschließend wird das `rslidar_sdk` gestartet, um die Punktwolke zu übertragen. Als nächstes wird diese in das *Velodyne*-Format umgewandelt und die *Point-Cloud* unter dem Namen `velodyne_points` veröffentlicht. Der Befehl `rosbag record /velodyne_points` zeichnet die Daten auf. Für eine vereinfachte Bedienung wurde ein *Shell-Script* auf dem Roboter erstellt, um die Befehle automatisch zu starten.

```

1 source mapping_catkin_ws/devel/setup.bash &
2 roscore &
3 roslaunch rslidar_sdk start.launch &
4 roslaunch rs_to_velodyne rstovelodyne.launch &
5 rosbag record /velodyne_points

```

Listing 4.6: Start- und Aufnahmeprozess

Während beiden Szenarien wurde der Roboter mithilfe der Fernbedienung gesteuert. Zur Verifikation der aufgezeichneten Datei kann der Befehl `rosbag info` verwendet werden. Die hier auffälligen Datumseinträge sind von untergeordneter Bedeutung. Hierbei handelt es sich um das vom Hersteller zuletzt eingestellte Datum des *NVIDIA Xavier NX* auf dem Roboter.

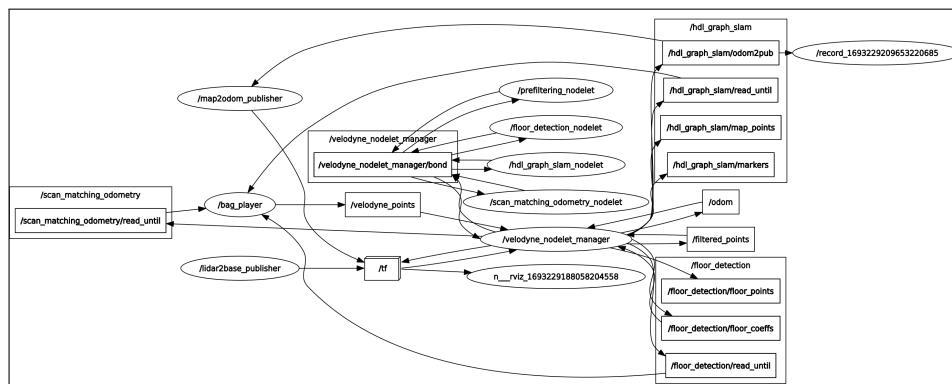
Nachdem sichergestellt wurde, dass die Datei fehlerfrei ist, kann der Algorithmus auf der `bag` Datei ausgeführt werden. Der *Mapping*-Prozess lässt sich mithilfe von *RViz* gut visualisieren. Das Framework bietet dabei zwei unterschiedliche *Launch*-Dateien abhängig davon, ob ein *Indoor* oder *Outdoor* Szenario betrachtet wird. Verändert werden, je nach Szenario, die *Scan-Matching* Parameter. Diese können in den *Launch*-Dateien zugeschnitten werden.

³⁸dt. Nachricht

	<i>iisys.bag</i>	<i>outdoorhs.bag</i>
duration	4:09s (249s)	9:45s (585s)
size	1.9 GB	2.8 GB
messages	2497	4660
types	sensor_msgs/PointCloud2	sensor_msgs/PointCloud2
topics	/velodyne_points	/velodyne_points

Tabelle 4.3: Kenndaten der erzeugten *bag* Dateien

Die erzeugten Karten werden mithilfe des Befehls `rosservice call /hdl_graph_slam/save_map`

Abbildung 4.3: *Nodes* und *Topics* des *rqt* Graphen für den *hdl_graph_slam* Algorithmus

gespeichert. Die beiden Abbildungen 5.2 und 5.4 zeigen die durch den SLAM-Algorithmus erzeugten 3D-Umgebungen. Im unteren rechten Eck sind die durch den Algorithmus erstellten Wegpunkte zu sehen. Nachdem die Implementierung abgeschlossen ist, folgt nun eine Analyse des SLAM Verfahren auf Basis der gewählten Szenarien. Abbildung 4.3 zeigt den Berechnungsgraphen mithilfe der `rqt_graph` Funktion von ROS. In diesem werden alle aktiven *Nodes* und *Topics* während des SLAM Verfahren angezeigt.

```

1  rosparam set use_sim_time true
2  roslaunch hdl_graph_slam hdl_graph_slam_501.launch
3  roscd hdl_graph_slam/rviz
4  rviz -d hdl_graph_slam.rviz
5  rosbag play --clock iisys.bag

```

Listing 4.7: Starten des *hdl_graph_slam* auf der bag Datei

Videos des *Mapping* Verfahren sind für den Innenbereich unter <https://www.youtube.com>.

com/watch?v=10QQdSywcgI&t=7s und für den Außenbereich https://www.youtube.com/watch?v=eKyQpy1L__Q&ab_channel=Jonas zu finden.

5 Ergebnisanalyse

Der wesentliche Inhalt dieses Kapitels ist die Aufbereitung und Interpretation der gewählten Szenarien für den SLAM Algorithmus. Zuletzt wird auf Einschränkungen sowie Hindernisse, die während der Testversuche auftraten, eingegangen.

5.1 Auswertung der Szenarien

Für die Auswertung der *Scan-Matching* Verfahren wurden die Algorithmen *FAST_GICP* und *FAST_VGICP* betrachtet. Der *FAST_GICP* Algorithmus wird von den Entwicklern des *hdl_graph_slam* Pakets empfohlen und ist gleichzeitig als Standardeinstellung vorkonfiguriert. Da die Literatur von *NDT_OMP* Algorithmen gute Ergebnisse mit der Verwendung eines mehrstrahligen LiDAR verspricht, wird dieser Ansatz ebenfalls betrachtet [Car+21]. Die Konfiguration der Standartparameter zeigte gute Ergebnisse. Dahingehend wurden keine Änderungen vorgenommen.

Aufgrund der Abstinenz von Echtheitsdaten, begrenzen sich die Auswertung auf visuelle Eindrücke und den Vergleich der erstellten Umgebungskarten untereinander. Mithilfe des Paket *evo*³⁹ würden sich wesentlich detailliertere Auswertungen vornehmen lassen. Durch den Abgleich mit *Ground-Truth* Daten könnten die Abweichungen der Laufbahnen sowie statistische Kenndaten wie *Absolute Pose Error (APE)* und *Relative Pose Error (RPE)* ermittelt werden.

³⁹zu finden unter: <https://github.com/MichaelGrupp/evo>

5.1.1 Betrachtung Indoor-Szenario

Wie bereits erwähnt wurde als Teststrecke für Innenräume das Foyer des Instituts für Informationssysteme der Hochschule Hof (IISYS) gewählt. Es wird angenommen, dass der Boden eine eben Fläche besitzt und die Wände im 90° Winkel zum Boden stehen. Das Foyer hat viele Fensterfronten, so dass auch Teile des Innenhofs und Vorlesungsräume vom LiDAR erfasst werden (siehe Abbildung 5.2). Messungen durch Fensterglas können zu nicht reproduzierbaren Messungen oder einer geringeren Genauigkeit führen. Dies liegt daran, dass Glas entweder eine Reflexion verursachen kann oder der Laserstrahl so gebrochen wird, dass er seinen Winkel oder seine Richtung ändert [AAP08].

Das Verfahren zur *Loop-Closure* hat ebenfalls funktioniert, wie Abbildung 5.2 zu entnehmen ist. Genauere Informationen über das *Loop-Closure* Verfahren sind in [KMM19] zu finden. Abbildung 5.1 zeigt den SLAM Graphen der Innenraum-Teststrecke sowie die erzeugte 3D-Umgebungskarte und ein Foto des Foyers. Die Verbindungen des Graphen an manchen Knoten bedeuten eine erfolgreich erkannte *Loop-Closure*.

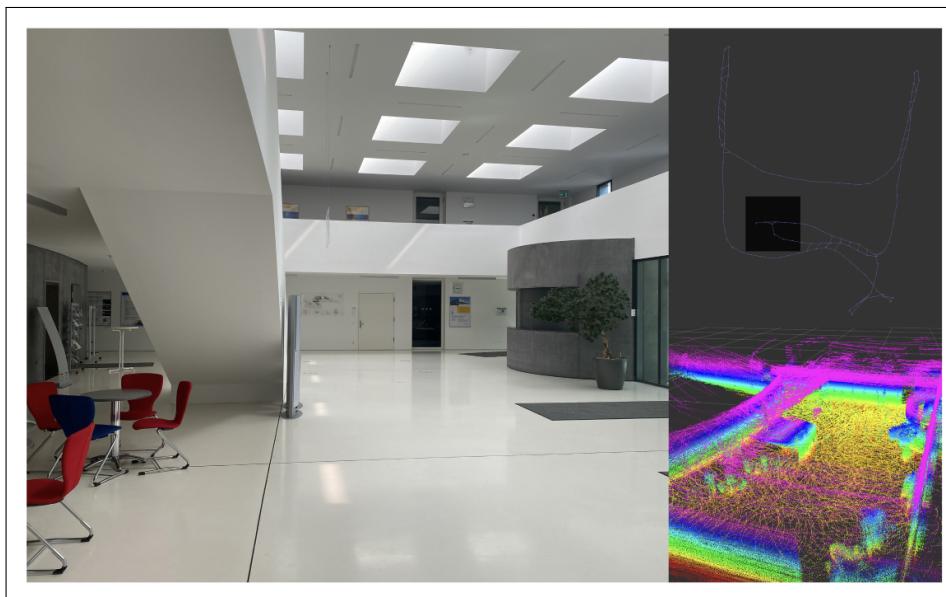


Abbildung 5.1: Foto mit dazugehöriger ermittelte 3D-Punktwolkenumgebung des Foyers und SLAM Graphen des IISYS

Die erzeugte 3D-Karte im Innenraum ist durchaus gelungen. Die besten Ergebnisse wurde mithilfe des *FAST_GICP* und *FAST_VGICP* Algorithmus erzielt. Abbildung 5.2 zeigt ein Satellitenbild zusammen mit verschiedenen Ansichten der durch *FAST_GICP* erstellten Umgebungskarte.

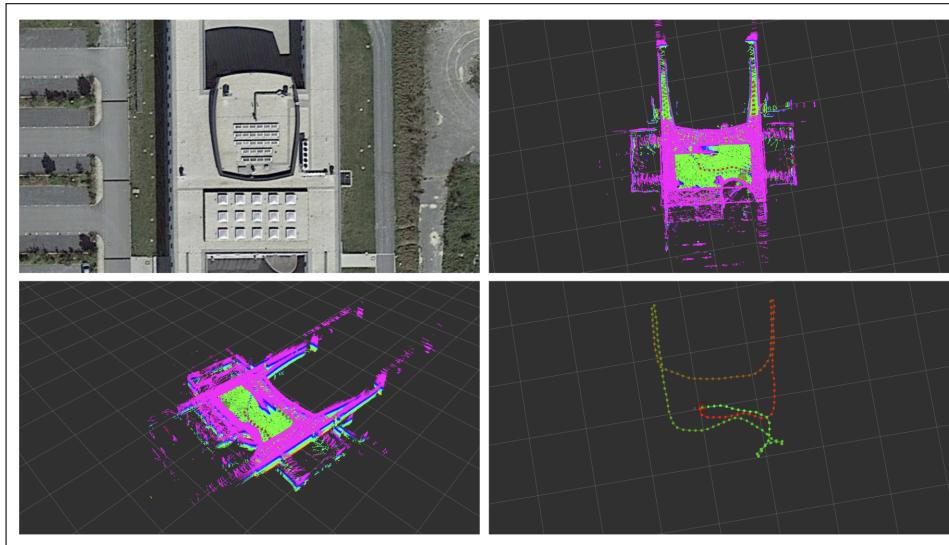


Abbildung 5.2: Verschiedene Ansichten der gemappten Umgebung des *Indoor-Szenarios (FAST_GICP)*

Während die zuvor erwähnten *Scan-Matching* Verfahren durchaus verwertbare Resultate lieferten, da die Qualität und Umgebungsdarstellung der Karte durchaus der Realität entsprechen, war die erstellte Karte durch *HDT_OMP* größtenteils unbrauchbar. Es erfolgte eine Verschiebung und Rotation der Punktwolke, was eine fehlerhafte Erzeugung der Umgebungskarte zufolge hatte. Abbildung 5.3 zeigt die durch *HDT_OMP* erstelle Karte.

Das Vorhandensein der Bodenerkennung hatte ebenfalls Auswirkungen auf die Messungen und resultierenden Karten. Während bei angeschalteter Bodenerkennung keinerlei Schrägen oder Steigungen auftraten, war das bei abgeschalteter Bodenerkennung nicht der Fall. Hier ergaben die Messungen in manchen Situationen Werte, als würde der Roboter eine Steigung oder Neigung ablaufen obwohl der Untergrund eben ist.

Um die Genauigkeit der erstellten Karte zu überprüfen, wird sie weiterhin auf eine *Rechtwinkligkeit* hin untersucht. Zu diesem Zweck werden die erkannten Wände mit den realen Wänden abgeglichen. Wird davon ausgegangen, dass das Gebäude rechteckig ist, kann festgehalten werden, dass an manchen Stellen Abweichungen auftreten. Die mit *hdl_graph_slam* erstellte Karte ist leicht verzerrt. Die Verwendung anderweitiger Verfahren oder zusätzliche Randbedingungen führte nicht zu signifikanten Verbesserungen.

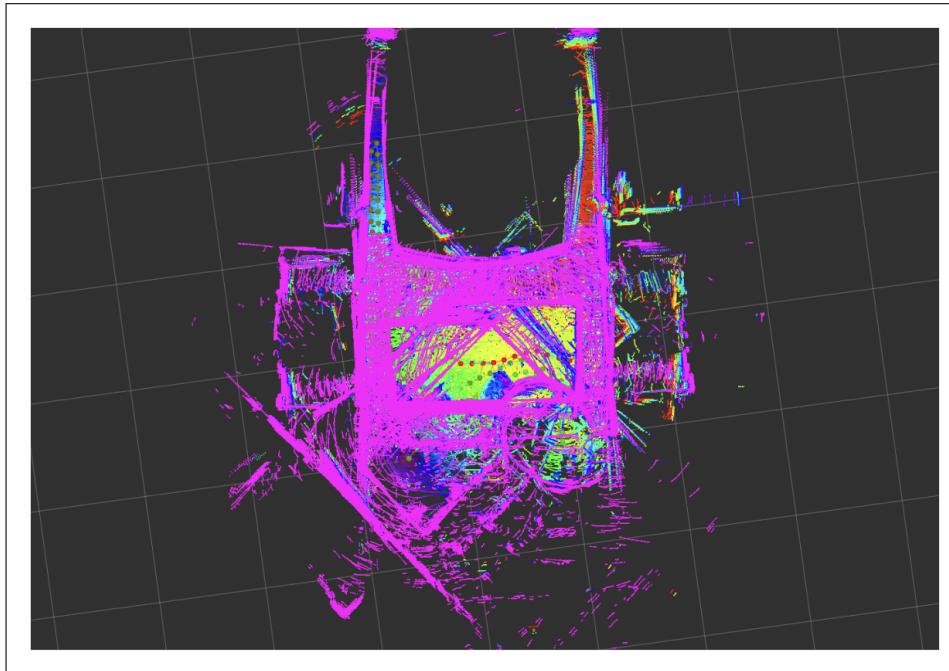


Abbildung 5.3: Fehlerhafte Umgebungskarte erzeugt durch *HDT_OMP*

5.1.2 Betrachtung Outdoor-Szenario

Für den Außenbereich wird eine zurückgelegte Strecke von ~ 560 Metern betrachtet. Diese erstreckt sich vom Haupteingang des Instituts für Informationssysteme über eine Treppe hinab auf den Parkplatz bis hin zum Hauptgebäude der Hochschule Hof. Dazu kann Abbildung 5.4 betrachtet werden. Sie zeigt eine Satellitenansicht mit eingezeichneter Strecke sowie verschiedene Ansichten der erstellen Umgebungskarte mithilfe *FAST_GICP*. Dazu befindet sich unten links im Bild die durch den Algorithmus ermittelte Bewegungsbahn. Während der *Outdoor*-Messungen war das Wetter sonnig, sodass keine großartigen Störungen auftraten. Jedoch ist erwähnenswert, dass es besonders warm ($\sim 30^\circ\text{C}$) war, was eventuell Rückschlüsse auf die kurze Akkulaufzeit (siehe Kapitel 5.2) zulässt.

Auch im Außenbereich wurden die besten Ergebnisse mit *FAST_GICP* und *FAST_VGICP* erreicht. Hingegen kam es bei der Verwendung von *HDT_OMP*, wie bereits im Innenbereich, zu Verzerrungen und Rotationen der Karte. In Betrachtung der *Floor-Detection* wurden im Außenbereich ebenfalls Test durchgeführt. Abbildung 5.5 zeigt die Steigung unmittelbar links des Instituts für Informationssysteme mit angeschlossenem Parkplatz. Darüber sind links der passende Ausschnitt der erstellten Umgebungskarte ohne aktivierte Bodenerkennung dargestellt. Oben rechts befindet sich der gleiche Aus-

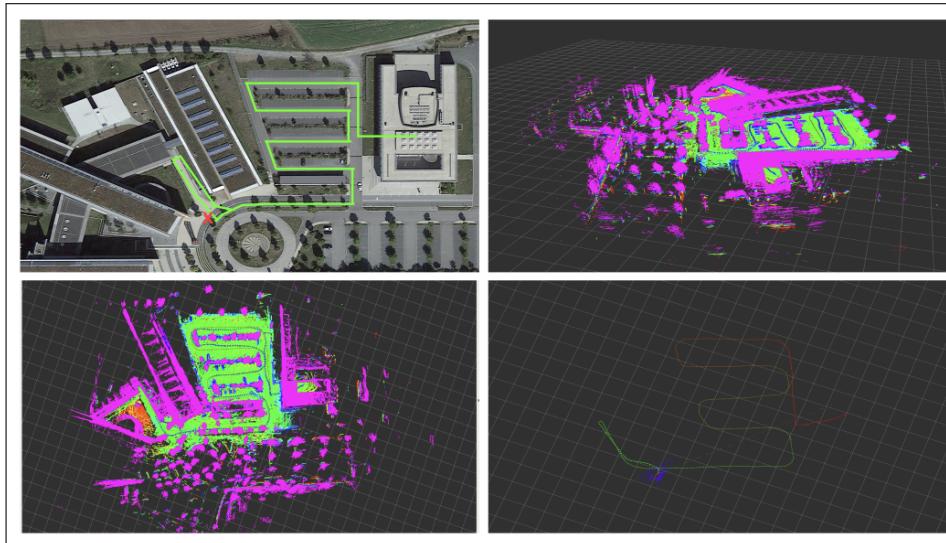


Abbildung 5.4: Verschiedene Ansichten der gemappten Umgebung des *Outdoor*-Szenario (*FAST_GICP*)

schnitt mit aktiverter Bodenerkennung. Es wird deutlich, dass das Deaktivieren der *Floor-Detection Node* für eine recht solide Erkennung der Treppe und Steigung sorgt. Hingegen ist bei aktiverter *Floor-Detection* eine verfälschte Karte entstanden.

Die aktivierte Bodenerkennung sorgt jedoch anderweitig an manchen Stellen für Probleme. Betrachtet wird dazu Abbildung 5.6. Sie zeigt einen Ausschnitt der Karte des Außenbereichs mit aktiverter Bodenerkennung. Zu erkennen ist dabei eine fälschliche Neigung der Umgebung in *Z* Richtung.

Vermutet wird hier, dass während der Roboter die Treppen hinabstieg und somit der LiDAR ebenfalls kurzzeitig geneigt war, eine falsche Referenzmessung der Umgebung erfolgte. Das führte dazu, dass nachdem der Roboter wieder auf eine geraden Ebene stand und die Messung vorsetzte, fälschlicherweise eine Neigung erzeugt wurde. Die Verwendung anderweitiger *Scan-Matching* Verfahren führte hier ebenfalls zu keinen Verbesserungen.

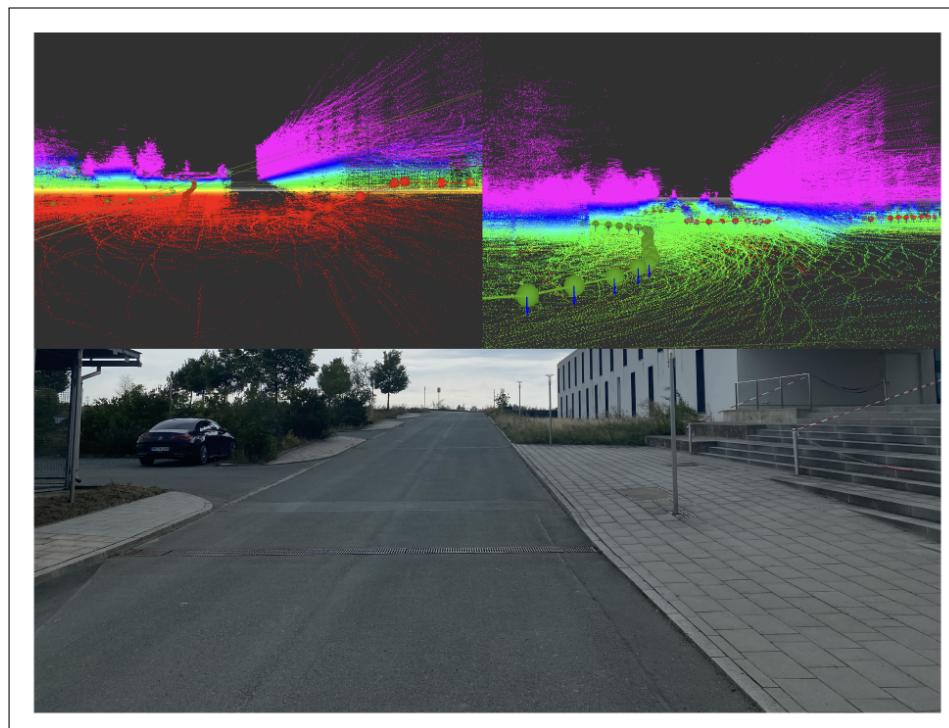


Abbildung 5.5: Vergleich der 3D-Karte für Steigungen und Treppen ohne *Floor-Detection* (links) und mit (rechts)

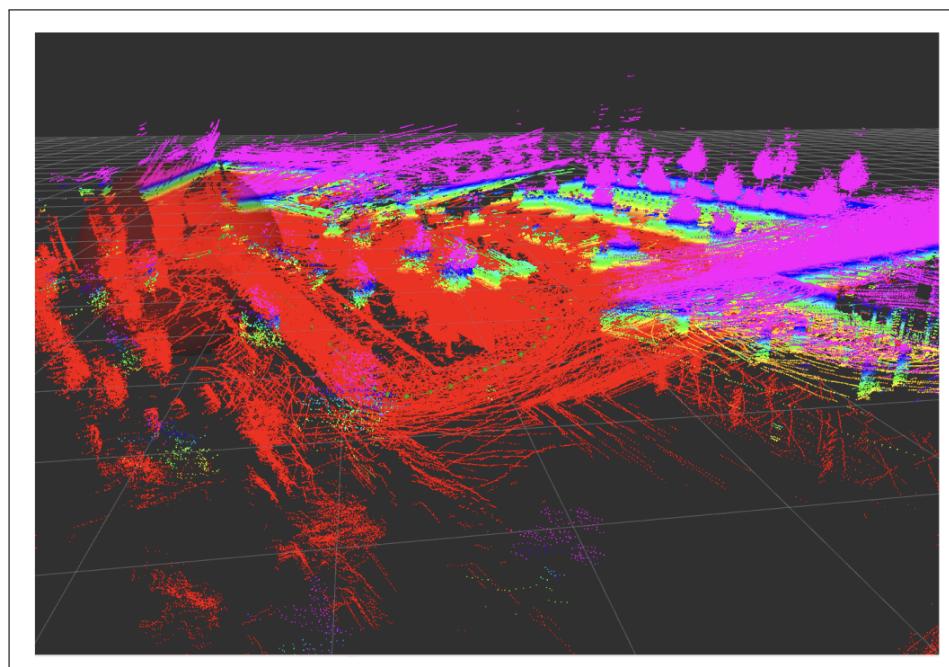


Abbildung 5.6: Darstellung der fehlerhaften 3D-Karte aufgrund geneigter Referenzmessungen

5.1.3 Haupterkenntnisse

Mit den durchgeführten Experimenten wurde dargestellt, dass der Einsatz von mobilen vierbeinigen Robotern mithilfe moderner LiDAR und SLAM Technologie umsetzbar ist. Es müssen einige wichtige Erkenntnisse gesondert betrachtet werden.

Das System zeigt eine große Leistungsfähigkeit und Zuverlässigkeit für Innenraumszenarien unter der Voraussetzung, dass die Bodenebene als Begrenzung, berücksichtigt wird. Andernfalls weichen die Resultate von der eigentlichen Umgebung ab. Als besonders vielversprechender Ansatz stellte sich in Innenräumen der *FAST_GICP* Algorithmus mit aktivierten Bodenbegrenzung heraus. Während der Versuche lieferte diese Konfiguration die besten Ergebnisse. Auch mithilfe *FAST_VGICP* wurden gute Leistungen erzielt. Lediglich das *NDT_OMP* Verfahren stellte sich als meist unbrauchbar was den Endzustand der erstellten 3D-Karte angeht heraus.

Das Einbeziehen der *Floor-Detection* kann jedoch auch für Außenszenarien relevant sein, so beispielsweise bei der Navigation in städtischen Umgebungen oder betonierten Gehwegen, wo ebene Bodenverhältnisse anzunehmen sind. Bei größeren Szenarien, in der die Navigation über mehrere Ebenen erfolgt, muss diese Begrenzung jedoch deaktiviert werden. *FAST_GICP* und *FAST_VGICP* stellten sich in diesem Zusammenhang ebenfalls als geeignet heraus und lieferten verwertbare Ergebnisse. Es ist zu beachten, dass eine alleinige Verwendung der aus dem SLAM resultierenden Odometriedaten in Außenbereiche jedoch meist unzureichende Ergebnisse liefert. Eine Einbeziehung zusätzlicher Daten wie aus Kameras, IMU oder GPS Daten ist also wesentlich für den Erfolg derartigen Szenarien.

5.2 Einschränkungen des Systems

Problematisch stellte sich während der Aufnahme des *Outdoor*-Szenarios die Akkulaufzeit dar. Der Akku wurde zuvor voll aufgeladen. Während der Aufnahme war der *Stair-Mode* des Roboter zum Überwinden von Treppen an manchen Stellen aktiviert. Weiterhin lief der Roboter mit einer wesentlich höheren Durchschnittsgeschwindigkeit, wie aus Tabelle 4.1 hervorgeht. Das *Mapping* musste im Außenbereich nach ~ 15 Minuten beendet werden, da der Roboter aufgrund des niedrigen Akkustand zusammenbrach. Ein weiteres Problem stellte die Konnektivität dar. Da der *NVIDIA Xavier NX* keine direkte Ethernet-Verbindung zur Verfügung stellt und der *RJ-45* Port auf dem Rücken

des Roboters mit dem LiDAR verbunden war, bestand die einzige Möglichkeit darin, den Aufnahmeprozess über den *Raspberry Pi* zu starten, was jedoch zu extrem hohen *Ping* Zeiten führte und somit ungünstig zu bedienen war.

Für die Auswertung hinderlich ist die Tatsache, dass keine *Ground-Truth*⁴⁰ Daten vorhanden sind. Der *Go1* bietet keine direkte Möglichkeiten Daten der IMU abzugreifen. Diese wären jedoch förderlich als zusätzliche Rahmenbedingungen für den *hdl_graph_slam* Algorithmus um die Genauigkeit zu verbessern. Es ist anzunehmen, dass umso mehr Begrenzungen (IMU, GPS, *Floor-Detection*) vorhanden sind, desto besser die Ausgabe der berechneten Umgebungskarte wird. GPS Lösungen oder *Motion-Capture* Aufnahmen zur Aufnahme von Echtheitsdaten wären dahingehend ratsam.

Für zukünftige Echtzeit-Anwendungen muss eine konstante Verbindung mit dem Netzwerk der Hochschule Hof vorhanden sein. Während der Versuche stellte sich heraus, dass die Verbindung instabil war oder teilweise nur sehr schwache Signale mit hohen Latenzzeiten empfangen wurden. Langfristig muss eine Lösung für die stabile Kommunikation zwischen dem Roboter und dem Netzwerk der Hochschule Hof gefunden werden.

⁴⁰ dt. Echtheitsdaten

6 Fazit

Abschließend werden die wichtigsten Erkenntnisse dieser Arbeit zusammengefasst. Anschließend folgt eine Betrachtung zukünftiger Forschungsmöglichkeiten im Hinblick auf die Verwendung vierbeiniger Robotersysteme.

6.1 Zusammenfassung

Das Ziel dieser Arbeit lag in der Entwicklung eines intelligenten lidarbasierten Navigationssystems für einen vierbeinigen Roboter. Im Vordergrund stand dabei die Untersuchung der Implementierung eines SLAM Verfahren auf dem *Unitree Go1*. Der Anwendungsbereich beschränkt sich auf die Umgebung der Hochschule für Angewandte Wissenschaften Hof sowie dem Institut für Informationssysteme der Hochschule Hof. Eine eigenständige Entwicklung eines SLAM Algorithmus ist nicht vorgesehen. Stattdessen wurde auf eine Reihe bekannter Bibliotheken zurückgegriffen.

Der Aufbau gliedert sich in sechs Kapitel. Zu Beginn wurden Grundlagen im Bereich Robotik, KI und SLAM vorgestellt. Ein kurzer geschichtlicher Umriss führte in das Thema der Robotik ein. Darauf folgte eine Betrachtung gängiger Roboterkomponenten wie Aktoren und Endeffektoren sowie Möglichkeiten Roboter zu programmieren. Als viel eingesetztes *Tool* wurde dabei ROS genauer betrachtet. ROS erleichtert die Entwicklung komplexer Robotersysteme durch eine Art des *Divide and Conquer* Prinzip, bei dem einzelne *Nodes* für eine explizite Aufgabe verantwortlich sind. Eine Kombination dieser *Nodes* ermöglicht die Entwicklung komplexer Gesamtsysteme.

Anschließend wurden Begriffe und Unterschiede von KI, ML und DL erläutert. ML und DL sind dabei Unterkategorien von KI. Vor allem im Bereich des ML gibt es Ansätze die für die Entwicklung von Navigationssystemen relevant sind. Die Verwendung von LiDAR Sensoren und die Verknüpfung intelligenter ML Ansätze sorgt dabei für die

Gewährleistung robuster Lokalisierungs- und Navigationssysteme. Konzepte wie *Graph-SLAM*, bei dem die Bewegungsbahn des Roboters sowie relevante sogenannte *Keyframes* mithilfe eines Graphen dargestellt werden können, unterstützen dieses Vorhaben. Als *Scan-Matching* Methodik wurde der ICP Algorithmus als Basis vieler weiterer darauf aufbauender Methoden genauer betrachtet. Eine Betrachtung der Herausforderungen und verwandten Arbeiten zu vierbeinigen Robotern und deren Einsatzmöglichkeiten rundeten das Kapitel ab.

Ein genauer Einblick in die verwendeten Komponenten des *Go1* sowie LiDAR Sensor folgte. Die Hauptkomponente des *Go1* sind dabei die vier Gliedmaßen, der Korpus und Kopf. Jeweils drei Motor-Gelenk-Systeme an vier Positionen des Körpers sorgen für insgesamt 12-DOF. Mithilfe von auf dem Rücken befestigten Anschläßen, lässt sich auf die im Roboter verbaute Hardware zugreifen. Dazu gehören u. a. ein *Raspberry Pi*, zwei *NVIDIA Jetson Nanos* sowie ein *NVIDIA Xavier NX*. Dazu ist der Roboter mit einer Reihe weiterer Sensoren, wie Ultraschallsensoren oder Stereo-Kamerasystemen ausgestattet. Mithilfe von *RSView* lassen sich die Sensordaten des *RS-Helios-16P* visualisieren. Der LiDAR verfügt dabei über unterschiedliche Betriebseinstellungen, die mithilfe einer Weboberfläche verändert werden können. Eine Simulationsumgebung für den *Go1* ist mithilfe von *Gazebo*, *RViz* und dem *unitree_ros* SDK möglich.

In Kapitel 4 wurde die Montage des LiDAR auf dem Roboter und die Einrichtung sowie Konfiguration des verwendeten *hdl_graph_slam* Frameworks vorgestellt. Dabei bietet der Algorithmus eine Reihe verschiedener *Scan-Matching* Verfahren (ICP, NDT, VGICP). Eine *floor_detection_node* die das *Mappen* in Gebäuden verbessert, indem eine planare Ebene als Boden angenommen wird, kann als zusätzliche Randbedingung aktiviert und deaktiviert werden. Abschließend wurde eine Analyse der Versuche vorgenommen. Dies geschah unter Betrachtung verschiedener Einstellungsmöglichkeiten des *hdl_graph_slam* Pakets. Dabei wurde eine Analyse des SLAM Graphen unter Berücksichtigung verschiedener *Scan-Matching* Verfahren vorgenommen. Als besonders geeignet stellte sich dabei der *FAST_GICP* Ansatz heraus, der in Kombination mit der *Floor-Detection* Bedingung die besten Ergebnisse in Innenräumen aufzeigte. In Außenbereichen beeinflussten vor allem Treppen und Neigungen den Messprozess aufgrund eingeschränkter Sensorik negativ.

SLAM mithilfe von vierbeinigen Robotern anzuwenden ist vielversprechend und bietet robuste Möglichkeiten Umgebung und Lokalisierung des Roboters ohne konkrete Verwendung von IMU- oder Odometriedaten vorzunehmen. Nachteilig sind die schnell zuneige gehende Akkulaufzeit bei Verwendung eines LiDAR. Hier müssen langfristig

anderweitige Lösungsansätze betrachtet werden.

6.2 Zukünftige Forschungsansätze

Die Möglichkeiten hinsichtlich der Weiterentwicklung des *Go1* sind nahezu unbegrenzt. Zunächst sollte sich jedoch auf die vollumfängliche Erforschung der bereits vorhandenen Möglichkeiten des *Go1* fokussiert werden. Die Erweiterung der Konnektivität, Untersuchungen der bereitgestellten SDK's und Ansätze den Roboter weiter in das Umfeld der Hochschule zu integrieren sollten dabei im Vordergrund stehen. Dazu gehört auch eine Betrachtung der in dieser Arbeit erwähnten Fehlermeldungen und Komplikationen die aufgetreten sind (*UnitreeCameraSDK*, Ultraschallsensoren mithilfe von LCM auslesen). Sie sollten behoben werden, um eine reibungslose Einbindung zukünftig zu ermöglichen. Um Echtzeit-Anwendungen mit beispielsweise Monitoring Komponenten auf dem Gesamtsystem zu realisieren, muss für eine stabile Konnektivität gesorgt werden. Die Versuche zeigten, dass das Verbinden auf den Roboter nur sehr umständlich funktionierte. Außerdem war die Verbindung auf das Netzwerk der Hochschule instabil und lieferte hohe Latenzzeiten. Die Sicherstellung einer stabilen Verbindung der jeweiligen Komponenten ist also unabdingbar für weitere Entwicklungen und sollte eine hohe Priorität für zukünftige Erweiterungen des Systems haben.

Langfristig soll der *Go1* in der Lage sein, autonom auf dem Hochschulgelände zu navigieren und dabei Interaktionsmöglichkeiten bieten. Mögliche Arbeiten können sich der Implementierung einer Schnittstelle der Robotersteuerung sowie der KI Software widmen. Dies kann z. B. unter Zuhilfenahme von ROS umgesetzt werden. Eine eingehende Studie der Bewegungsmöglichkeiten des Roboters auf Grundlage der *Unitree SDK's* ist dafür ratsam.

Um die Auswertung zukünftiger SLAM und KI Algorithmen auf dem Roboter zu vereinfachen sollten Daten für den Abgleich erfasst werden. Das beinhaltet das Erstellen von *Ground-Truth* Daten für zukünftige Analysen weiterer Versuche. Umsetzbar wäre dies mithilfe von *Motion-Capture* Verfahren, um die Echtheit der Bahndaten des Roboters zu ermitteln. Auch die Überprüfung und Veränderung des Messprozesses und der Parameter können neue Erkenntnisse hinsichtlich der Verwendung von SLAM Verfahren und vierbeinigen Robotern liefern. Ein Vergleich zu anderen *Mapping* und SLAM Frameworks wie *LIO-LOAM* wäre spannend.

In gleicher Hinsicht müssen Lösungen für die Erkennung von Personen entwickelt wer-

den. Die Vermeidung von Kollisionen kann mithilfe verschiedenster Sensoren erfolgen. Da die mitgelieferte KI Software nach einigen Versuchen unzureichende Ergebnisse lieferte, müssen Alternativen erforscht werden. Mithilfe von *OpenCV* oder *YOLO* Algorithmen ist es möglich schnell effiziente Systeme zu entwickeln. Langfristig sollte hier jedoch eine Fusion der verschiedenen Sensoren erfolgen. Das Zusammenspiel von LiDAR, Ultraschall-sensoren sowie Kameras erhöht die Robustheit und Resilienz autonomer Roboter.

Der *Go1* kann mit einer fast beliebigen Anzahl an Erweiterungen ausgerüstet werden. So lässt sich beispielsweise zusätzlich ein Arm als Endeffektor auf dem Rücken platzieren. Das schafft neue Möglichkeiten hinsichtlich der Interaktion mit seiner Umgebung. So wäre es denkbar, dass der Roboter selbstständig in der Lage ist Türen zu öffnen oder Schließfächer zu bedienen. Denkbar ist auch eine Verwendung im Rahmen des Gebäudemanagement der Hochschule zu Kontrollzwecken.

Der Einsatz von KI Anwendungen ermöglicht weitere denkbare Szenarien. *Cloud* Technologien eröffnen die Möglichkeit die Rechenlast der auf dem Roboter befindlichen Hardware zu reduzieren. Vor allem rechenintensive ML und DL Prozesse sorgen für eine hohe Auslastung der Robotersysteme. Eine Verlagerung dieser Prozesse in die *Cloud* ist also ein folglich logischer Schritt. Vor allem, da im späteren Verlauf eine Verwendung mehrerer Sensorfusion (Kamerasysteme, erweiterte Funktionalität der Sensorik) angestrebt werden sollte. Die begrenzte Rechenleistung der Roboter-Hardware wird dadurch ihre Belastungsgrenze erreichen. Diese Migration auf cloudbasierte Dienste bringt eigene Herausforderungen und viele spannende Ansätze, um das Vorhaben den Roboter auf dem Hochschulcampus einzusetzen, langfristig zu gestalten.

Da ein zweiter *Go1* Roboter vorhanden ist, können Kooperations-Szenarien in naher Zukunft erschlossen werden. Indem mehrere Roboter koordiniert arbeiten, können gemeinsam Aufgaben bewältigt werden. Diese könnten große Gebiete erkunden, das effiziente Sammeln von Daten oder das kollektive Lösen von Herausforderungen, wie dem Transport schwerer Lasten umfassen.

Zuletzt sollten auch ethische und rechtliche Fragen in Betracht gezogen werden. Vor allem hinsichtlich Privatsphäre, Datenschutz und Sicherheit im Kontext der Mensch-Roboter-Interaktion sollten weiterführende Forschungen in Betracht gezogen werden, um eine verantwortungsbewussten Einsatz zu gewährleisten.

A Anhang: Abbildungen

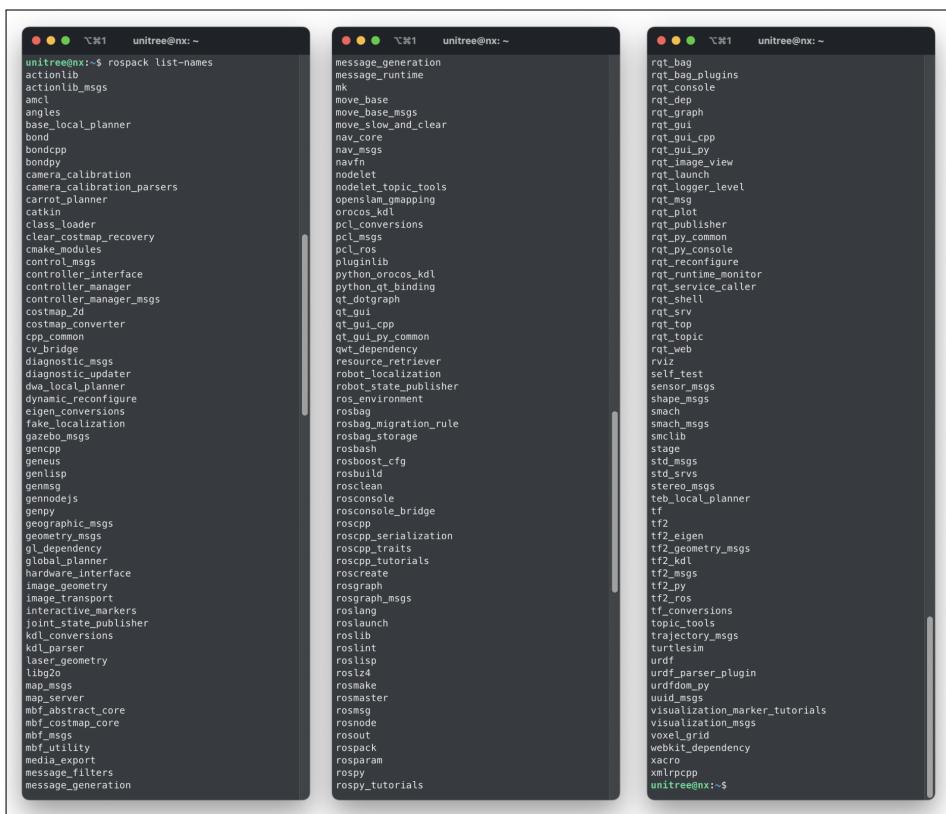


Abbildung A.1: Ausgabe der installierten ROS Packages auf dem *NVIDIA Xavier NX*

```
pi@raspberrypi:~/ros_catkin_ws/src $ ls -la
total 220
drwxr-xr-x 52 pi pi 4096 Jun 22 04:32 .
drwxr-xr-x 6 pi pi 4096 Jul 30 2021 ..
drwxr-xr-x 8 pi pi 4096 Aug 7 2019 actionlib
drwxr-xr-x 5 pi pi 4096 Nov 6 2018 actionlib_msgs
drwxr-xr-x 5 pi pi 4096 Jan 8 2020 angles
drwxr-xr-x 3 pi pi 4096 Oct 2 2018 bond
drwxr-xr-x 4 pi pi 4096 Jul 19 2021 bondcpp
drwxr-xr-x 7 pi pi 4096 Jul 11 2021 catkin
drwxr-xr-x 8 pi pi 4096 Jul 11 2021 class_loader
drwxr-xr-x 4 pi pi 4096 Jul 11 2021 cmake_modules
drwxr-xr-x 12 pi pi 4096 Jul 30 2021 daniel
drwxr-xr-x 11 pi pi 4096 Mar 19 2020 dynamic_reconfigure
drwxr-xr-x 5 pi pi 4096 Jul 11 2021 gen.cpp
drwxr-xr-x 5 pi pi 4096 Jul 11 2021 geneus
drwxr-xr-x 5 pi pi 4096 Jul 11 2021 genlisp
drwxr-xr-x 7 pi pi 4096 Jul 11 2021 genmsg
drwxr-xr-x 5 pi pi 4096 Jul 11 2021 genodejs
drwxr-xr-x 7 pi pi 4096 Jul 11 2021 genpy
drwxr-xr-x 3 pi pi 4096 Nov 6 2018 geometry_msgs
drwxr-xr-x 4 pi pi 4096 Mar 10 2020 kdl_conversions
drwxr-xr-x 2 pi pi 4096 Jul 11 2021 message_generation
drwxr-xr-x 2 pi pi 4096 Jul 11 2021 message_runtime
drwxr-xr-x 5 pi pi 4096 Nov 6 2018 nav_msgs
drwxr-xr-x 6 pi pi 4096 Apr 27 2018 nodelet
drwxr-xr-x 4 pi pi 4096 Apr 27 2018 nodelet_topic_tools
drwxr-xr-x 9 pi pi 4096 Mar 21 2018 orocos_kdl
drwxr-xr-x 4 pi pi 4096 Apr 2 2020 pcl_conversions
drwxr-xr-x 3 pi pi 4096 Mar 21 2018 pcl_msgs
drwxr-xr-x 9 pi pi 4096 Apr 2 2020 pcl_ros
drwxr-xr-x 2 pi pi 4096 Apr 2 2020 perception_pcl
drwxr-xr-x 5 pi pi 4096 Jul 11 2021 pluginlib
drwxr-xr-x 13 pi pi 4096 Jul 11 2021 ros
drwxr-xr-x 22 pi pi 4096 Jul 11 2021 ros_comm
drwxr-xr-x 4 pi pi 4096 Jul 11 2021 ros_comms
drwxr-xr-x 10 pi pi 4096 Jul 11 2021 rosconsole
drwxr-xr-x 6 pi pi 4096 Jul 11 2021 rosCPP_core
drwxr-xr-x 4 pi pi 4096 Jul 11 2021 ros_environment
drwxr-xr-x 10 pi pi 4096 Jun 22 04:32 ros_foxglove_bridge
-rw-r--r-- 1 pi pi 11398 Jul 11 2021 .rosinstall
drwxr-xr-x 6 pi pi 4096 Mar 21 2018 roslint
drwxr-xr-x 12 pi pi 4096 Jul 11 2021 ros_lisp
drwxr-xr-x 5 pi pi 4096 Jul 11 2021 rospack
drwxr-xr-x 8 pi pi 4096 Nov 6 2018 sensor_msgs
drwxr-xr-x 4 pi pi 4096 Oct 2 2018 smclib
drwxr-xr-x 4 pi pi 4096 Jul 11 2021 std_msgs
drwxr-xr-x 9 pi pi 4096 Mar 10 2020 tf
drwxr-xr-x 5 pi pi 4096 Nov 16 2018 tf2
drwxr-xr-x 4 pi pi 4096 Nov 16 2018 tf2_eigen
drwxr-xr-x 6 pi pi 4096 Nov 16 2018 tf2_kdl
drwxr-xr-x 6 pi pi 4096 Nov 16 2018 tf2_msgs
drwxr-xr-x 3 pi pi 4096 Nov 16 2018 tf2_py
drwxr-xr-x 6 pi pi 4096 Nov 16 2018 tf2_ros
drwxr-xr-x 5 pi pi 4096 Mar 10 2020 tf_conversions
pi@raspberrypi:~/ros_catkin_ws/src $ |
```

Abbildung A.2: Ausgabe der installierten ROS Packages auf dem *Raspberry Pi*

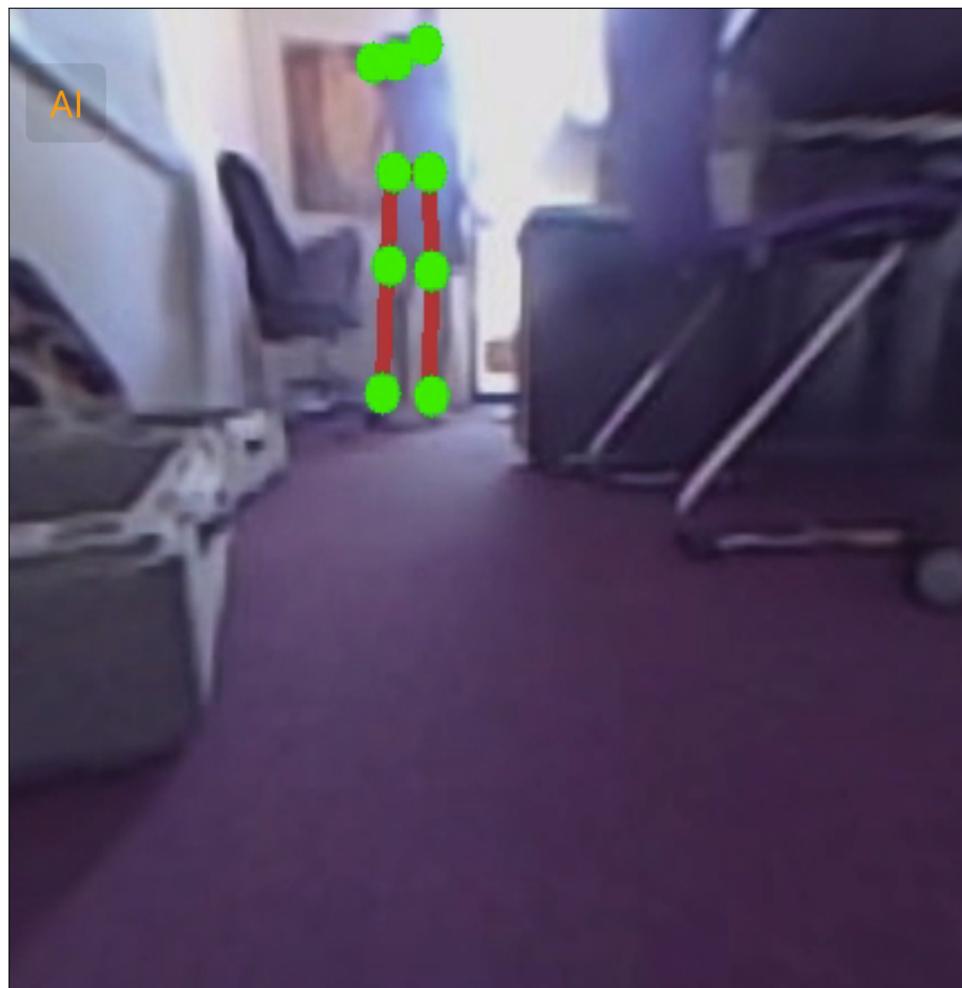


Abbildung A.3: Ausschnitte aus der Web-Applikation und Anzeige der Personenerkennungssoftware

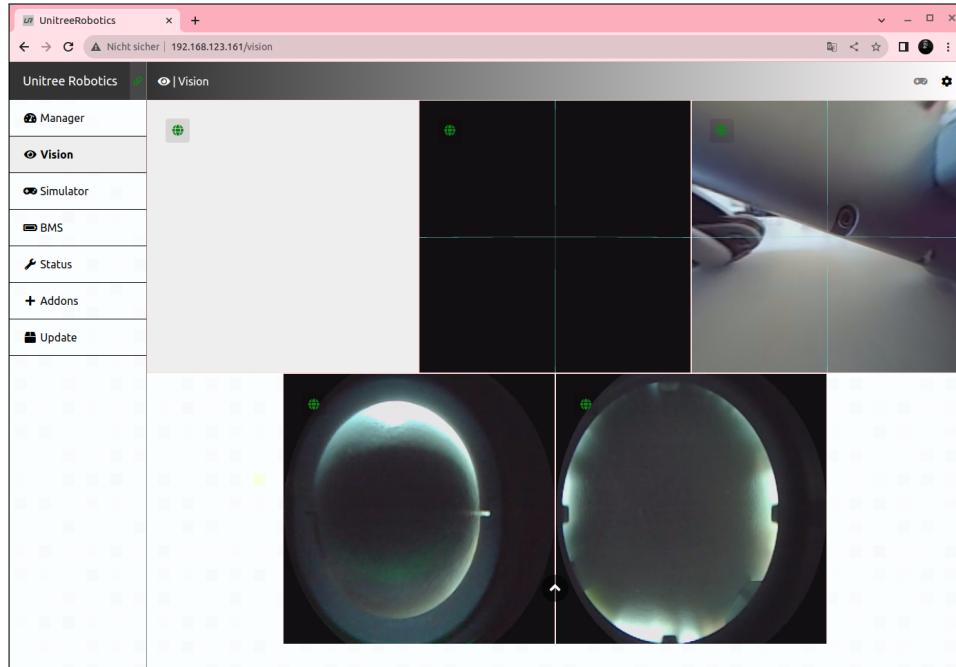


Abbildung A.4: Ausschnitte aus der Web-Applikation und Anzeige der verschiedenen Kameras

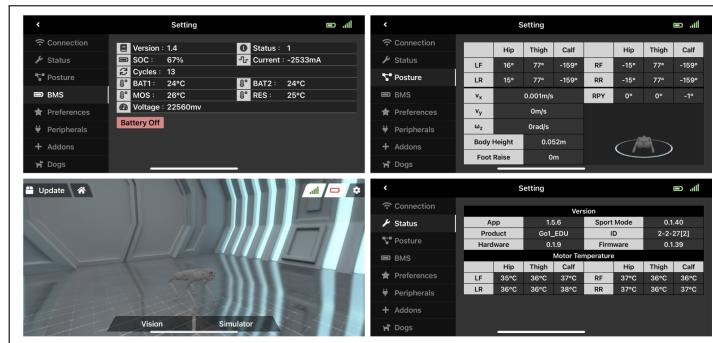
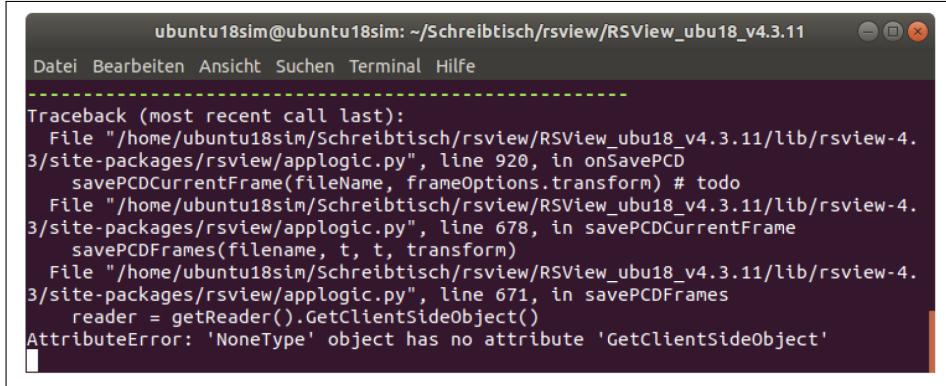
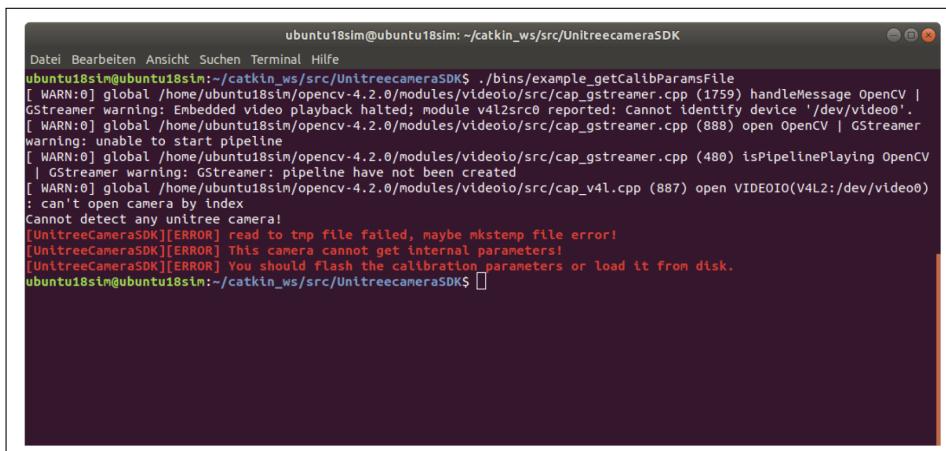


Abbildung A.5: Ausschnitte aus der iOS App



```
ubuntu18sim@ubuntu18sim: ~/Schreibtisch/rsview/RSView_ubu18_v4.3.11
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
-----
Traceback (most recent call last):
  File "/home/ubuntu18sim/Schreibtisch/rsview/RSView_ubu18_v4.3.11/lib/rsview-4.3/site-packages/rsview/applogic.py", line 920, in onSavePCD
    savePCDCurrentFrame(fileName, frameOptions.transform) # todo
  File "/home/ubuntu18sim/Schreibtisch/rsview/RSView_ubu18_v4.3.11/lib/rsview-4.3/site-packages/rsview/applogic.py", line 678, in savePCDCurrentFrame
    savePCDFrames(filename, t, t, transform)
  File "/home/ubuntu18sim/Schreibtisch/rsview/RSView_ubu18_v4.3.11/lib/rsview-4.3/site-packages/rsview/applogic.py", line 671, in savePCDFrames
    reader = getReader().GetClientSideObject()
AttributeError: 'NoneType' object has no attribute 'GetClientSideObject'
```

Abbildung A.6: Ausgabe der Fehlermeldung bei dem Versuch die Sensordaten zu speichern.



```
ubuntu18sim@ubuntu18sim: ~/catkin_ws/src/UnitreecameraSDK
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
ubuntu18sim@ubuntu18sim:~/catkin_ws/src/UnitreecameraSDK$ ./bins/example_getCalibParamsFile
[ WARN:0] global /home/ubuntu18sim/opencv-4.2.0/modules/videoio/src/cap_gstreamer.cpp (1759) handleMessage OpenCV | GStreamer warning: Embedded video playback halted; module v4l2src0 reported: Cannot identify device '/dev/video0'.
[ WARN:0] global /home/ubuntu18sim/opencv-4.2.0/modules/videoio/src/cap_gstreamer.cpp (888) open OpenCV | GStreamer warning: unable to start pipeline
[ WARN:0] global /home/ubuntu18sim/opencv-4.2.0/modules/videoio/src/cap_gstreamer.cpp (480) isPipelinePlaying OpenCV | GStreamer warning: GStreamer: pipeline have not been created
[ WARN:0] global /home/ubuntu18sim/opencv-4.2.0/modules/videoio/src/cap_v4l.cpp (887) open VIDEOIO(V4L2:/dev/video0) : can't open camera by index
Cannot detect any unitree camera!
[UnitreeCameraSDK][ERROR] read to tmp file failed, maybe mkstemp file error!
[UnitreeCameraSDK][ERROR] This camera cannot get internal parameters!
[UnitreeCameraSDK][ERROR] You should flash the calibration parameters or load it from disk.
ubuntu18sim@ubuntu18sim:~/catkin_ws/src/UnitreecameraSDK$
```

Abbildung A.7: Ausgabe der Fehlermeldung bei Verwendung des *CameraSDK*

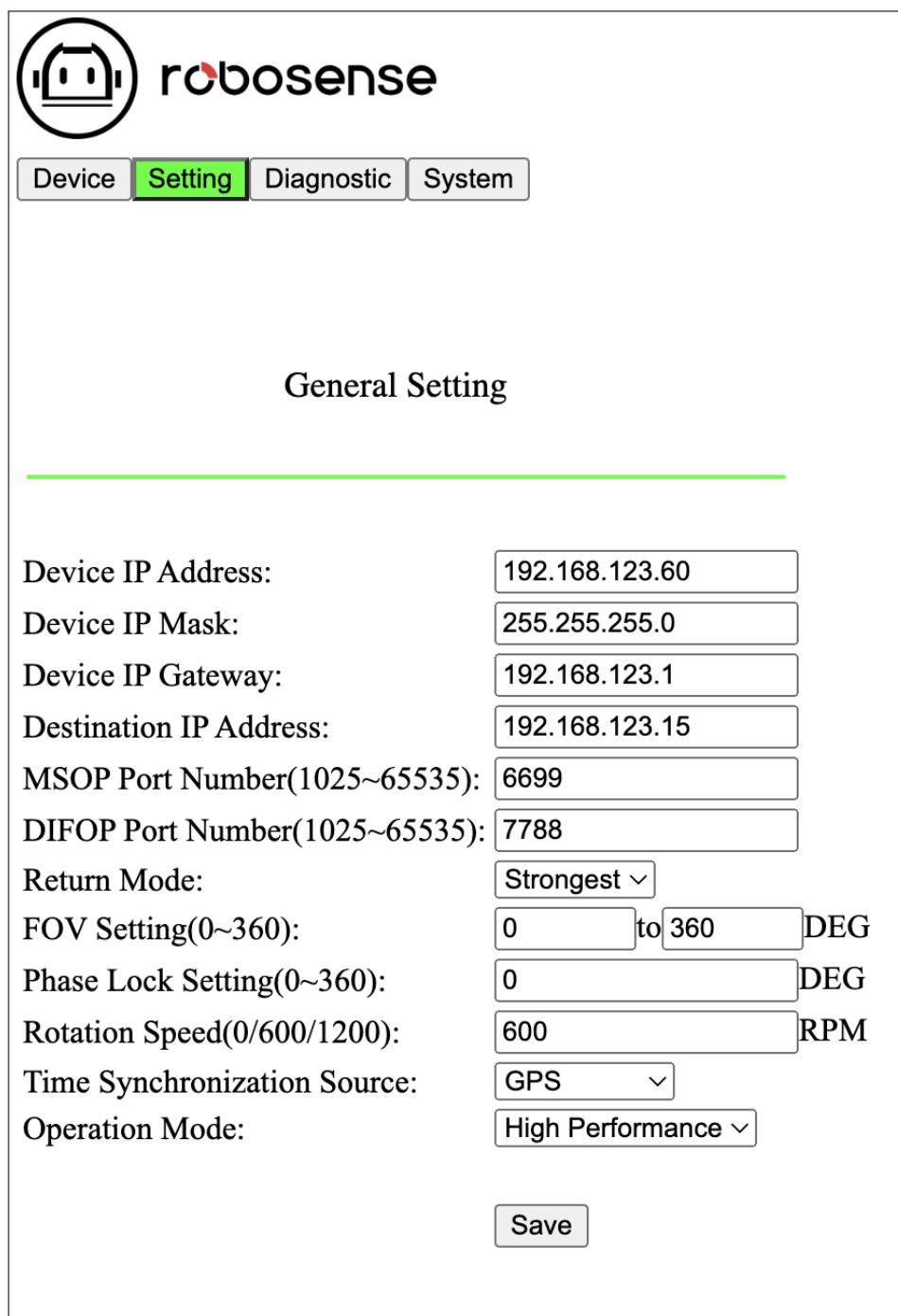


Abbildung A.8: Screenshot aus den Konfigurationseinstellungen des LiDAR

```
▶ unitree@nx:~/mapping_catkin_ws$ rosrun rs_to_velodyne rs_to_velodyne XYZIRT XYZIR
[ INFO] [1639976295.399523272]: Listening to /rslidar_points .....
Failed to find match for field 'intensity'.
```

Abbildung A.9: Ausgabe der Fehlermeldung bei der Konvertierung der LiDAR Formate

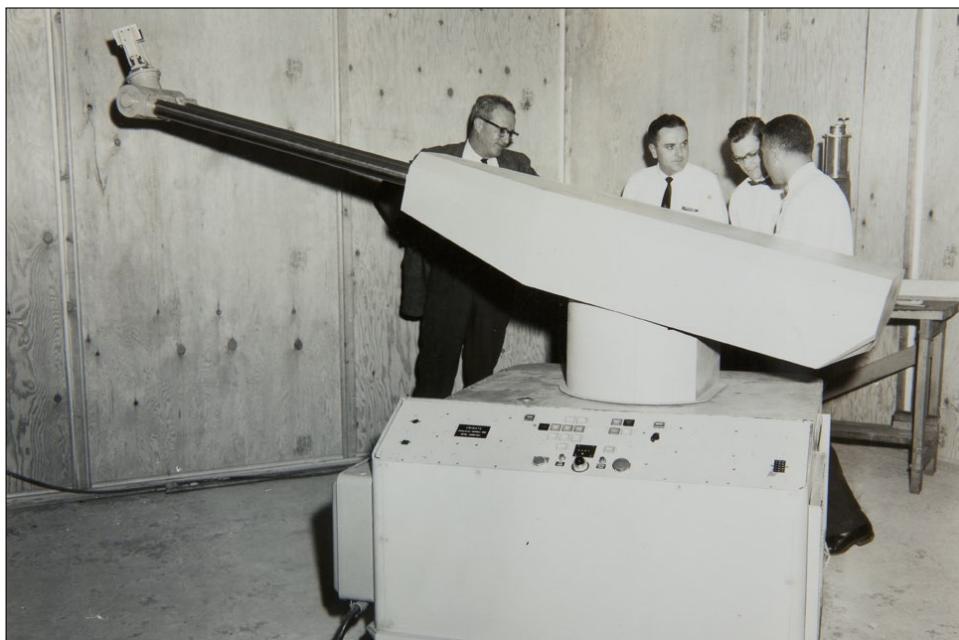


Abbildung A.10: Darstellung des *Unimate*-Roboterarm aus [Rob18]

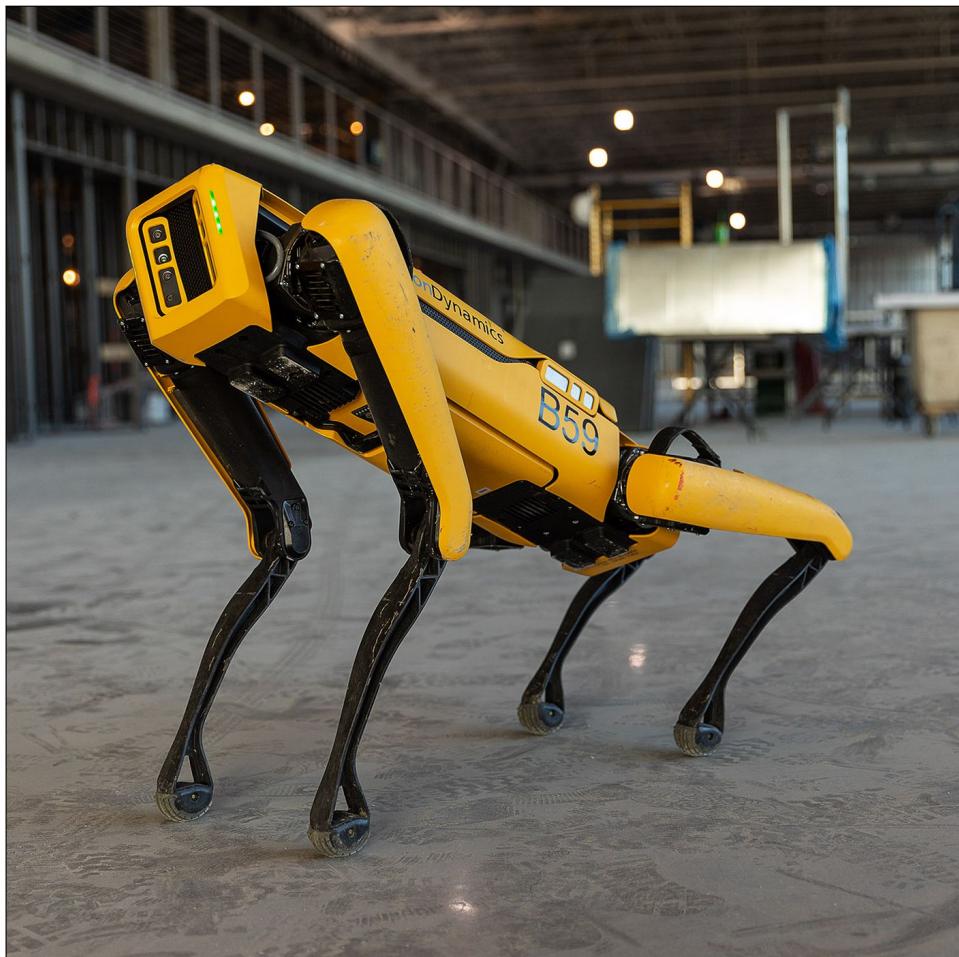


Abbildung A.11: Darstellung des *Spot*-Roboter aus [Ver20]

B Anhang: Code

```

1 #include "UnitreeCamera.hpp" // Assuming this is the header file for
2   // the UnitreeCamera SDK
3
4 int main(int argc, char *argv[]) {
5   // Initialize a UnitreeCamera object using a configuration file
6   // named "trans_rect_config.yaml"
7   UnitreeCamera cam("trans_rect_config.yaml");
8
9   // Check if the camera is opened successfully
10  if (!cam.isOpened())
11    exit(EXIT_FAILURE);
12
13  // Start capturing rectified stereo frames with image H.264
14  // encoding disabled and share memory sharing enabled
15  cam.startCapture(true, false);
16
17  // Main loop for capturing and potentially displaying rectified
18  // stereo frames
19  while (cam.isOpened()) {
20    cv::Mat left, right, feim;
21
22    // Attempt to get rectified stereo frames
23    if (!cam.getRectStereoFrame(left, right)) {
24      // If getting the stereo frames fails, wait for a short
25      // time and continue
26      usleep(1000);
27      continue;
28    }
29
30    // Uncomment the following line to display the frames

```

```

29     // cv::imshow("UnitreeCamera-RectifiedStereo", left);
30
31     // Wait for a short period (10 milliseconds) and check for user
32     // input
33     char key = cv::waitKey(10);
34
35     // If the ESC key (ASCII code 27) is pressed, exit the loop
36     if (key == 27)
37         break;
38
39     // Stop camera capturing
40     cam.stopCapture();
41
42     return 0;
43 }
```

Listing B.1: putImageTrans.cpp mit Kommentaren

```

1 #include <opencv2/opencv.hpp>
2 #include <iostream>
3
4 int main(int argc, char** argv)
5 {
6     std::string IpLastSegment = "13"; // Last segment of the camera's
7     // IP address - 13 for Head
8     int cam = 1; // Camera view index, default is 1
9
10    // Check if a camera view index is provided as a command-line
11    // argument
12    if (argc >= 2)
13        cam = std::atoi(argv[1]);
14
15    // GStreamer pipeline string components
16    std::string udpstrPrevData = "udpsrc address=192.168.123." +
17    IpLastSegment + " port=";
18    std::array<int, 5> udpPORT = std::array<int, 5>{9201, 9202, 9203,
19    9204, 9205};
20
21    std::string udpstrBehindData = " ! application/x-rtp,media=video,
22    encoding-name=H264 ! rtph264depay ! h264parse ! avdec_h264 !
23    videoconvert ! appsink";
24
25 }
```

```

18 // Construct the complete GStreamer pipeline string
19 std::string udpSendIntegratedPipe = udpstrPrevData + std::to_string
(udpPORT[cam - 1]) + udpstrBehindData;
20
21 std::cout << "udpSendIntegratedPipe: " << udpSendIntegratedPipe <<
std::endl;
22
23 // Open the video capture using the constructed GStreamer pipeline
24 cv::VideoCapture cap(udpSendIntegratedPipe);
25
26 // Check if the video capture was successfully opened
27 if (!cap.isOpened())
28     return 0;
29
30 cv::Mat frame;
31
32 // Continuous loop to capture and display video frames
33 while (1)
34 {
35     cap >> frame;
36     if (frame.empty())
37         break;
38
39     // Display the captured frame
40     imshow("video", frame);
41
42     // Wait for a short period (20 milliseconds)
43     cv::waitKey(20);
44 }
45
46 // Release the video capture resources
47 cap.release();
48
49 return 0;
50 }

```

Listing B.2: getimagetrans.cpp mit Kommentaren

```

1 #include "unitree_legged_sdk/unitree_legged_sdk.h"
2 #include <math.h>
3 #include <iostream>
4 #include <unistd.h>

```

```
5 #include <string.h>
6
7 using namespace UNITREE_LEGGED_SDK;
8
9 class Custom
10 {
11 public:
12     // Custom(uint8_t level): safe(LeggedType::A1), udp(level){
13     Custom(uint8_t level): safe(LeggedType::A1), udp(8090, "192.168.123.11", 8082, sizeof(HighCmd), sizeof(HighState)){
14     // Custom(uint8_t level): safe(LeggedType::A1), udp(8090, "192.168.123.161", 8082, sizeof(HighCmd), sizeof(HighState)){
15         udp.InitCmdData(cmd);
16         udp.SetDisconnectTime(dt, 1);
17         // udp.SetDisconnectTime(0, 0);
18     }
19     void UDPRecv();
20     void UDPSend();
21     void RobotControl();
22
23     Safety safe;
24     UDP udp;
25     HighCmd cmd = {0};
26     HighState state = {0};
27     int motiontime = 0;
28     float dt = 0.002;      // 0.001~0.01
29 };
30
31
32 void Custom::UDPRecv()
33 {
34     udp.Recv();
35 }
36
37 void Custom::UDPSend()
38 {
39     udp.Send();
40 }
41
42 void Custom::RobotControl()
43 {
44     motiontime += 2;
```

```

45     udp.GetRecv(state);
46     cmd.mode = 0;           // 0:idle, default stand      1:forced stand
47     2:walk continuously
48     cmd.gaitType = 0;
49     cmd.speedLevel = 0;
50     cmd.footRaiseHeight = 0;
51     cmd.bodyHeight = 0;
52     cmd.euler[0] = 0;
53     cmd.euler[1] = 0;
54     cmd.euler[2] = 0;
55     cmd.velocity[0] = 0.0f;
56     cmd.velocity[1] = 0.0f;
57     cmd.yawSpeed = 0.0f;

58 printf("rangeObstacle= [Head: %f, Left: %f, Right: %f, %f]\n",
59         state.rangeObstacle[0],
60         state.rangeObstacle[1],
61         state.rangeObstacle[2],
62         state.rangeObstacle[3]);
63
64     udp.SetSend(cmd);
65 }
66
67 int main(void)
68 {
69     std::cout << "Communication level is set to HIGH-level." << std::endl
70             << "WARNING: Make sure the robot is standing on the
71 ground." << std::endl
72             << "Press Enter to continue..." << std::endl;
73     std::cin.ignore();
74
75     Custom custom(HIGHLVEL);
76     InitEnvironment();
77     LoopFunc loop_control("control_loop", custom.dt, boost::bind(&
78     Custom::RobotControl, &custom));
79     LoopFunc loop_udpSend("udp_send", custom.dt, 3, boost::bind(&
80     Custom::UDPSend, &custom));
81     LoopFunc loop_udpRecv("udp_recv", custom.dt, 3, boost::bind(&
82     Custom::UDPRecv, &custom));
83
84     loop_udpSend.start();

```

```

81     loop_udpRecv.start();
82     loop_control.start();
83
84     while(1){
85         sleep(10);
86     };
87
88     return 0;
89 }
```

Listing B.3: Modifizierte example_walk Datei für die Ausgabe der Ultraschallsensoren

```

1 pi@raspberrypi:~ $ rostopic list
2 /camera1/point_cloud_face
3 /camera1/range_visual_face
4 /camera2/point_cloud_chin
5 /camera3/range_visual_left
6 /camera4/point_cloud_right
7 /camera4/range_visual_right
8 /camera5/point_cloud_rearDown
9 /cmd_odom
10 /cmd_vel
11 /cmd_vel_2
12 /joint_states
13 /lcm_node/obs_env
14 /lcm_node/ultrasonic_env
15 /move_base_simple/goal
16 /pointcloud_process/ground_pointcloud
17 /range_front
18 /range_left
19 /range_right
20 /range_ultrasonic_face
21 /range_ultrasonic_left
22 /range_ultrasonic_right
23 /ros2udp/odom
24 /ros2udp_motion_mode_adv/joystick
25 /rosout
26 /rosout_agg
27 /tf
28 /tf_static
29 /ukd_triple/pose
30 /ukd_triple/state
```

```
31 /ukd_triple_2_goal/path_tag_line  
32 /ukd_triple_2_goal/path_tag_window
```

Listing B.4: Ausgabe der auf dem Raspberry Pi laufenden ROS-Topics

C Anhang: Tabellen

Onboard Raspberry Pi Kenndaten

Modell	Raspberry Pi Compute Module 4 Rev 1.0
Chipset	Broadcom BCM2711 quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
RAM	1.8 GB
Speicher	32 GB SD-Karten Festplattenspeicher
Betriebssystem	Debian 10 (Buster)
Kernel	Linux Kernel 5.4.81 - rt45-v8+

Tabelle C.1: Kenndaten des *Raspberry Pi* an Bord des *Go1*

NVIDIA Jetson Xavier NX Kenndaten

Modell	NVIDIA Jetson Xavier NX
Chipset	6-Core NVIDIA Carmel ARM v8.2 64-bit CPU
RAM	8 GB
Speicher	16 GB SD-Karten Festplattenspeicher / 120 GB SSD
Betriebssystem	Ubuntu 18.04.5 LTS
Kernel	4.9.201 -tegra

Tabelle C.2: Kenndaten des *NVIDIA Xavier NX* an Bord des *Go1*

NVIDIA Jetson Nano Kenndaten

Modell	NVIDIA Jetson Nano Developer Kit
Chipset	Quad-Core ARM Cortex-A57 MPCore
GPU	NVIDIA Maxwell-GPU128 Cores
RAM	4 GB
Speicher	16 GB SD-Karten Festplattenspeicher
Betriebssystem	Ubuntu 18.04.5 LTS
Kernel	4.9.201 -tegra

Tabelle C.3: Kenndaten der *NVIDIA Jetson NANO* an Bord des *Go1*

Literaturverzeichnis

- [AAP08] Haider Ali, Basheer Ahmed und Gerhard Paar. *Robust window detection from 3d laser scanner data*. Bd. 2. 2008, S. 115–118.
- [Ble+18] Gerardo Bledt u. a. *MIT Cheetah 3: Design and Control of a Robust, Dynamic Quadruped Robot*. Madrid, Spain: IEEE Press, 2018, 2245–2252. DOI: 10.1109/IROS.2018.8593885. URL: <https://doi.org/10.1109/IROS.2018.8593885>.
- [BM92] PJ Besl und ND McKay. *A method for registration of 3-D shapes*, *IEEE T. Pattern Anal.*, 14, 239–256. 1992.
- [BS03] Peter Biber und Wolfgang Straßer. *The normal distributions transform: A new approach to laser scan matching*. Bd. 3. 2003, S. 2743–2748.
- [Bur20] Paweł Burdziakowski. *Increasing the geometrical and interpretation quality of unmanned aerial vehicle photogrammetry products using super-resolution algorithms*. Bd. 12. 5. MDPI, 2020, S. 810.
- [Cad+16] Cesar Cadena u. a. *Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age*. Bd. 32. 6. IEEE, 2016, S. 1309–1332.
- [Car+21] Alexander Carballo u. a. *Characterization of Multiple 3D LiDARs for Localization and Mapping Performance using the NDT Algorithm*. 2021, S. 327–334. DOI: 10.1109/IVWorkshops54471.2021.9669244.
- [CD22] Jerred Chen und Frank Dellaert. *A1 SLAM: Quadruped SLAM using the A1’s Onboard Sensors*. 2022. arXiv: 2211.14432 [cs.RO].
- [DWB06] H Durrant-Whyte und T Bailey. *Simultaneous Localisation and Mapping (SLAM): Part i the essential algorithms*. *Robotics & Automation Magazine, IEEE*, 13 (2), 99–110. 2006.

- [Eng+17] Francis Engelmann u. a. *Exploring Spatial Context for 3D Semantic Segmentation of Point Clouds*. IEEE, 2017. DOI: 10.1109/iccvw.2017.90. URL: <https://doi.org/10.1109%2Ficcvw.2017.90>.
- [FB81] M. Fischler und R. Bolles. *Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography*. Bd. 24. 6. 1981, S. 381–395. URL: /brokenurl#http://publication.wilsonwong.me/load.php?id=233282275.
- [Gai+16] Adrien Gaidon u. a. *Virtual Worlds as Proxy for Multi-Object Tracking Analysis*. 2016. arXiv: 1605.06457 [cs.CV].
- [GBC16] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [Gér19] A. Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019. ISBN: 9781492032595. URL: <https://books.google.de/books?id=HnetDwAAQBAJ>.
- [Hac+17] Timo Hackel u. a. *Semantic3D.net: A new Large-scale Point Cloud Classification Benchmark*. 2017. arXiv: 1704.03847 [cs.CV].
- [HBMO22] Stefan Hensel, Marin B. Marinov und Markus Obert. *3D LiDAR Based SLAM System Evaluation with Low-Cost Real-Time Kinematics GPS Solution*. Bd. 10. 9. 2022. DOI: 10.3390/computation10090154. URL: <https://www.mdpi.com/2079-3197/10/9/154>.
- [Hen+20] Stefan Hensel u. a. *Experimental Set-up for Evaluation of Algorithms for Simultaneous Localization and Mapping*. 2020, S. 433–444.
- [Hes+16] Wolfgang Hess u. a. *Real-time loop closure in 2D LIDAR SLAM*. 2016, S. 1271–1278.
- [HSL13] C. Hayot, S. Sakka und P. Lacouture. *Contribution of the six major gait determinants on the vertical center of mass trajectory and the vertical ground reaction force*. Bd. 32. 2. 2013, S. 279–289. DOI: <https://doi.org/10.1016/j.humov.2012.10.003>. URL: <https://www.sciencedirect.com/science/article/pii/S0167945712001212>.
- [IPP21] Malinka Ivanova, Petya Petkova und Nikolay Petkov. *Machine Learning and Fuzzy Logic in Electronics: Applying Intelligence in Practice*. Bd. 10. 22. MDPI, 2021, S. 2878.

- [JC21] L. Joseph und J. Cacace. *Mastering Ros for Robotics Programming: Best Practices and Troubleshooting Solutions When Working with Ros*. Packt Publishing, 2021. ISBN: 9781801071024. URL: <https://books.google.de/books?id=08GQzgEACAAJ>.
- [Kat18] Benjamin Katz. *A low cost modular actuator for dynamic robots*. Jan. 2018.
- [Kit+16] Satoshi Kitano u. a. *TITAN-XIII: sprawling-type quadruped robot with ability of fast and energy-efficient walking*. Bd. 3. März 2016. DOI: 10.1186/s40648-016-0047-1.
- [KL80] V. Klema und A. Laub. *The singular value decomposition: Its computation and some applications*. Bd. 25. 2. 1980, S. 164–176. DOI: 10.1109/TAC.1980.1102314.
- [Koi+21] Kenji Koide u. a. *Voxelized GICP for fast and accurate 3D point cloud registration*. 2021, S. 11054–11059.
- [Leh23] Noah Lehmann. *Integration eines Unitree Go1 Quadruped Roboters in ein Hochschulökosystem*. Okt. 2023.
- [LIG20] You Li und Javier Ibanez-Guzman. *Lidar for Autonomous Driving: The Principles, Challenges, and Trends for Automotive Lidar and Perception Systems*. Bd. 37. 4. 2020, S. 50–61. DOI: 10.1109/MSP.2020.2973615.
- [Ma+20] Qi Ma u. a. *A Cloud-based Quadruped Service Robot with Multi-Scene Adaptability and various forms of Human-Robot Interaction*. Bd. 53. 5. 3rd IFAC Workshop on Cyber-Physical & Human Systems CPHS 2020. 2020, S. 134–139. DOI: <https://doi.org/10.1016/j.ifacol.2021.04.092>. URL: <https://www.sciencedirect.com/science/article/pii/S2405896321001786>.
- [Mai16] Helmut Maier. *Grundlagen der Robotik*. Berlin: VDE-Verlag, 2016. ISBN: 978-3-8007-3944-8.
- [MB+22] Andréa Macario Barros u. a. *A Comprehensive Survey of Visual SLAM Algorithms*. Bd. 11. 1. 2022. DOI: 10.3390/robotics11010024. URL: <https://www.mdpi.com/2218-6581/11/1/24>.
- [McC+06] John McCarthy u. a. *A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence, August 31, 1955*. Bd. 27. Jan. 2006, S. 12–14.

- [MF13] Aaron Martinez und Enrique Fernndez. *Learning ROS for Robotics Programming*. Packt Publishing, 2013. ISBN: 1782161449.
- [Mol+13] E. Molinos u. a. *Comparison of Local Obstacle Avoidance Algorithms*. Hrsg. von Roberto Moreno-Díaz, Franz Pichler und Alexis Quesada-Arencibia. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, S. 39–46. ISBN: 978-3-642-53862-9.
- [MSD18] Ievgeniia Maksymova, Christian Steger und Norbert Druml. *Review of Li-DAR sensor data acquisition and compression for automotive applications*. Bd. 2. 13. 2018, S. 852.
- [OEC19] OECD. *Artificial Intelligence in Society*. 2019, S. 152. DOI: <https://doi.org/https://doi.org/10.1787/eedfee77-en>. URL: <https://www.oecd-ilibrary.org/content/publication/eedfee77-en>.
- [RN21] Stuart Russell und Peter Norvig. *Artificial Intelligence, Global Edition A Modern Approach*. Pearson Deutschland, 2021, S. 1168. ISBN: 9781292401133. URL: <https://elibrary.pearson.de/book/99.150005/9781292401171>.
- [Sap+21] Azhar Aulia Saputra u. a. *AQuRo: A Cat-like Adaptive Quadruped Robot With Novel Bio-Inspired Capabilities*. Bd. 8. 2021. DOI: 10.3389/frobt.2021.562524. URL: <https://www.frontiersin.org/articles/10.3389/frobt.2021.562524>.
- [SB18] R.S. Sutton und A.G. Barto. *Reinforcement Learning, second edition: An Introduction*. Adaptive Computation and Machine Learning series. MIT Press, 2018, S. 1–26. ISBN: 9780262039246. URL: <https://books.google.de/books?id=sWV0DwAAQBAJ>.
- [Spa19] Günter Spanner. *Robotik und Künstliche Intelligenz*. eBook, PDF. Elektor International Media, 2019. ISBN: 978-3-89576-345-8.
- [Spe+21] Mariusz Specht u. a. *Concept of an innovative autonomous unmanned system for bathymetric monitoring of shallow waterbodies (INNOBAT system)*. Bd. 14. 17. MDPI, 2021, S. 5370.
- [Tar12] Sasu Tarkoma. *Publish/Subscribe Systems: Design and Principles*. English. United States: Wiley, Aug. 2012. ISBN: 9781119951544. DOI: 10.1002/9781118354261.

- [TKDT14] Than Trong Khanh Dat und Phuc Tran. *A Study on Locomotions of Quadruped Robot*. Bd. 282. Jan. 2014, S. 595–604. ISBN: 978-3-642-41967-6. DOI: 10.1007/978-3-642-41968-3_59.
- [Tur09] Alan M. Turing. *Computing Machinery and Intelligence*. Hrsg. von Robert Epstein, Gary Roberts und Grace Beber. Dordrecht: Springer Netherlands, 2009, S. 23–65. ISBN: 978-1-4020-6710-5. DOI: 10.1007/978-1-4020-6710-53. URL: https://doi.org/10.1007/978-1-4020-6710-5_3.
- [Wal09] Richard S. Wallace. *The Anatomy of A.L.I.C.E.* 2009, S. 181. DOI: 10.1007/978-1-4020-6710-513.
- [Wan+18] Guowei Wan u. a. *Robust and precise vehicle localization based on multi-sensor fusion in diverse city scenes*. 2018, S. 4670–4677.
- [Xia22] P. Xiao. *Artificial Intelligence Programming with Python: From Zero to Hero*. Wiley, 2022. ISBN: 9781119820864. URL: <https://books.google.de/books?id=1KR-zgEACAAJ>.
- [Zha+14] Jiaqi Zhang u. a. *Trot Gait Design and CPG Method for a Quadruped Robot*. Bd. 11. 1. 2014, S. 18–25. DOI: [https://doi.org/10.1016/S1672-6529\(14\)60016-0](https://doi.org/10.1016/S1672-6529(14)60016-0). URL: <https://www.sciencedirect.com/science/article/pii/S1672652914600160>.
- [Zha+21] Chi Zhang u. a. *FRL-SLAM: A Fast, Robust and Lightweight SLAM System for Quadruped Robot Navigation*. 2021, S. 1165–1170. DOI: 10.1109/ROBI54168.2021.9739499.
- [ZS14] Ji Zhang und Sanjiv Singh. *LOAM: Lidar odometry and mapping in real-time*. Bd. 2. 9. 2014, S. 1–9.

Internetquellen

- [cha] champ. *ROS Packages for CHAMP Quadruped Controller*. URL: <https://github.com/chvmp/champ> (besucht am 21.08.2023).
- [CSI23] Fraunhofer Institute for Cognitive Systems IKS. *Artificial Intelligence*. 2023. URL: <https://www.iks.fraunhofer.de/de/themen/kuenstliche-intelligenz.html> (besucht am 06.06.2023).
- [DEV23] NVIDIA DEVELOPER. *Isaac Gym - Preview Release*. 2023. URL: <https://developer.nvidia.com/isaac-gym> (besucht am 30.07.2023).
- [Dyn23] Boston Dynamics. *Spot*. 2023. URL: <https://www.bostondynamics.com/products/spot> (besucht am 03.07.2023).
- [Ele] Reichelt Elektronik. *QR GO1 BATT*. URL: <https://www.reichelt.de/de/de/quadruped-go1-akku-4-500-mah-qr-go1-batt-p334403.html?r=1> (besucht am 10.08.2023).
- [ENG23] A TRIBUTE TO JOSEPH ENGELBERGER. *UNIMATE The First Industrial Robot*. 2023. URL: <https://www.automate.org/a3-content/joseph-engelberger-unimate> (besucht am 21.08.2023).
- [ENN23] Ennomotive. *Industrial IoT Sensor Prices*. 2023. URL: <https://www.ennomotive.com/industrial-iot-sensor-prices/#:~:text=The%20cost%20of%20sensors%20is,decisions%20at%20a%20lower%20cost>. (besucht am 06.06.2023).
- [Fau22] Alberto Paitoni Faustinoni. *History of Robots: Origins, Myths, Facts*. 2022. URL: <https://robotics24.net/blog/history-of-robots-origins-myths-facts/> (besucht am 24.06.2023).
- [Fra19] Historisches Museum Frankfurt. *Shakey der Roboter*. 2019. URL: <https://blog.hnf.de/shakey-der-roboter/> (besucht am 26.06.2023).

- [IBM23a] IBM. *IBM100 - Deep Blue*. 2023. URL: <https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/> (besucht am 06.06.2023).
- [IBM23b] IBM. *Unsupervised Learning*. 2023. URL: <https://www.ibm.com/topics/unsupervised-learning# :~:text=Unsupervised%20learning%2C%20also%20known%20as,need%20for%20human%20intervention.> (besucht am 06.06.2023).
- [Ind21] The Independent. *SpaceX Starship rocket blows up after landing in latest test flight*. 2021. URL: <https://www.independent.co.uk/space/spot-robot-dog-spacex-sn10-b1813085.html> (besucht am 03.07.2023).
- [koi23] koide3. *hdl graph slam*. 2023. URL: https://github.com/koide3/hdl_graph_slam (besucht am 21.08.2023).
- [Mat19] Kayla Matthews. *5 materials to evaluate for designing, building robust robots*. 2019. URL: <https://www.therobotreport.com/materials-rugged-robot-design-building/> (besucht am 30.07.2023).
- [MAV] MAVProxUser. *HangZhou Yushu Tech Unitree Go1*. URL: <https://github.com/MAVProxyUser/YushuTechUnitreeGo1> (besucht am 10.08.2023).
- [Nvi] Nvidia. *Quadruped*. URL: https://docs.omniverse.nvidia.com/isaacsim/latest/ext_omni_isaac_quadruped.html (besucht am 21.08.2023).
- [Par20] Devin Partida. *What Are the Main Components of Robots?* 2020. URL: <https://rehack.com/trending/science/what-are-the-main-components-of-robots/> (besucht am 07.07.2023).
- [Ph.20] Michele Baker Ph.D. *Hydraulic vs. Pneumatic vs. Electric Actuators*. 2020. URL: <https://yorkpmh.com/resources/hydraulic-vs-pneumatic-vs-electric-actuators/#:~:text=Hydraulic%20power%20performance%20is%20also,than%20hydraulic%20and%20electric%20actuators.> (besucht am 07.07.2023).
- [RL] RoboSense-Lidar. *RS Lidar SDK*. URL: https://github.com/RoboSense-LiDAR/rslidar_sdk (besucht am 10.08.2023).
- [Roba] Robosense. *Helios Series*. URL: <https://www.robosense.ai/en/rslidar/RS-Helios> (besucht am 10.08.2023).
- [Robb] Unitree Robotics. *GO-M8010-6 Motor*. URL: <https://shop.unitree.com/products/go1-motor> (besucht am 10.08.2023).

- [Robc] Unitree Robotics. *Unitree Go1 Battery*. URL: <https://shop.unitree.com/products/unitree-go1-battery> (besucht am 10.08.2023).
- [Robd] Unitree Robotics. *UnitreecameraSDK*. URL: <https://github.com/unitreerobotics/UnitreecameraSDK> (besucht am 18.08.2023).
- [Robe] Generation Robots. *Go1 Datasheet EN*. URL: https://www.generationrobots.com/media/unitree/Go1%20Datasheet_EN%20v3.0.pdf (besucht am 10.08.2023).
- [Rob16a] Open Robotics. *Concepts*. 2016. URL: <http://wiki.ros.org/ROS/Concepts> (besucht am 18.07.2023).
- [Rob16b] Open Robotics. *Messages*. 2016. URL: <http://wiki.ros.org/Messages> (besucht am 18.07.2023).
- [Rob16c] Open Robotics. *Services*. 2016. URL: <http://wiki.ros.org/Services> (besucht am 18.07.2023).
- [Rob16d] Open Robotics. *Topics*. 2016. URL: <http://wiki.ros.org/Topics> (besucht am 18.07.2023).
- [Rob18] Kawasaki Robotics. *Kawasaki Robotics - History*. 2018. URL: https://robotics.kawasaki.com/en1/anniversary/history/history_01.html (besucht am 25.06.2023).
- [ROB19] UNIVERSAL ROBOTS. *TYPES OF SENSORS IN ROBOTICS — UNIVERSAL ROBOTS*. 2019. URL: <https://www.universal-robots.com/in/blog/types-of-sensors-in-robotics-universal-robots/> (besucht am 10.07.2023).
- [Rob21] Open Robotics. *Why ROS?* 2021. URL: <https://www.ros.org/blog/why-ros/> (besucht am 18.07.2023).
- [Rob23a] Open Robotics. *Concepts*. 2023. URL: <https://docs.ros.org/en/iron/Concepts.html> (besucht am 18.07.2023).
- [Rob23b] Open Robotics. *ROS Index*. 2023. URL: <https://index.ros.org/packages/> (besucht am 18.07.2023).
- [ROS] micro ROS. *micro-ROS for STM32CubeMX/IDE*. URL: https://github.com/micro-ROS/micro_ros_stm32cubemx_utils (besucht am 10.08.2023).

- [Sha21] Shasthrasnehi. *Quadruped Robotics: The Evolution of Four-Legged Robots*. 2021. URL: <https://shasthrasnehi.com/quadruped-robotics-the-evolution-of-four-legged-robots/> (besucht am 03.07.2023).
- [Sta10] Stanford University. *Stanford LittleDog Project*. 2010. URL: <https://cs.stanford.edu/groups/littledog/> (besucht am 03.07.2023).
- [Ver20] The Verge. *Boston Dynamics' robot dog is patrolling a SpaceX site in latest video*. 2020. URL: <https://www.theverge.com/2020/2/19/21144648/boston-dynamics-spot-robot-mass-state-police-trial-issues> (besucht am 02.07.2023).
- [Web17] ROS-Industrial Website. *Using rqt Tools for Analysis*. 2017. URL: https://industrial-training-master.readthedocs.io/en/melodic/_source/session6/Using-rqt-tools-for-analysis.html (besucht am 21.08.2023).

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde nach meiner besten Kenntnis bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Hof, den 13. September 2023

JONAS KEMNITZER