# Integration eines Unitree Go1 Quadruped Roboters in ein Hochschul-Ökosystem

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

**Hochschule für Angewandte Wissenschaften Hof**
**Fakultät Informatik**
**Studiengang Master Informatik**

**Vorgelegt bei**
**Prof. Dr. Christian Groth**
**Alfons-Goppel-Platz 1**
**95028 Hof**

**Vorgelegt von**
**Noah Lehmann**

**Hof, 31. August 2023**

# Inhaltsverzeichnis

**1 Einleitung** **1**
1.1 Unitree Robotics GO1 Edu . . . 2
1.2 Vorgehensweise . . . 3
1.3 Zielsetzung . . . 4

**2 Grundlagen** **6**
2.1 Robotik . . . 6
2.1.1 Industrieroboter . . . 7
2.1.2 Serviceroboter . . . 7
2.1.3 Cobots . . . 8
2.1.4 Quadrupedie . . . 9
2.2 Stand der Forschung . . . 9
2.2.1 Quadruped Roboter . . . 9
2.2.2 Integration von Robotern . . . 11
2.2.3 Unitree Go1 Ressourcen . . . 14
2.3 Herausforderungen . . . 17

**3 Roboterarchitektur und Systemkomponenten** **20**
3.1 Aufbau . . . 20
3.1.1 Überblick . . . 20
3.1.2 Mechanische Komponenten . . . 21
3.1.3 Sensorik . . . 22
3.1.4 Recheneinheiten und Schnittstellen . . . 25
3.2 Hardware Architektur . . . 27
3.2.1 Überblick . . . 28
3.2.2 Kernelemente . . . 29
3.3 Netzwerk . . . 35
3.3.1 Überblick . . . 36
3.3.2 Gekabelte Verbindung . . . 37
3.3.3 Kabellose Verbindung . . . 37
3.3.4 Weitere Netzwerk Funktionen . . . 38

**4 Analyse des Roboters** **39**
4.1 Inbetriebnahme . . . 39
4.1.1 Lieferumfang . . . 39
4.1.2 Mobile Inbetriebnahme . . . 41
4.1.3 Stationäre Inbetriebnahme . . . 42
4.1.4 Grafische Anwendungen . . . 43
4.2 Funktionen . . . 45
4.2.1 Software Autostart . . . 45
4.2.2 Fernsteuerung . . . 47
4.2.3 Lokales Netzwerk . . . 51
4.2.4 Audio Interfaces . . . 52
4.2.5 Kopfbeleuchtung . . . 55
4.2.6 Video Streaming . . . 57

4.2.7 Batterie Management . . . . . 58
4.3 Weitere Funktionen . . . . . 61

**5 Funktionserweiterungen 63**
5.1 Konnektivität . . . . . 63
5.2 BMS . . . . . 71
5.3 Remote Video Streaming . . . . . 76
5.4 Fernsteuerung . . . . . 78

**6 Fazit 82**
6.1 Rückblick . . . . . 82
6.2 Einschätzung . . . . . 83
6.3 Ausblick . . . . . 84

**A Listings 87**

# Abkürzungsverzeichnis

APN . . . . . Access Point Name

BMS . . . . . Batterie Management System

CPU . . . . . Central Processing Unit

DARPA . . . . . Defense Advanced Research Projects Agency

DHCP . . . . . Dynamic Host Configuration Protocol

GB . . . . . Gigabyte

GO1 . . . . . Unitree Robotics Go1 Edu

GPS . . . . . Global Positioning System

GPU . . . . . Graphics Processing Unit

HDMI . . . . . High Definition Multimedia Interface

hostapd . . . . . Host Access Point Daemon

HTML5 . . . . . Hypertext Markup Language V.5

HTTP . . . . . Hypertext Transfer Protocol

IP . . . . . Internet Protocoll

KI . . . . . Künstliche Intelligenz

LED . . . . . Leuchtdioden

Lidar . . . . . Light Detection and Ranging

LTE . . . . . Long Term Evolution

LXDE . . . . . Lightweight X11 Desktop Environment

MCU . . . . . Main Control Unit

MIT . . . . . Massachusetts Institute of Technology

ML . . . . . Machine Learning

NM . . . . . Newton/Meter

PDF . . . . . Portable Document Format

PIN . . . . . Personal Identification Number

PIXEL . . . . . Pi Improved Xwindows Environment, Lightweight

RCTA . . . . . Robotics Collaborative Technology Alliance

ROI . . . . . Return on Investment

ROS . . . . . Robot Operation System

RTSP . . . . . Real-Time Streaming Protocoll

SD . . . . . Secure Digital

SIM . . . . . Subscriber Identity Module

SLAM . . . . . Simultaneous Localization and Mapping

SoC . . . . . State of Charge

SSD . . . . . Solid State Drive

SSH . . . . . Secure Shell

SSID . . . . . Service Set Identifier

URL . . . . . Uniform Resource Locator

US . . . . . United States

USA . . . . . Vereinigte Staaten von Amerika

USB . . . . . Universal Serial Bus

USD . . . . . US-Dollar

VDI . . . . . Verband Deutscher Ingenieure

VPN . . . . . Virtual Private Network

WebRTC . . . . . Web Real Time Communications

WLAN . . . . . Wireless Local Area Network

WWAN . . . . . Wireless Wide Area Network

# Abbildungsverzeichnis


1 Verkaufsbild des Unitree GO1 . . . . . 3
2 Vergleich zwischen Industrie- und Servicerobotern . . . . . 7
3 Big Dog (links), MIT Cheetah 3 (mitte) und Spot (rechts) . . . . . 10
4 Überblick über den GO1 . . . . . 20
5 Mechanische Komponenten des Go1 . . . . . 22
6 Sensorik des Laufaparats . . . . . 23
7 Darstellung der verbauten Kamera und Sensorik . . . . . 24
8 Blick auf die internen Komponenten . . . . . 25
9 Ansicht des Kopfes von hinten . . . . . 26
10 Vogelperspektive mit Hardware . . . . . 27
11 Überblick über die interne Architektur des Go1 . . . . . 28

12 Überblick über Netzwerkkonfiguration . . . . . 36
13 Ansicht der Transportbox und des Lieferumfangs . . . . . 40
14 Ausgangsposition des Roboters . . . . . 41
15 XT-30 auf Hohlstecker Verkabelung und XT-30 Orientierung . . . . . 43
16 Screenshot des Webinterfaces . . . . . 44
17 Screenshot der mobilen Anwendung . . . . . 45
18 Hauptfernbedienung (links) und Label-Controller (rechts) . . . . . 48
19 App-Menüpunkte `Peripherals > Bluetooth Gamepad` und `Gamepad List` 49
20 Bildschirmaufnahme des App-Controllers . . . . . 49
21 Bildschirmaufnahme des Web-Controllers . . . . . 50
22 App-Menüpunkte `Peripherals > Track Tag` und die Tracking-Übersicht 51
23 Überblick über Netzwerkkonfiguration . . . . . 53
24 Ausgabe eines MQTT Explorers in Verbindung mit dem Raspberry Pi als Broker . . . . . 56
25 MQTT Explorer mit Topic `face_light/color` (links) und App-Funktion (rechts) . . . . . 56
26 Kamerabild des GO1 . . . . . 59
27 Batterieinformationen in der Webseite (links) und App (rechts) . . . . . 59
28 Sequenzdiagramm der Batterie-Monitoring Komponenten . . . . . 72
29 Das Kamerabild des Kopfes in der Browseransicht . . . . . 78
30 Diagramm der Komponenten einer beispielhaften Fernsteuerungssoftware . 80

# Listings

1 Inhalt der Autostartdatei `/home/pi/UnitreeUpgrade/start.sh` . . . . . 46
2 Konfiguration des hostapd in `configNetwork.sh` . . . . . 51
3 Konfiguration des `hostapd` . . . . . 51
4 Inhalt der Webpack Datei `bmsReceivers.ts` . . . . . 60
5 Verbindes des `wlan2` Interface . . . . . 65
6 Aufrufen des `connectWlan2.sh` Skripts . . . . . 65
7 Modemkonfiguration des `wwan0` Interfaces . . . . . 70
8 Datei `install.sh` zur BMS-Moniitor Initialisierung . . . . . 73
9 Entkopplung des BMS-Monitor durch `run.sh` . . . . . 73
10 Dekodierung der binären BMS Daten . . . . . 73
11 Konstantes Leuchten der LEDs durch `const_led()` . . . . . 74
12 Rotes Blinken der LEDs durch `alert_led()` . . . . . 74
13 Hinlegen des Hundes im Skript `sniff_bms.py` . . . . . 75
14 Konstanten der Python Umgebung des BMS Monitors . . . . . 75
15 Ausschnitt der Datei `Vision.vue` . . . . . 78

# Tabellenverzeichnis

1 Kenndaten des Raspberry Pi . . . . . 30
2 Kenndaten des NVIDIA Jetson Nano im Kopf des Roboters . . . . . 33
3 Kenndaten des NVIDIA Jetson Xavier NX . . . . . 35

4 Zusammenfassung der Tastenfunktionen des Label Controllers . . . . . 51
5 Übersicht der BMS Message Payload . . . . . 61

# 1 Einleitung

Der stetige Wachstum weltweiter Wirtschaftsräume und die ständigen Erneuerungen bekannter Mechanismen un Entwicklung, Produktion und Vertrieb bedürfen einer stetigen Effizienzsteigerung in allen Bereichen, die dei Kennzahlen zur Messung des Erfolges der Wirtschaft beeinflussen. Eine gängige Möglichkeit zur Steigerung der Effizienz ist die Automation diverser Prozesse, welche oftmals mit der Eliminierung menschlichen Fehlerpotentials in der Umsetzung einhergeht. Ansätze hierfür sind oftmals der Einsatz künstlicher Intelligenzen und hoch spezialisierter Maschinen, in vielen Fällen sogar die Kombination aus beidem. Die Folge ist eine schnelle und präzise Reaktion auf gegebene Einflüsse und Daten.

Zu beobachten ist dieses Phänomen bereits seit Jahrzehnten im Bereich der Industrie, besonders in der Produktion, Materialhandhabung und Lagerhaltung. Maschinen werden hier so konfiguriert, dass sie einzelne Schritte einer Produktionskette effizient und weitgehend fehlerfrei übernehmen können, was besonders im Vergleich zum Einsatz menschlicher Arbeitskraft die Kosten senkt und die Qualitätssicherung vereinfacht. Solche Maschinen werden ab einem gewissen Automationsgrad Roboter genannt, was im Bereich der menschlichen Unterstützung und teilweise Ersetzung besonders passend ist, da das Wort *Roboter* vom tschechischen *robota* kommt, was so viel wie Frondienst bedeutet, den Menschen also unterstützend.

Mit der Verbreitung der Roboter im industriellen Bereich und dem Einsatz dessen in diversen Szenarien weltweit sind auch die akademische und die konsumorientierten Bereiche auf das Konzept aufmerksam geworden und haben sich dessen angenommen. So unterstützen Roboter nicht nur industrielle Prozesse, sondern kommen seit vielen Jahren im konsumorientierten Umfeld auch in der Spielzeugindustrie und als Haushaltshelfer vor. Für diese konsumorientierten und auch bekannten industriellen Zwecken nimmt sich die Forschung vermehrt des Themas an und entwickelt weltweit neue Arten und Einsatzzwecke der Robotik und wertet diese im realen Einsatz in Punkten wie beispielsweise Effizienz, Mensch-Maschine-Kommunikation und Grenzen des Möglichen aus. So haben beispielsweise Einrichtungen wie die Firma *Boston Dynamics*, das *MIT (Massachusetts Institute of Technology)* und auch der chinesische Roboter-Hersteller *Unitree Robotics* in der nahen Vergangenheit an weit entwickelten vierbeinigen Robotern gearbeitet, die in ihrer Fortbewegung und dem Aussehen stark an Tiere wie Großkatzen oder Hunde erinnern. Akademische Einrichtungen wie das MIT setzen sich neben der Entwicklung dieser sogenannten *Quadruped Robotern* ebenfalls mit der Erforschung möglicher Einsatzzwecke und deren Sinnhaftigkeit auseinander. Auch die Möglichkeiten und Grenzen solcher Roboter müssen stets getestet werden, besonders, wenn neue Modelle mit neuen Versprechungen und Funktionen auf den Markt kommen.

Mit diesem Thema setzt sich diese Arbeit auseinander. Im Rahmen des Erwerb zweier Quadruped Roboter des Typs *GO1 (Unitree Robotics Go1 Edu)* werden die Roboter initial in zwei Phasen untersucht und deren Funktionen getestet und etwaige Schwierigkeiten dokumentiert. Die erste Phase wird sich im Rahmen dieser Arbeit mit den Robotern selbst auseinandersetzen und die Stärken und Schwächen der Geräte erarbeiten, die zweite Phase im Rahmen einer weiteren Arbeit[1] setzt sich mit dem tatsächlichen Einsatz des Roboters in einem ersten möglichen Umfeld auseinander.

## 1.1 Unitree Robotics GO1 Edu

Um eine Grundlage für die weitere Arbeit zu verschaffen, soll im folgenden Kapitel kurz auf den Roboter eingegangen werden, um den es sich in allen nicht Grundlagen-orientierten Erarbeitungen dieser Arbeit gehen wird.

Der Roboter-Hersteller *Unitree* wurde 2016 von Wang Xingxing in der chinesischen Provinz Zhejiang gegründet. Nach ersten Ausführungen eigener Quadruped Roboter wurde im Juni 2021 der Roboter GO1 veröffentlicht. Dieser wird seither vom Hersteller als erster bezahlbarer und konsumorientierter Quadruped Roboter und somit als Meilenstein in der zugänglichen Robotik beworben.[2] Der Roboter ist laut Herstellerangabe für rund 2700 USD (US-Dollar) in der Grundausstattung (GO1 Air) erwerblich, welche jedoch keine Möglichkeit bietet, die Funktionalitäten zu erweitern oder auf die internen Komponenten des Roboters zuzugreifen. Neben der Grundausstattung bietet Unitree den GO1 noch in den Varianten *Pro* und *Edu*, welche gegenüber dem *Air* einige Erweiterungen in Sensorik und Rechenleistung mitbringen. Der GO1 Edu ist die umfangreichste und teuerste Version des GO1 und besitzt eine umfangreiche Ausstattung an Sensoren, Kameras und Recheneinheiten, welche später in Kapitel 3 genauer aufgelistet werden. Diese Arbeit beschäftigt sich ausschließlich mit dem Unitree GO1 Edu. Abbildung 1 zeigt den GO1, wie er auf der Herstellerwebsite beworben wird.

Im Bearbeitungszeitraum dieser Arbeit ist bereits der Nachfolger in der Baureihe *GO* erschienen – der Unitree GO2, welcher durch einen noch günstigeren Einstiegspreis von beworbenen 1600 USD und den Spezifikationen nach zu urteilen deutlich verbesserten und erweiterten Funktionalitäten als verbraucherfreundliche Alternative zum GO1 angesehen werden kann.

[1] Jonas Kemnitzer. „Entwicklung eines Intelligenten lidarbasierten 3D Navigationssystems für den Unitree GO1“. Masterarbeit. Hochschule für Angewandte Wissenschaften Hof, 2023.

[2] Unitree Robotics. *About Unitree Robotics*. URL: `https://www.unitree.com/en/about/` (besucht am 28. 08. 2023).

[3] Unitree Robotics. *Unitree Go1*. URL: `https://shop.unitree.com/cdn/shop/products/75_540x.jpg?v=1668073774` (besucht am 28. 08. 2023)

Abbildung 1: Verkaufsbild des Unitree GO1[3]

## 1.2 Vorgehensweise

Die folgende Arbeit besteht im Grunde aus vier Teilen, den Grundlagen, der Roboterarchitektur, der Analyse des Roboters und der Funktionserweiterung des Roboters. Im ersten Teil, den Grundlagen, soll ein allgemeiner Überblick über das Thema Robotik geschaffen werden, der besonders zur Einordnung des bearbeiteten GO1 in das Feld allgemein dienen soll. Darauf folgend wird der aktuelle Stand der Forschung und Industrie im Bereich der abgegrenzten Einordnung des Roboters in die Robotik gezeigt werden. Hierfür werden zuerst Arbeiten und Veröffentlichungen sowie Produkte im Bereich der Quadruped Robotik und deren Nutzung gezeigt. Danach sollen Ressourcen zum GO1 gezeigt werden, die besonders in der Arbeit mit dem Roboter nützlich sind. Hierunter fallen nicht nur offizielle Ressourcen, sondern auch verwandte Veröffentlichungen und ungeprüfte Ressourcen aus nicht-akademischem Umfeld, die aufgrund ihres Umfangs dennoch beachtet werden sollten. Als Abschluss der Grundlagen werden einige Herausforderungen bei der Arbeit mit dem GO1 aufgelistet, die zu Teilen im Laufe der Arbeit wieder aufgegriffen werden sollen.

Nach Schaffung der Grundlagen zum Thema Robotik allgemein und den spezielleren Informationen zum GO1 wird dieser genauer betrachtet. Hierfür wird der Aufbau des Roboters detailliert beschrieben und anhand von Grafiken und Fotografien veranschaulicht. Nach einem Überblick zum Aufbau des Roboters werden die verbauten mechanischen Komponenten, die Sensorik und die Recheneinheiten dokumentiert. Darauf folgt eine genaue Analyse der verbauten Recheneinheiten, deren einzelner Funktionen und der Kommunikation dieser

untereinander und mit dem Nutzer. Zuletzt sollen die technischen Limitierungen des GO1 gezeigt werden.

Aufbauend auf dem geschaffenen Wissen zum Aufbau des Roboters wird in der Analyse des Roboters die Funktion dessen dokumentiert. Nach der Beschreibung der Inbetriebnahme des GO1 wird der Großteil der Funktionen, die ab Werk möglich sind, dokumentiert und bewertet. Als Ergänzungen werden Funktionen, die im Rahmen der Arbeit nicht getestet werden konnten, noch aufgelistet.

Zum Abschluss werden einige Funktionen aus den vorigen Kapiteln aufgegriffen und erweitert. Es werden nützliche Funktionen zum realen Einsatz des Roboters entwickelt, als auch Möglichkeiten zur Eliminierung oder Relativierung einiger Limitierungen gezeigt. Diese sollen dann noch in die Zielsetzung der Arbeit eingeordnet werden, worauf die Arbeit mit dem GO1 noch einmal final bewertet werden soll.

## 1.3 Zielsetzung

Folgendes Kapitel soll die Erarbeitungen im Rahmen dieser Arbeit eingrenzen und die Ziele klar definieren. Eine Evaluation des Erreichten findet am Ende dieser Arbeit statt.

**Ziele**

Die Ziele dieser Arbeit leiten sich vom Titel dieser ab – *Integration eines Unitree Go1 Quadruped Roboters in ein Hochschul-Ökosystem*. Die genannte *Integration* des Roboters kann in verschiedenen Aspekte des Hochschulökosystems erreicht werden. Hierunter gehören die für diese Arbeit relevanten Aspekte *Forschung*, *Lehre* und *Nutzen*. Die Nutzung des Roboters im Hochschulökosystem soll in dieser Arbeit nur exemplarisch dargestellt werden. Erarbeitungen, die eine gewisse Einordnung der Erkenntnisse bedürfen, können mögliche Nutzungen exemplarisch darstellen. Eine tatsächliche Umsetzung eines Projektes anhand des Roboters zur Nutzung auf dem Hochschulgelände ist jedoch nicht im Rahmen dieser Arbeit umgesetzt.

Der zweite relevante Aspekt des Ökosystems ist die Forschung, welche in Kombination mit der Lehre besonders von der Anschaffung eines GO1 durch die Hochschule profitieren kann. Als Grundlage für weitere Arbeiten am Roboter, besonders des Testens der Grenzen, des Erforschens neuer Einsatzmöglichkeiten oder der Interaktion des Roboters mit dem Menschen, sollen ein Großteil der Funktionen genau getestet und ihre Limitierungen, wenn vorhanden, dargestellt werden.

Im Bereich der Lehre gilt ein ähnliches Prinzip, die Darstellung der Funktionen und die Dokumentation der Einzelheiten des Roboters soll es den Lehrenden und Studierenden

vereinfachen, Arbeiten und Versuche am Roboter durchzuführen. Auch die Dokumentation des Aufbaus soll es Neulingen im Bereich der Quadruped Robotik erleichtern, sich einen Überblick über die Funktion eines solchen Roboters zu verschaffen.

Zusammengefasst gilt diese Arbeit als eine Art *Survey* zum GO1 und dessen Funktionen. Hierbei wird dieser genau inspiziert, all dessen Komponenten dokumentiert, die wichtigsten Funktionalitäten getestet und dokumentiert und durch Erweiterungen die gravierendsten Limitierungen des Roboters behoben sowie die einfachsten Anwendungen dessen erleichtert.

**Abgrenzung**

Der Roboter enthält diverse Vorbereitungen zum Thema KI (Künstliche Intelligenz), welche in dieser Arbeit maximal erwähnt werden sollen. Auch die Umsetzung dieser Funktionen wird nicht dokumentiert. Unter diese Funktionen fallen Themen wie Kamerabildoptimierung, Umgebungserkennung durch Ultraschall und Lidar (Light Detection and Ranging), Objekterkennung durch die diversen Sensoren und Optimierung der Bewegungsabläufe. Auch die mechanischen und elektrotechnischen Grundlagen der Motorik und des Bewegungsapparats werden nicht weiter geschildert.

# 2 Grundlagen

Folgendes Kapitel beschreibt die Robotik und der möglichen Einordnung des GO1 in diese. Hierfür wird die Robotik oberflächlich definiert, eingeteilt und die relevanten Teilbereiche genauer betrachtet. Da sich diese Arbeit mit einem GO1 beschäftigt, wird dieser klassifiziert und in die Bereiche der Forschung, Industrie und weiterer Nutzung eingeordnet. Zum weiteren Verständnis des Ziels der Arbeit – der Analyse und Integration des GO1 in ein bestehendes Ökosystem - wird der aktuelle Stand der Forschung genauer betrachtet. Relevant sind hier besonders die bereits erarbeiteten Erkenntnisse der Nutzung gleicher oder ähnlicher Roboter, als auch die Einbindung anderer Modelle in bestehende Ökosysteme. Hierfür soll nicht nur die Forschung allein betrachtet werden, sondern auch die Arbeit privater Unternehmen und Entwickler. Mit dem Wissen der korrekten Einordnung des GO1 und dem Forschungsstand zu verwandten Themen sollen die Herausforderungen dieser Arbeit hervorgehoben werden.

## 2.1 Robotik

Die Robotik befasst sich mit dem Wissensgebiet rund um *Roboter*. Der Begriff *Roboter* stammt vom tschechischen Wort *robota*, was so viel wie *Frondienst* oder *Zwangsdienst* bedeutet. So treffend diese Übersetzung auf die heutige Nutzung des Wortes ist, so vage ist diese Beschreibung auch. Ähnlich unklar sind auch die anerkannten Definitionen des Wortes *Roboter*. Eine gängige Definition im deutschsprachigen Raum ist die des VDI (Verband Deutscher Ingenieure):

> [Roboter] sind universell einsetzbare Bewegungsautomaten mit mehreren Achsen, deren Bewegungen hinsichtlich Bewegungsfolge und Wegen bzw. Winkeln frei (d.h, ohne mechanischen Eingriff) programmierbar und ggf. sensorgeführt sind[4].

Auch wenn die VDI-Richtlinie mittlerweile zurückgezogen wurde, wird die Definition aufgrund ihrer treffenden Eingrenzung des Begriffes noch häufig zitiert.

Eine genaue und anerkannte Definition des Wortes *Roboter* findet sich bis heute kaum, was auch die Einteilung der Robotik in Teilbereiche erschwert. Da für diese Arbeit und die Einordnung des GO1 keine aufwändige Kategorisierung der Robotik nötig ist, wird eine Aufteilung der Roboter in zwei Klassen verwendet, Industrieroboter und Serviceroboter. Abbildung 2 zeigt die beiden Klassifizierungen und deren Eigenschaften im Überblick.

[4] VDI-Gesellschaft Produktionstechnik. *VDI 2860/ Montage- und Handhabungstechnik. Handhabungsfunktionen, Handhabungseinrichtungen; Begriffe, Definitionen, Symbol*. Beuthe Verlag, 1990.

[5] Sophie Fischer. *Robotics – Market data analysis & forecasts*. Statista Technology Market Outlook. Statista, Aug. 2022. URL: `https://de.statista.com/statistik/studie/id/116785/dokument/robotics-report/` (besucht am 04. 07. 2023)

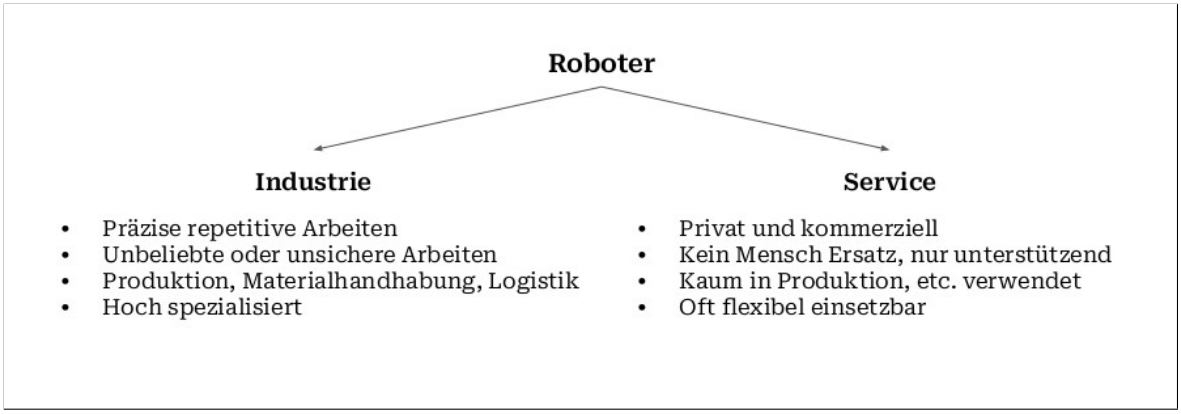


Abbildung 2: Vergleich zwischen Industrie- und Servicerobotern[5]

### 2.1.1 Industrieroboter

Industrieroboter sind Roboter, die im industriellen Umfeld eingesetzt werden, den Mensch meist in nicht sicheren, nicht rentablen oder nicht begehrenswerten – repetitiven – Arbeiten ersetzen. Sie sind aufgrund ihrer hohen Spezialisierung an den Einsatzzweck großenteils stationär. Der Vorteil industriell eingesetzter Roboter ist die hohe Effizienz und Genauigkeit der Arbeiten und die Anpassbarkeit an den Einsatzzweck. Als Nachteil ergibt sich hieraus der hohe Aufwand der Implementierung, da ein hoch spezialisierter Roboter für jeden Einsatzzweck neu geschaffen werden muss. Einsatzzwecke von Industrierobotern sind unter anderen das Schweißen, Lackieren, Montieren und die Materialhandhabung in Produktionsumgebungen.

Die Limitierung des Einsatzes und die strikte Einteilung zu industriellen Einsätzen lässt schlussfolgern, dass der GO1 nicht in die Klasse der Industrieroboter eingegliedert werden kann. Stattdessen kann man ihn in der in dieser Arbeit verwendeten, einfachen Unterteilung den Servicerobotern zuteilen.

### 2.1.2 Serviceroboter

Serviceroboter unterscheiden sich im Kern von Industrierobotern in der Annahme, dass sie Menschen in ihrem Einsatzzweck nicht ersetzen, sondern unterstützen oder von Menschen unterstützt werden. Der Begriff *Serviceroboter* ist absichtlich weit gefasst und schließt im Umfang dieser Arbeit lediglich industrielle Roboter, so wie sie im vorigen Paragrafen beschrieben werden, aus. Bei Servicerobotern lässt sich grundsätzlich noch zwischen kommerziell eingesetzten und konsumorientierten Robotern unterscheiden[6].

Serviceroboter sind in der Regel nicht stationär, da sie in ihren Einsatzmöglichkeiten deutlich flexibler sind, als Industrieroboter. Aus dem Vorteil der Flexibilität lässt sich ebenfalls

[6]Fischer, s. Anm. 5.

der Nachteil der Komplexität ableiten. Das weitere Einsatzfeld der Serviceroboter steigert in der Regel den Aufwand der Entwicklung, steigert aber ebenfalls den Ertrag des Roboters.

Möchte man die Klasse der Serviceroboter weiter untergliedern, so sind unter anderen folgende Untergliederungen denkbar:

- Spielzeugroboter
- Erkundungsroboter
- Militär-/ Kampfroboter
- Assistenzroboter

Zu vermerken ist, dass die simple Klassifizierung der Roboter in lediglich zwei Klassen der Industrieroboter und der Serviceroboter gewählt wurde, um die Einsatzzwecke des GO1 in dieser Arbeit nicht einzuschränken. Wie bereits erarbeitet kann man den GO1 vielseitig einsetzen, jedoch ist er nicht geeignet, Menschen in seinen Einsatzgebieten vollständig zu ersetzen, noch ist er strikt auf industrielle Zwecke beschränkt. Eine Zuteilung zu industriellen Robotern ist somit nicht möglich. Eine genauere Klassifizierung innerhalb der Klasse der Serviceroboter ist zwar möglich, jedoch vom finalen Einsatzzweck des Quadruped Roboters abhängig. Mögliche Einsatzzwecke im Hochschulumfeld werden in Kapitel 2.2.2 angerissen.

### 2.1.3 Cobots

Ein weiterer passender Begriff, der der Klassifizierung des GO1 als Serviceroboter nicht widerspricht, ist *Cobot*. Das Wort *Cobot* setzt sich aus dem englischen Wort *collaborate* – (zusammenarbeiten, kollaborieren) – und dem Wort *Roboter* zusammen. Einige Merkmale sind:[7]

- Kollaborativ und sicher, im Gegensatz zu stationär und abgesichert
- Interaktiv und adaptiv zur Umgebung
- Einfache Inbetriebnahme durch vorausschauende Entwicklung
- Flexible Einsatzzwecke
- Schnelleres ROI (Return on Investment)

[7] Fischer, s. Anm. 5.

Durch seine flexiblen Fortbewegungsmöglichkeiten und der hohen Anzahl an Sensorik und Erweiterungsmöglichkeiten des GO1 sowie den zur Kommunikation eingebauten Lautsprecher[8] lässt sich der GO1 neben der Klassifizierung als Serviceroboter ebenfalls als *Cobot* bezeichnen.

### 2.1.4 Quadrupedie

Die letzte Einordnung des GO1 ist die in den Begriff der Quadruped Robotik. Laut dem *Lexikon der Biologie* ist die *Quadrupedie* eine vierbeinige Fortbewegungsart, in der alle vier Extremitäten in Kontakt zum Boden stehen[9]. Quadruped Roboter sind also stark an die Biologie angelehnt, was die restliche äußere Form des GO1 bestätigt. Der Begriff *Quadruped Roboter* bedeutet somit nur, dass der gemeinte Roboter vier Beine nutzt, um sich fortzubewegen.

## 2.2 Stand der Forschung

Im Bereich der Quadruped Roboter ist bereits viel erforscht worden. Besonders im Fokus der Arbeiten sind in der Regel die Steuerung der Motoren zur Fortbewegung, die autonome Navigation der Roboter in unbekanntem Umfeld und das Testen der Einsatzmöglichkeiten der vierbeinigen Roboter. Diese Arbeit hingegen beschäftigt sich mit der Integration des Roboters in ein Hochschulökosystem, welche die möglichen Einsatzzwecke nicht vorwegnimmt. Deshalb werden im folgenden nur kurz allgemein interessante Forschungen an Quadruped Robotern gezeigt, wonach allgemeine Forschungen zur Einbindung und Nutzung von Servicerobotern dargestellt werden. Zuletzt sollen noch Arbeiten speziell zum Modell GO1 gezeigt werden. Diese sind nicht zwingend akademischen Ursprungs und sollen dem Leser der Arbeit einen Überblick über die vorhandenen Ressourcen zum Modell GO1 bieten.

### 2.2.1 Quadruped Roboter

Die erste Umsetzung eines Quadruped Roboters, die öffentliche Aufmerksamkeit erlangte, ist der sogenannte *Big Dog* der Firma *Boston Dynamics* aus Boston, USA (Vereinigte Staaten von Amerika). Dieser wurde 2008 aus militärischem Interesse an einer geländegängigen Alternative zu gängigen Militärfahrzeugen entwickelt und deshalb auch vom amerikanischen DARPA (Defense Advanced Research Projects Agency) finanziert. *Big Dog* wurde mit hydraulischen Extremitäten entwickelt, welche ihm ermöglichen sollten, schweres Gepäck im Militäreinsatz tragen zu können. Die Flexibilität der vier Beine und die Möglichkeit

[8] Siehe Kapitel 3.1.3
[9] Spektrum. *Lexikon der Biologie*. Spektrum Akademischer Verlag Heidelberg, 1999.

dieser, sich relativ schnell durch schweres Gelände bewegen zu können, gaben *Big Dog* einen möglichen Vorteil gegenüber gängigen Fortbewegungsmitteln auf Basis von Ketten oder Rädern. Weitere Entwicklungen am *Big Dog* wurden später vom RCTA (Robotics Collaborative Technology Alliance) des US (United States) Army Research Laboratory finanziert.[10,11]

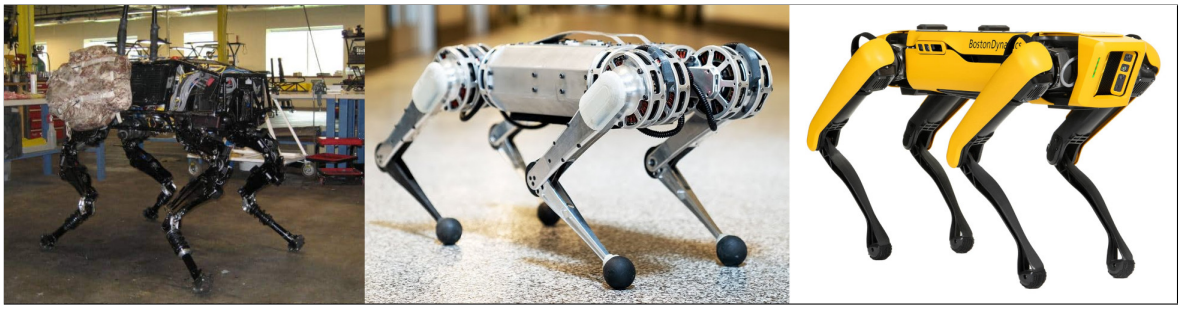

Abbildung 3: Big Dog (links), MIT Cheetah 3 (mitte) und Spot (rechts)[12]

2009 hat das *Biometric Robotics Lab* des MIT ein Projekt namens *Cheetah* – auf Deutsch *Gepard* – bekannt gegeben. Das Projektziel war es, einen Quadruped Roboter zu entwickeln, der in den Bereichen Tempo und Energieeffizienz der echten Tierwelt Konkurrenz macht. Auch dieses Projekt wurde vom DARPA finanziert[13,14]. Im Gegenzug zum *Big Dog* wurde der *Cheetah* aber nie kommerziell oder militärisch beworben. Stattdessen liegt der Fokus des Roboters im Bereich der Forschung. Bereits 2012 veröffentlichte das *Biometric Robotics Lab* die ersten Videos des Roboters im Lauf, was impliziert, dass der Roboter zu dieser Zeit bereits fertiggestellt wurde. Seit der Projektankündigung wurden mehrere Iterationen des Roboters entwickelt. Die aktuelle Iteration des Roboters ist in ihrer Bauart optimiert und somit kostentechnisch effizient im Einkauf und in der Reparatur. Im Gegensatz zum *Big Dog* ist der MIT *Cheetah* voll elektrisch, was das Ziel der einfachen Entwicklung hat.[15] Viele

[10] Marc Raibert u. a. *BigDog, the Rough-Terrain Quaduped Robot*. Boston Dynamics, 8. Apr. 2008.

[11] Defense Advanced Research Projects Agency. *Big Dog*. URL: `urhttps://www.darpa.mil/about-us/timeline/big-dog` (besucht am 07. 08. 2023).

[12] Boston Dynamics. *Foto eines BigDog*. URL: `https://www.bostondynamics.com/wp-content/uploads/2023/07/bigdog-1-1.jpg` (besucht am 28. 08. 2023)

MIT. *Foto eines Cheetah 3*. URL: `https://news.mit.edu/sites/default/files/styles/news_article__image_gallery/public/images/201903/MIT-Mini-Cheetah-01_0.jpg` (besucht am 28. 08. 2023)

Boston Dynamics. *Foto eines Spot*. URL: `https://bostondynamics.com/wp-content/uploads/2023/05/spot-explorer-web-2-2048x1152.jpg` (besucht am 28. 08. 2023)

[13] Devan Joseph. „MIT reveals how its military-funded Cheetah robot can now jump over obstacles on its own“. In: *Business Insider* (3. Juni 2015).

[14] Defense Advanced Research Projects Agency. *Maximum Mobility and Manipulation (M3)*. URL: `https://www.darpa.mil/program/maximum-mobility-and-manipulation` (besucht am 07. 08. 2023).

[15] Jason Falconer. „MIT Cheetah Robot Runs Fast, and Efficiently. It's now the second fastest legged robot in the world“. In: *IEEE Spectrum* (14. Mai 2013).

Erkenntnisse aus dem MIT *Cheetah* Projekt werden heute noch in neuen Projekten verwendet. So auch bei der Umsetzung des GO1, worauf in dieser Arbeit aber weniger eingegangen wird.

Die aktuell vermutlich bekannteste Umsetzung eines Quadruped Roboters ist der von *Boston Dynamics* entwickelte *Spot*, welcher in seiner ersten Form bereits 2015 öffentlich beworben wurde. Anders als seine Vorgänger Roboter bei *Boston Dynamics* war er der erste Roboter der Firma, der zu 100% elektrisch betrieben wurde. Das lässt ihn sehr ähnlich zum *Cheetah* des MIT wirken, jedoch hatte die Plattform *Spot* von Anfang an den Fokus auf einen kommerziellen oder auch militärischen Einsatz. Besonders nennenswert an dem Roboter ist die Kompaktheit seinen Vorgängern gegenüber. Die aktuelle Version des *Spot* ist bereits kommerziell verfügbar und wurde initial für etwa 75 000 US-Dollar[16] zum Kauf beworben.[17]

Die Veranschaulichung *The Rise of the Robot Quadrupeds* gibt einen Überblick über die Zeitschiene der einflussreichsten Implementierungen von Quadruped Robotern in der jüngeren Vergangenheit[18].

### 2.2.2 Integration von Robotern

Im Bereich der Robotik liegt neben dem Fokus des Testens der Möglichkeiten immer die Suche nach einer sinnvollen Nutzung der gewonnenen Erkenntnisse und der entwickelten Roboter in realen Szenarien. Hierbei sind die Möglichkeiten kaum limitiert und hängen stets von der Form und Reife der Roboterimplementierungen ab. Im Folgenden werden einige aktuelle Ansätze zur Integration und Nutzung von Quadruped Robotern in realen Szenarien vorgestellt.

Die unten genannten Einsatzmöglichkeiten von Quadruped Robotern sind lediglich eine Auswahl des offensichtlichen Potenzials und sollen lediglich einen Überblick schaffen. Eine vollständige Auflistung der Integrationsansätze ist hier nicht angestrebt.

**Kontrolle und Überwachung** Quadruped Roboter sind aufgrund ihrer Fähigkeit, sich in unstrukturierten Umgebungen fortzubewegen, gut dafür geeignet, in gefährlicheren Gebieten die Kontrollfunktion des Menschen zu übernehmen. Zudem kann ein weit genug entwickelter Roboter autonom Aufgaben übernehmen und diese in regelmäßigen Intervallen durchführen, was einen großen Vorteil dem Menschen gegenüber darstellt. Eine mögliche Einsatzmöglichkeit von Quadruped Robotern ist laut des Artikels „Real-Time and Remote Construction

[16] Evan Ackerman. „Boston Dynamics' Spot Robot Dog Now Available for $74,500. For the price of a luxury car, you can now get a very smart, very capable, very yellow robotic dog". In: *IEEE Spectrum* (16. Juni 2020).

[17] Boston Dynamics. *Legacy Robots*. URL: `https://bostondynamics.com/legacy/` (besucht am 08. 08. 2023).

[18] Jeremy Moses und Geoffrey Ford. *The Rise of the Robot Quadrupeds*. 11. Dez. 2020. URL: `https://mappinglaws.net/rise-robot-quadrupeds.html` (besucht am 07. 08. 2023).

Progress Monitoring with a Quadruped Robot Using Augmented Reality“ das Überwachen und Prüfen des Fortschritts auf Baustellen. Die Autoren bezeichnen die manuelle Kontrolle einer Baustelle durch einen menschlichen Inspektor als zeitlich ineffizient, unstrukturiert und unzuverlässig. Sie bemängeln zudem, dass die menschlichen Kontrollen aufgrund kleinster Abweichungen in der Systematik kaum wiederholbar sind und somit schlecht einzuordnen sind. Ihr Ansatz, die zu kontrollierenden Umgebung zu virtualisieren und dann anhand automatisierter Abläufe und Scans durch Quadruped Roboter mit den zu erzielenden Ergebnissen abzugleichen, birgt zwar einige Hürden, verspricht jedoch ein konsistentes Ergebnis über den zeitlichen Verlauf der Baustelle hinweg. Zudem heben die Autoren die zeitliche Ersparnis und somit die Effizienz des Ansatzes hervor, da selbst bei einer nur teilweise automatisierten Umsetzung ein verteiltes Monitoring möglich ist, bei dem sich nur der Roboter mit Sensorik und Kameras vor Ort befinden muss.[19]

Im Bereich der Integration eines solchen Quadruped Roboters im Hochschulumfeld lässt sich dieser Ansatz beispielsweise in der automatisierten Modellierung des Campus übersetzen. Ein vorheriges Erstellen eines exakten Modells ist bei diesem Ansatz nicht nötig, man macht sich hier lediglich die Möglichkeit der Modellierung durch Sensorik wie beispielsweise Ultraschall, Lidar und Kameras zu nutzen.

**Sicherheit** Da die Umgebung in urbanen Umgebungen zu großen Teilen unstrukturiert ist, is der Einsatz dedizierter Roboter für den Betrieb auf Straßen oder in der Luft oft erheblich eingeschränkt. Neben der flexibleren Manövrierbarkeit von Quadruped Robotern ist die Erweiterbarkeit dieser ein weiterer Vorteil gegenüber einfacheren Robotern auf Basis von Rädern oder Dronen. So ermöglicht die Erweiterung eines serienmäßigen *Spot* Roboters durch einen Arm mit Greifer laut der Studie „Robotics’ Role in Public Safety“ die Gefahrenminimierung für Menschen in Einsatzzwecken wie der Kontrolle verdächtiger Objekte und dem möglichen Entfernen dieser aus einer Gefahrenzone, dem Aufspüren potenziell gefährlicher Substanzen in der Umgebung (Strahlung, Leckagen, Flüssigkeiten) oder der vorzeitigen Prüfung einer Umgebung, um den Einsatzkräften in Notfallsituationen einen besseren Überblick zu verschaffen, ohne sich selbst in Gefahr zu bringen. *Boston Dynamics* heben in ihrer Studie besonders die hohe Modularität von Quadruped Robotern hervor, welche sie flexibel einsatzbar macht.[20]

Für den GO1 im Hochschulumfeld wäre im Sinne der Sicherheit eine Nutzung des Roboters bei Brandbildung zur Suche nach Verletzten denkbar. Auch ein Ablaufen der potenziell

[19] Srijeet Halder u. a. „Real-Time and Remote Construction Progress Monitoring with a Quadruped Robot Using Augmented Reality“. In: *Buildings* (Dez. 2022). URL: `https://doi.org/10.3390/buildings12112027`.

[20] Boston Dynamics. „Robotics’ Role in Public Safety. How robots like Boston Dynamics’ Spot are keeping people safe.“ In: (2023).

gefährlichen Labore und der Prüfung nach Leckagen ist denkbar. Zu Schließzeiten könnten Quadruped Roboter wie der GO1 ebenso genutzt werden, um den Campus nach nicht autorisierten Besuchern abzusuchen.

**Automatisierung und Effizienz**

Wie in Kapitel 2.1 erläutert, bringt die Herkunft des Wortes *Roboter* eine der Kernziele der Robotik zum Vorschein. Die Automatisierung und Entlastung des Menschen bei einfachen oder wenig begehrenswerten Aufgaben liegt hier im Fokus. So können Roboter einfache und repetitive Aufgaben übernehmen und teilweise mit höherer Präzision und Ausdauer übernehmen, als der ersetzte Mensch dies hätte tun können. Übersetzt man diese Erkenntnis auf Quadruped Roboter, so ergeben sich in unstrukturierten Umgebungen, wie besonders der Landwirtschaft, neue Automatisierungsmöglichkeiten. Die Arbeit *Towards Computer-Vision Based Vineyard Navigation for Quadruped Robots* beschreibt beispielsweise die Nutzung eines Quadruped Roboters, um im unebenen Umfeld von Weingärten Weinreben automatisch zu kontrollieren und mithilfe einer Erweiterung auch zu kürzen, sodass die Pflanze keine unnötigen Ressourcen verschwendet. Gleiches lässt sich auch auf andere Gebiete der Landwirtschaft übersetzen. So könnten Quadruped Roboter Kontrollgänge von Jägern und Förstern teilweise ersetzen oder die Frequenz dieser erhöhen.

Auch hier lässt sich die Brücke zur Integration in ein Hochschulökosystem schlagen. Ein automatisierter Roboter könnte auf mehreren Ebenen eines Hochschulgebäudes beispielsweise die Schließzeiten kontrollieren und durchsetzen. Ein Ablaufen und Kontrollieren der Schlösser und gegebenenfalls ein Abschließen dieser ist mit wenig Erweiterungsaufwand möglich.

**Service** Auch im Bereich der allgemeinen Unterstützung des Menschen ist der Quadruped Roboter besonders in unstrukturierten Umgebungen hilfreich. Der bereits in Kapitel 2.2.1 beschriebene *Big Dog* wurde beispielsweise entwickelt, um in Kriegsgebieten und Kampfeinsätzen schweres Gepäck befördern zu können. Dies entlastet den menschlichen Soldaten ungemein und ermöglicht ihm, andere Aufgaben zu übernehmen.[21] Dieser Ansatz des Lastentragens ist auch im außermilitärischen Bereich denkbar. Ein weiterer Ansatz ist der der Navigationshilfe. Durch die flexible Fortbewegung über unebene Untergründe kann ein solcher Quadruped Roboter einem Menschen bei der Navigation zu einem Ziel helfen.

Dieses Szenario ist auch im Hochschulumfeld denkbar. So könnte der Roboter neue Gäste empfangen und auf Anfrage über den Campus hinweg Navigationshinweise zu gesuchten Orten liefern und die Gäste auch begleiten.

[21] Raibert u. a., s. Anm. 10.

### 2.2.3 Unitree Go1 Ressourcen

Im Folgenden werden einige für die Arbeit am GO1 hilfreiche Ressourcen aufgelistet und bewertet. Diese sind ausschließlich nicht nur akademischer Art und müssen dementsprechend mit besonderer Vorsicht betrachtet werden. Dennoch sind diese Ressourcen besonders hilfreich und können eine gute Grundlage und Ergänzung zu den Ergebnissen im Verlauf der Arbeit bieten.

**Offizielle Dokumentation**

Die Webseite `https://www.docs.quadruped.de/projects/go1/html/index.html` [22] bietet einen grundlegenden Überblick über den Aufbau des GO1 und die Funktionen, die im Lieferumfang des Roboters mit enthalten sind. Sie enthält alle nötigen Informationen, die die Entwickler des Roboters gesammelt und veröffentlicht haben. Die Seite bietet einen guten Einstieg, um die Anweisungen direkt am Roboter anzuwenden, ohne sie weiter nachvollziehen zu können.

Auf der Unterseite `/operation.html` sind auch alle offiziellen Anleitungen dokumentiert, welche als PDF (Portable Document Format) heruntergeladen werden können. Des weiteren enthält die Webseite einige einfache Konfigurationsanleitungen für die Software-Erweiterungen, welche ab Werk installiert und freigeschaltet sind.

Neben der allgemeinen Dokumentation des Roboters sind auch einige *GitHub*-Repositories und Dokumentationen zu einzelnen Funktionen des GO1 bereitgestellt. Die wichtigsten sind:

- **Unitree-Legged-SDK**
  `https://github.com/unitreerobotics/unitree_legged_sdk`
  → Eine Bibliothek, um die Steuerung des Roboters zu beeinflussen

- **Unitree-Camera-SDK**
  `https://github.com/unitreerobotics/UnitreecameraSDK`
  → Eine Bibliothek, um die Kameras des Roboters zu nutzen

- **Unitree-Actuator-SDK**
  `https://github.com/unitreerobotics/unitree_actuator_sdk`
  → Eine Bibliothek, die die verbauten Motoren einzeln steuert

- **Unitree-ROS2-to-Real**
  `https://github.com/unitreerobotics/unitree_ros2_to_real`

[22] Unitree Robotics. *GO1 Manuals - GO1 Tutorials 1.0.0 documentation*. URL: `https://www.docs.quadruped.de/projects/go1/html/index.html` (besucht am 08. 08. 2023).

→ Eine Bibliothek, welche das veraltete ROS (Robot Operation System) (1) auf den Robotern ersetzen kann

Anzumerken ist, dass die offizielle Dokumentation auf der Webseite und den GitHub-Repositories zwar einen guten Einstieg in die direkte Nutzung des Roboters liefern, jedoch meist unverständlich geschrieben, unvollständig und teils nicht übersetzt, sondern in der Originalsprache *Chinesisch* veröffentlicht wurden. Deshalb wird in dieser Arbeit dazu geraten, die später gelisteten alternativen Wissensquellen ebenfalls zu nutzen, wenn auch mit kritischem Blick, da vieles nicht offiziell bestätigt oder geprüft wurde.

**MIT Cheetah Dokumentation**

Der mechanische Aufbau und die allgemeine Erscheinung des GO1 sind der des MIT *Cheetah* 3 ähnlich und in Teilen sogar identisch. Ein optischer Vergleich kann durch Abbildung 3 durchgeführt werden. Es ist also anzunehmen, dass *Unitree Robotics* Teile der Ergebnisse der Forschung am MIT *Cheetah* 3 für die Entwicklung des GO1 wiederverwertet haben. Wenn auch kurz nach Veröffentlichung unklar, wird die Lizenz der MIT-Software, die im GO1 verwendet wird, mittlerweile anerkannt und referenziert. Aus diesem Grund ist es ratsam, die hervorragend dokumentierten Erkenntnisse über den MIT Cheetah 3 zur Hand zu nehmen, wenn man genaueres über den Aufbau und die Funktion des GO1 wissen will. Ein Einstieg gewährt hier der Artikel „MIT Cheetah 3: Design and Control of a Robust, Dynamic Quadruped Robot“, welcher parallel zur Veröffentlichung des Cheetah 3 erschien.

Seit der Veröffentlichung des Cheetah 3 sind seitens des MIT weitere Forschungsartikel über diesen erschienen. Hier hilft ein Blick auf die Webseite des Instituts *MIT Biometric Robotics Lab* – `https://biomimetics.mit.edu/publications` – an welchem der Cheetah 3 und seine Vorgänger entwickelt wurden. Für einen technischeren Überblick über die Hard- und Software-Steuerungen des MIT Cheetah 3 und somit auch des GO1 ist das *GitHub*-Repository des Biometrics Robotic Lab hilfreich[23].

Im Gegensatz zur offiziellen Dokumentation der Firma *Unitree Robotics* sind die Erkenntnisse aus der MIT Dokumentation fundiert und nachvollziehbar. Leider beziehen sich die Überschneidungen der beiden Roboter nur auf allgemeine Konzept und Teile der konkreten Umsetzung des GO1, die genaue Funktionalität ist bei beiden Robotern sehr unterschiedlich.

**Innoffizielle Repositories**

Durch die niedrige finanzielle Hürde beim Erwerb eines GO1 hat sich nach Veröffentlichung des Roboters schnell eine vergleichsweise große und aktive Community gebildet, welche sich

[23] `https://github.com/mit-biomimetics/Cheetah-Software`

in der Arbeit mit dem GO1 unterstützt und untereinander austauscht. Innerhalb der Community haben sich einige wenige Teilhaber durch das Anlegen gut dokumentierter Repositories mit Anwendungsbeispielen und Implementierungen neuer Funktionen hervorgetan. Da die realen Namen dieser Nutzer aus den Foren und Repositories nicht hervorgehen, werden ihre Referenzen hier gewürdigt und eingeordnet.

- `https://github.com/MAVProxyUser/YushuTechUnitreeGo1`
  Ein sehr detailreiches Repository, welches eine Nutzer-fokussierte Herangehensweise an die Dokumentation des Roboters pflegt. Der Autor beschreibt den Aufbau nicht nur in einfachen Worten, sondern gibt auch einfache Anweisungen zu Konfiguration und Nutzung einzelner Funktionen. Besonders zum Einstieg eine gute Anleitung, um der erweiterten Nutzung näherzukommen.

- `https://github.com/maggusscheppi/Go1`
  Ein weniger detailliertes Repository, welches jedoch gute Ansätze zur Lösung für gängige Probleme aufweist und zudem pragmatische Implementierungen neuer Funktionen enthält. Empfehlenswert im Bereich Batteriemanagement und Monitoring[24].

- `https://github.com/Bin4ry/free-dog-sdk`
  Ein aufwändiges Repository, welches sich mit dem Reverse-Engineering der offiziellen Unitree Bibliotheken beschäftigt. Die erarbeiteten Ergebnisse des Autors können genutzt werden, um die Versionen *Air* und *Pro* zu Teilen mit den Funktionen des *Edu* auszustatten oder um die Bibliotheken besser nachvollziehen zu können.

**Inoffizielle Foren**

Wie bereits erwähnt, ist die Nutzerbasis und die Community um das Thema GO1 nach Veröffentlichung und Verkaufsstart des Roboters stark gewachsen. Neue Nutzer des GO1 sollten sich demnach nicht scheuen, nach Foren und ähnlichen Ressourcen zu suchen, um sich bei Problemen an erfahrenere Besitzer des Roboters zu wenden. Das wohl präsenteste und auch aktivste Forum ist der *Slack*-Kanal *The Dog Pound animal control for Stray robot dogs*[25]. Der Beitritt neuer Nutzer ist erwünscht, die aktive Beteiligung ebenfalls. Der Link in der Fußnote kann genutzt werden, um dem Kanal beizutreten, eine Nutzung der Ressource ohne Beitritt ist leider nicht möglich.

[24] Siehe Kapitel 5

[25] `https://join.slack.com/t/robotdogs/shared_invite/zt-1fvixx89u-7T79~VxmDYdFSIoTnSagFQ`

## 2.3 Herausforderungen

Abschließend zu den Grundlagen über die Robotik und dem Stand der Forschung zum Thema Integration, Quadruped Robotik und dem GO1 sollen einige Herausforderungen geschildert werden, die bei der Arbeit mit dem Roboter zu beachten sind und die im Laufe der Arbeit teilweise gemeistert werden sollen.

**Dokumentation**

Wie aus dem vorigen Kapitel 2.2.3 entnommen werden kann, ist die offizielle Dokumentation der Firma *Unitree Robotics* unzureichend, weshalb auch ein Blick auf verwandte und inoffizielle Ressourcen zu empfehlen ist. Nicht nur die allgemeine Nutzung des Roboters in seinem Lieferzustand, sondern auch die mitgelieferten und erweiternden Bibliotheken sind kaum oder unzureichend beschrieben. Viele Bibliotheken zur Erweiterung der Grundfunktion sind so beispielsweise kaum kommentiert oder dokumentiert. Diese Herausforderung im Umgang mit dem GO1 soll im Laufe der Arbeit für große Teile der Grundfunktionalitäten und für die Erweiterungen in Kapitel 5 beseitigt werden. Ziel der Arbeit ist es unter anderem, die Nutzung und Arbeit am GO1 so zu dokumentieren, dass er im Hochschulumfeld zukünftig effizienter genutzt werden kann – als eigenständiger Teil des Ökosystems, aber auch als Erweiterung der Lehre.

**Veraltete Ressourcen und Bibliotheken**

Der 2021 veröffentlichte GO1 ist in den vergangenen Jahren, inklusive der nur zu schätzenden Entwicklungszeit des Roboters durch die Firma *Unitree Robotics*, in seinem Softwarestand teilweise gravierend veraltet. Der Softwareanteil des Roboters wird in Kapitel 3.2 in Verbindung mit den verbauten Hardwarekomponenten genauer betrachtet, weshalb hier nur zwei Probleme exemplarisch dargestellt werden.

1. **Betriebssysteme**
   Auf den internen Rechnern des GO1 ist ausschließlich Debianbasierte Software installiert, deren Stand mittlerweile veraltet und kaum noch gepflegt ist. Teile der installierten Bibliotheken sind dadurch außer Wartung genommen oder länger nicht aktualisiert worden. Das ist besonders in der Betrachtung der Sicherheit des Roboters relevant und kann zu unerwarteten Problemen führen. Eine Aktualisierung ist aufgrund der komplexen Konfigurationen und der vielzahl an nachinstallierten Bibliotheken nicht trivial.

2. **Robotik Software** Ähnlich zur Problematik der Betriebssysteme ist die Software zur Steuerung des Roboters – ROS – in der ersten Version installiert. Diese ist seit längerer

Zeit durch den Nachfolger ROS 2 abgelöst und mit moderner Software und dritten Bibliotheken und Erweiterungen kaum noch kompatibel. Das schränkt die Nutzung des Roboters stark ein und hatte sogar zur Folge, dass nur für dieses Problem eine Adapter-Bibliothek vom Hersteller selbst entwickelt wurde[26].

Ähnliche Probleme lassen sich in vielen Bestandteilen des Roboters wiederfinden und erschweren somit die Arbeit am Gerät. Diese Arbeit wird sich nur entfernt mit der Aktualisierung des Bestandes beschäftigen. Stattdessen wird in den Erweiterungen – wo relevant – in Kapitel 5 der Umgang mit der veralteten Software beschrieben.

**Unzureichende Qualität der Bestandsfunktionen**

Parallel zur mangelnden Qualität der Dokumentation steht die unzureichende Qualität der Bestandsfunktionen. Gemeint sind hiermit alle Funktionen, die ab Werk zur Auslieferung des GO1 verfügbar und funktionstüchtig waren. Darunter fallen Funktionen wie das Fortbewegen, die Kamerabildübertragung, die Sensorik und die Autonomie in der Verarbeitung und Reaktion auf gesammelte Daten. Einzeln betrachtet erfüllen alle Funktionen die Minimalanforderungen, kombiniert man jedoch einige Funktionen, so lässt die Qualität der Ergebnisse jeder einzelnen Funktion stark nach. Gründe hierfür sind oftmals die unausgewogene Zuteilung der Ressourcen bei parallel laufenden Prozessen, welche sowohl zu hoher Auslastung der Recheneinheiten als auch zur grundlegend hohen Auslastung aller Bausteine im Gesamten führt. Besonders im Bereich der Kombinationen aus Fortbewegung und Rechenleistung für Auswertungen der Videodaten und Sensorik kommt es häufig zu sogenanntem *Throttling* [27] der Recheneinheiten aufgrund von Überhitzung der Bauteile durch die thermischen Emissionen der mechanischen Bauteile. Jedoch auch die mangelnde Qualität der verbauten Teile des GO1 lässt die teils durchaus ausreichend gute Implementierung der Funktionen auf Softwareseite an ihre Grenzen kommen.

Teile dieser Arbeit beschäftigen sich mit der effizienten Auslagerung mancher Funktionen, um die Ressourcen des Roboters sinnvoll einzusetzen und in möglichen Teilbereichen zu schonen. Die verbauten Teile des Roboters sowie die eingesetzte Software wird – wo relevant für die aufgezeigten Erweiterungen und Funktionen des GO1 – auf ihre Effizienz und Einsatzfähigkeit bewertet.

**Energetische und mechanische Komplexität**

[26] Siehe Kapitel 2.2.3

[27] Als Throttling bezeichnet man das Auslassen von Takten oder das Verringern der Taktfrequenz bei Prozessoren oder Grafikeinheiten, um die Temperatur zu senken und die Einheit somit vor Schäden zu schützen.

Die Nutzung der biologisch inspirierten vier Beine des Roboters mit hoher Flexibilität in der Rotation und des Bewegungsradius eines jeden Gelenkes bietet hinsichtlich der Vielseitigkeit der Fortbewegung und Reaktionsmöglichkeit auf schwer zu überwindendes Terrain einen großen Vorteil gegenüber einfacherer Lösungen in der Robotik, wie Rädern oder Ketten. Die präzise Steuerung der Motoren im gegenseitigen Zusammenspiel hingegen birgt mit ihrer Komplexität einen enormen Nachteil gegenüber anderer Fortbewegungsmöglichkeiten. Nicht nur ist der Bewegungsablauf deutlich aufwändiger zu implementieren, die Motoren sind in ihrer Vielzahl auch um einiges Energieineffizienter, als einfachere Bewegungsabläufe, wie die Rotation in einer Achse auf Rädern. Änderungen der Standardabläufe der Fortbewegung sind somit schwerer zu implementieren. Auch die Ausdauer des verbauten Batteriesystems leidet unter dem hohen Energieaufwand der Motoren und Recheneinheiten des Roboters. Ausdauernde oder vollkommen autonome Prozesse sind somit nur bedingt möglich, auch aufgrund der Notwendigkeit eines manuellen Batteriewechsels nach vollständiger Entladung.

Aufgrund des fehlenden oder mindestens mangelnden Schutzes des Roboters bei niedriger Ladung der eingesetzten Batterie beschäftigt sich diese Arbeit im Sinne dieser Herausforderung mit der Absicherung und Resilienz besonders des Batteriesystems gegenüber[28]. Im Sinne der sinnvollen Integration des GO1 in ein Hochschulökosystem werden die oben genannten Herausforderungen kritisch betrachtet und, wenn im Laufe der Arbeit sinnvoll, auch bewertet. Im folgenden Kapitel wird der Kern der Arbeit, der GO1, präzise analysiert und beschrieben, sodass für den weiteren Verlauf der Arbeit die Kenntnis des Roboters ausreichend ist, um den Schilderungen der Erweiterungen zu folgen.

[28] Siehe Kapitel 5

# 3 Roboterarchitektur und Systemkomponenten

Folgendes Kapitel beschreibt den physischen Aufbau des GO1 und die Konfiguration dessen im Werkszustand. Um die Komplexität des Systems besser verstehen zu können, wird der Aufbau in mehreren Schritten erklärt. Zuerst soll der äußere Aufbau dargestellt werden, danach der Aufbau der internen Systemelemente und der Sensorik und zuletzt soll die vereinfachte interne Netzwerkkommunikation der Recheneinheiten dargestellt werden.

## 3.1 Aufbau

Im Folgenden soll die Architektur des GO1 im Detail dargestellt werden. Hierfür werden einige Perspektiven des Roboters gezeigt, um die nach Einsatzzweck klassifizierten Bauteilgruppen zu erläutern und darzustellen.

### 3.1.1 Überblick

Durch die quadrupedale Fortbewegung und die enge Anlehnung an die Biologie können die Bezeichnungen der äußerlich erkennbaren Bauteile hergeleitet werden. Abbildung 4 zeigt die äußerlichen Merkmale im Überblick.

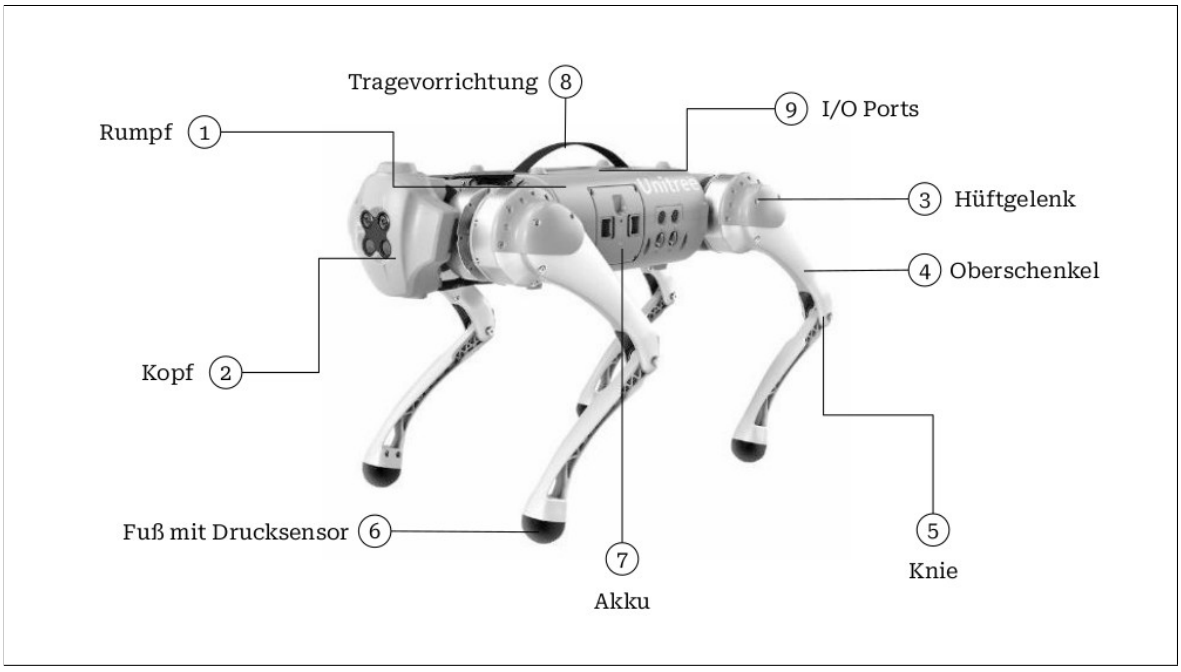


Abbildung 4: Überblick über den GO1

Die Grundlage des Roboters bildet der Körper – auf Abbildung 4 mit 1 gekennzeichnet. In diesem sind die meisten Komponenten des GO1 verbaut, unter Anderen die folgenden:

- Interne Hardwarekomponenten
- Intelligenter Akku
- Teile der Sensorik und Kameras
- Hüftgelenke und Motoren der Beine

Eine genauere Beschreibung der Einzelteile findet sich in den folgenden Unterkapiteln. An der Vorderseite des Körpers ist der Kopf ② des Roboters verbaut. In diesem sind beispielsweise ein *NVIDIA Jetson Nano*, eine Stereo-Kamera und Stereo Ultraschall Sensoren und weitere Bauteile wie Lautsprecher verbaut. An den vier äußeren Ecken des Körpers sind die Beine des Roboters verbaut. Innerhalb des Körpers sind die Motoren zur Steuerung der Hüftgelenke ③ integriert. Außerhalb der Hüftgelenke an der Oberseite der vier Oberschenkel ④ sind vier weitere Motoren zur vertikalen Steuerung der Beine verbaut. Parallel zu diesen Motoren sind im äußeren Teil des oberen Oberschenkels identische Motoren zur Steuerung der Knie ⑤ integriert, die die Gelenke jeweils durch steife Achsen und Seilzüge anwinkeln und strecken können. An den Enden der Beine sind jeweils Füße ⑥ verbaut, in denen Drucksensoren integriert sind.

Neben den äußerlich auffälligen Merkmalen ist auf der linken Seite des Körpers ein intelligenter Akku ⑦ verbaut. Auf der Oberseite des Körpers sind unterhalb der Tragevorrichtung ⑧ Schnittstellen ⑨ zur physischen Verbindung auf die integrierten Hardwarekomponenten verbaut.

### 3.1.2 Mechanische Komponenten

Der Anteil der in Abbildung 5 gezeigten mechanischen Elemente des GO1 hält sich in Grenzen. So sind am Körper selbst lediglich vier bewegliche Teile angebaut – die vier Servomotoren der Hüftgelenke ①. An den Hüftgelenken befestigt sind die einzigen weiteren beweglichen Bauteile des Roboters, die pro Bein jeweils zwei weiteren Motoren für die Neigung der Beine ② und der Kniegelenke ③.
Zum Strecken des Kniegelenks wird eine am äußeren Motor angebrachter Seilzug ④ verwendet, zum Anwinkeln des unteren Beines wir im Gegenzug eine starre Verbindung ⑤ an der Vorderseite des Kniegelenks genutzt.

#### Eigenschaften der Servomotoren

Die Servomotoren am Hüftgelenk – Abbildung 5, Bauteil ①, die Motoren an den Beininnenseiten ② und die Motoren an den Beinaußenseiten ③ sind des gleichen Models - *Unitree*

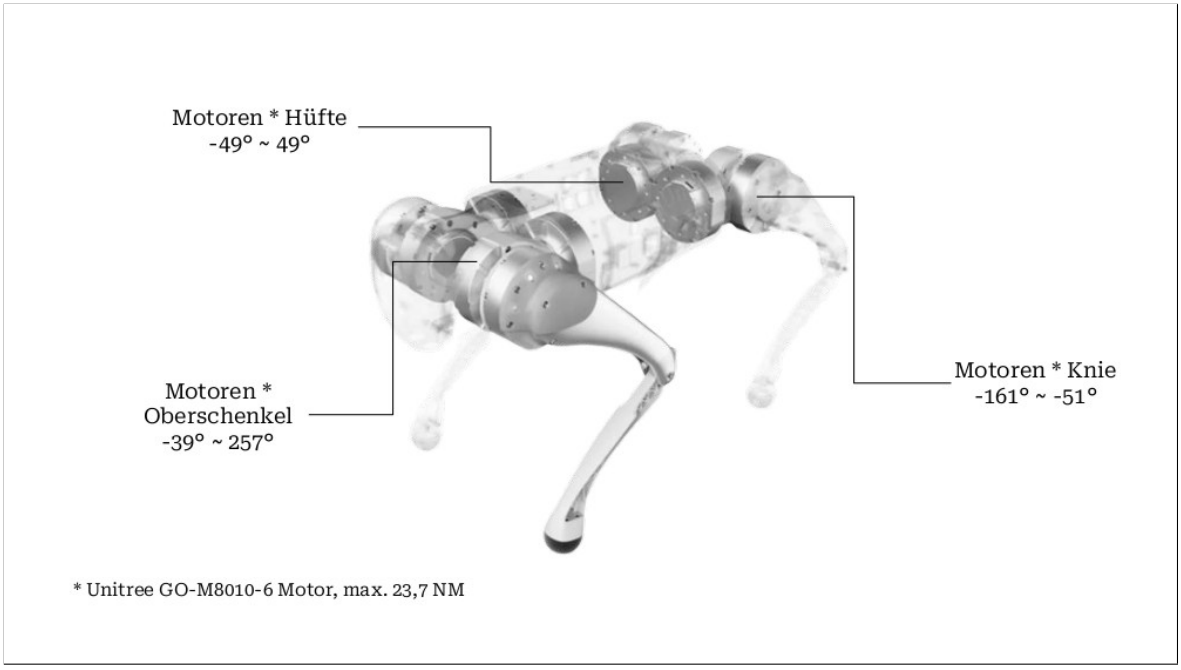


Abbildung 5: Mechanische Komponenten des GO1

*Robotics GO-M8010-6 Motor*[29]. Die Servomotoren haben ein maximales Drehmoment von 23.7NM (Newton/Meter) und können in 3 verschiedene Konfigurationen eingeteilt werden:

1. **Hüftmotoren**
   Bewegungsradius von $-49°$ bis $49°$
2. **Oberschenkelmotoren**
   Bewegungsradius von $-39°$ bis $257°$
3. **Kniemotoren**
   Bewegungsradius von $-161°$ bis $-51°$

Die Motoren sind ebenfalls mit Sensorik bestückt, welche den aktuellen Zustand des Bauteils erkennen und an die MCU (Main Control Unit) schicken können. Diese Funktionalität wird im Kapitel 3.1.3 näher beschrieben.

### 3.1.3 Sensorik

Zur intelligenten Nutzung des GO1 sind an vielen Stellen im Roboter einige Sensoren oder intelligente Hardware verbaut. Neben der dedizierten Sensorik zur Erkennung des Umfeldes ist ebenfalls einfachere Sensorik in einigen Bauteilen wie den mechanischen Bauteilen des

[29] Unitree Robotics. *GO-M8010-6 Motor - Unitree Robotics*. URL: `https://shop.unitree.com/products/go1-motor` (besucht am 07. 07. 2023).

Laufapparats integriert. Abbildung 6 zeigt die hierfür verbauten Sensoren und intelligenten Hardwarebausteine.

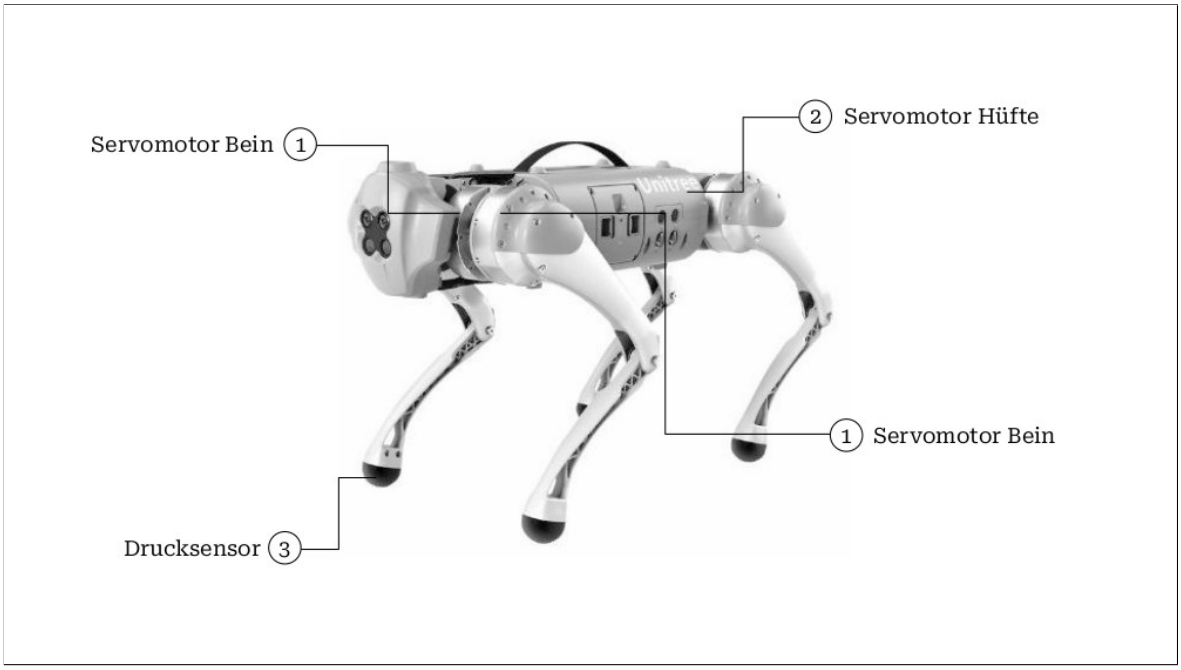


Abbildung 6: Sensorik des Laufaparats

Ähnlich der mechanischen Eigenschaften der zwölf Servomotoren des Typs *GO-M8010-6* – zwei pro Bein des Roboters ① und je ein Motor im Körper des Roboters ② – sind die sensorischen Funktionen dieser identisch. In Abbildung 12 in Kapitel 3.3.1 ist erkennbar, dass die zwölf Motoren des GO1 über eine *RS-485* Schnittstelle mit der MCU verbunden sind. Diese wertet die Informationen aus und steuert die einzelnen Motoren über dieselbe Schnittstelle an. Bei Rotationsbewegungen messen die Motoren, welche Kraft für die Ausführung benötigt wird und was die Differenz zur erwarteten Kraft ist. So können die Motoren allein bereits eine ausreichende Datenmenge zur Steuerung des Bewegungsapparats liefern.

Als Ergänzung zur intelligenten Messung der Motoren sind in den Enden aller vier Beine ③ Drucksensoren verbaut, welche die erzeugte Kraft auf die Unterseite der Beine messen und an die MCU senden. Die Kombination der Rotationsdaten und -kräfte sowie der gemessenen Kräfte an den Enden der Beine ermöglichen es dem GO1, die Motoren zum Ausgleich der Bewegungen anzusteuern und somit eine möglichst effiziente Fortbewegung zu ermöglichen.

Neben der integrierten Sensorik zur Steuerung des Laufapparats sind im GO1 Kameras, Ultraschallsensoren und Audiotechnik verbaut worden. Abbildung 7 zeigt hier die Verteilung der Kameras und Ultraschallsenoren.
Insgesamt sind im GO1 fünf Stereokamerasysteme verbaut, die durch ihre doppelte Bauart ebenso zur Tiefenerkennung fähig sind. Die Kameras sind folgendermaßen verteilt:

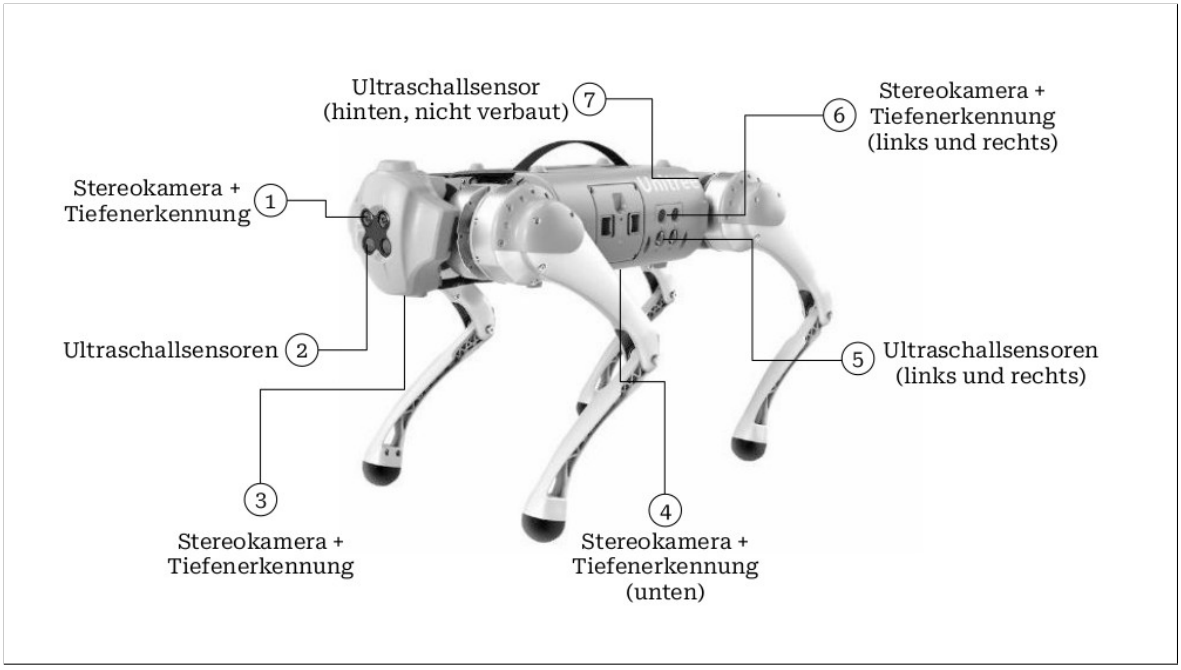


Abbildung 7: Darstellung der verbauten Kamera und Sensorik

- ① Kopfeinheit nach vorne gerichtet
- ③ Kopfeinheit nach unten gerichtet
- ④ Rumpf links und rechts
- ⑥ Rumpf nach unten gerichtet

Die Kameras ① und ③ am Kopf des GO1 werden durch den Jetson Nano im Kopf gesteuert, die beiden nach außen gerichteten Kameras ④ von einem weiteren Nano und die Kamera am Rumpf nach unten gerichtet ⑥ vom letzten der drei verbauten Jetsons, dem Xavier NX.[30] Mehr zur Steuerung und den Funktionen der Kameras und Recheneinheiten in Kapitel 3.2 und 4.2.6. Unter drei der fünf Kamerasysteme sind Ultraschallsensoren verbaut. Diese liegen am Kopf nach vorne ② und am Körper zu den Seiten gerichtet ⑤. Laut der Dokumentation der Ultraschallsensoren ist softwareseitig ein weiteres Ultraschallmodul am Rumpf nach hinten gerichtet ⑦ vorgesehen, welches aber hardwareseitig nie realisiert wurde[31]. Die Ultraschalleinheit im Kopf des Roboters wird vom Jetson Nano im Kopf des Roboters gesteuert während die beiden seitlichen Sensoren mit dem Raspberry Pi im Rumpf des GO1 verbunden sind.

[30] *Development and use of Go1 binocular fisheye camera*. Unitree Robotics.
[31] *Development and use of Go1 ultrasonic module*. Unitree Robotics.

### 3.1.4 Recheneinheiten und Schnittstellen

Die *Edu* Version des GO1 ist mit einiger integrierter Rechenleistung versehen worden. Innerhalb des Rumpfes und des Kopfes sind diverse Kleinplatinenrechner verbaut, um beispielsweise die Daten der oben beschriebenen Sensoren zu verwerten, den Bewegungsapparat manuell zu manipulieren und dem Roboter weitere Funktionen hinzuzufügen. Abbildung 8 zeigt die interne Zusammensetzung des GO1.

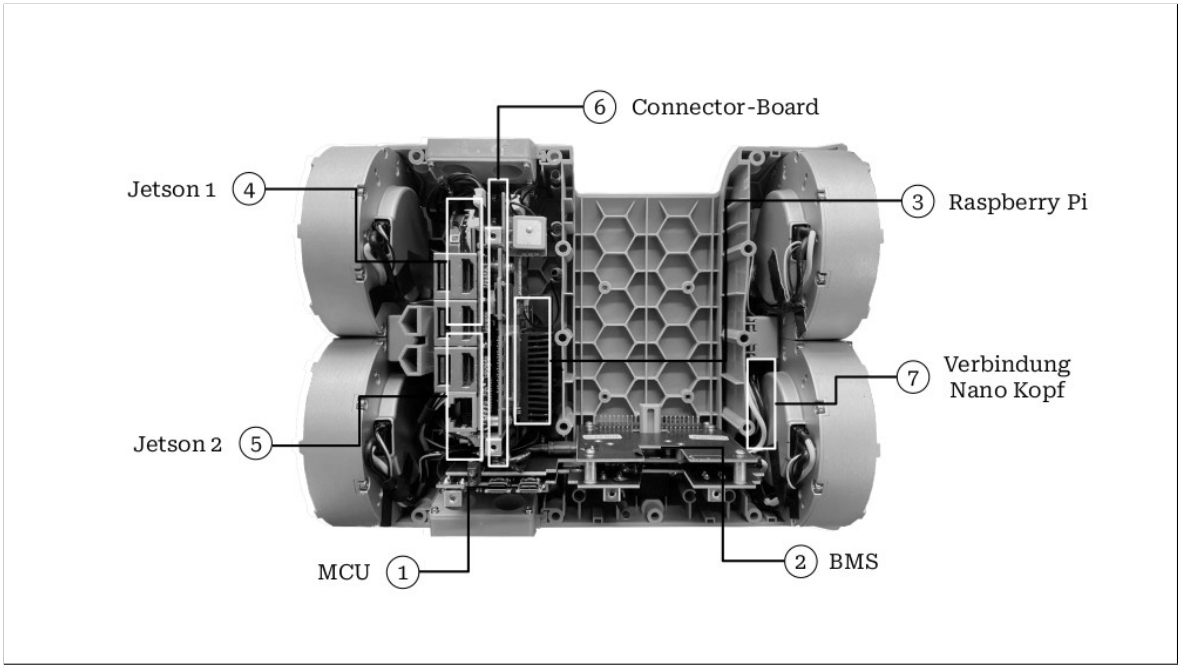


Abbildung 8: Blick auf die internen Komponenten

Herzstück des Roboters ist die MCU ①, welche auf der rechten Seite des Rumpfes verbaut ist. Diese steuert die Motoren und greift Daten des BMS (Batterie Management System) ② über den intelligenten Akku[32] ab. Mittig hinter dem eingebauten Akku liegt die Kernschnittstelle des GO1 mit den Nutzern, der verbaute *Raspberry Pi* ③. Dieser ist direkt über eine Erweiterungsplatine ⑥ für Schnittstellen und feste Verbindungen mit den beiden im Rumpf verbauten *NVIDIA Jetsons* ④+⑤ verbunden. Entlang des für den Akku reservierten Platzes im Rumpf verläuft die gekabelte Verbindung ⑦ zwischen den im Rumpf verbauten Komponenten und denen im Kopf des Roboters. Abbildung 9 zeigt die restlichen im Kopf verbauten Komponenten des Hundes.

Die zentrale Steuereinheit der Komponenten im Kopf des Roboters ist der verbaute *NVIDIA Jetson Nano* ①, der über gekabelte Verbindungen ② mit den im Rumpf verbauten Komponenten gekoppelt ist. Direkt am Nano ist ein Lautsprecher ③ verbaut, der im *Edu*-Modell frei

[32] Abbildung 10 ①

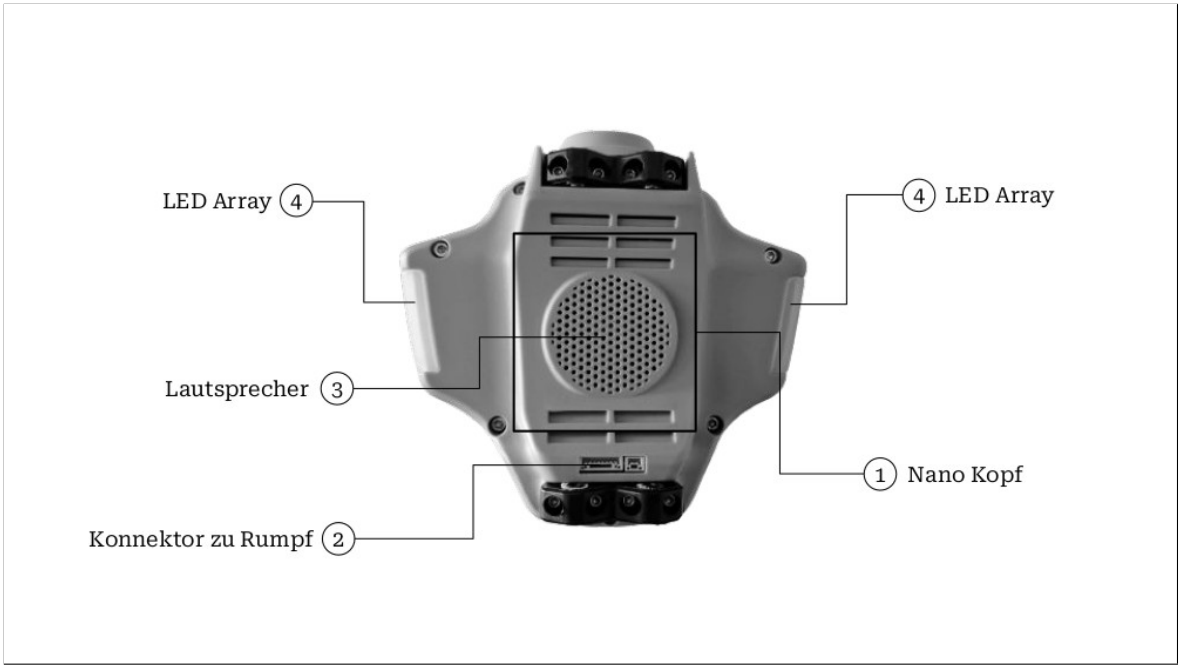


Abbildung 9: Ansicht des Kopfes von hinten

nutzbar ist[33]. Ebenso direkt an den Nano angeschlossen sind links und rechts am Kopf zwei nach hinten und zur Seite orientierte LED (Leuchtdioden)-Reihen. Auch diese sind frei programmierbar[34]. Eine detailliertere Beschreibung der Recheneinheiten und derer Funktionen folgt in Kapitel 3.2.

Auf der Oberseite des GO1 sind einige Schnittstellen verbaut, die die Arbeit an den internen Komponenten und Recheneinheiten des Roboters erleichtern sollen. Abbildung 10 zeigt die Oberseite und die verbauten Schnittstellen. Alternativ zur Stromversorgung über den verbauten Akku ① auf der linken Außenseite des Roboters ist eine *XT-30*-Steckverbindung ② auf der Oberseite verbaut. Diese kann entweder den GO1 anstelle des Akkus betreiben oder externe Erweiterungen durch den Akku mit Strom versorgen. Genaueres hierzu in Kapitel 4.1. Direkt oberhalb dieser Verbindung sind zwei USB (Universal Serial Bus) *Typ C* Ports ③, welche als Schnittstellen zur MCU genutzt werden können. Erweiterungen, wie beispielsweise GPS (Global Positioning System)-Module können über eine serielle Schnittstelle ④ angeschlossen werden.

Zur gekabelten Verbindung eines externen Rechners mit dem intern verbauten Switch und somit den Recheneinheiten[35] ist eine *RJ-45*-Ethernetbuchse ⑤ verbaut. Als Alternative zum kabellosen Netzwerk kann für eine Verbindung mit dem Internet oder einem Mobilfunknetz-

[33] Siehe Kapitel 4.2.4
[34] Kapitel 4.2.5
[35] Siehe Kapitel 3.3

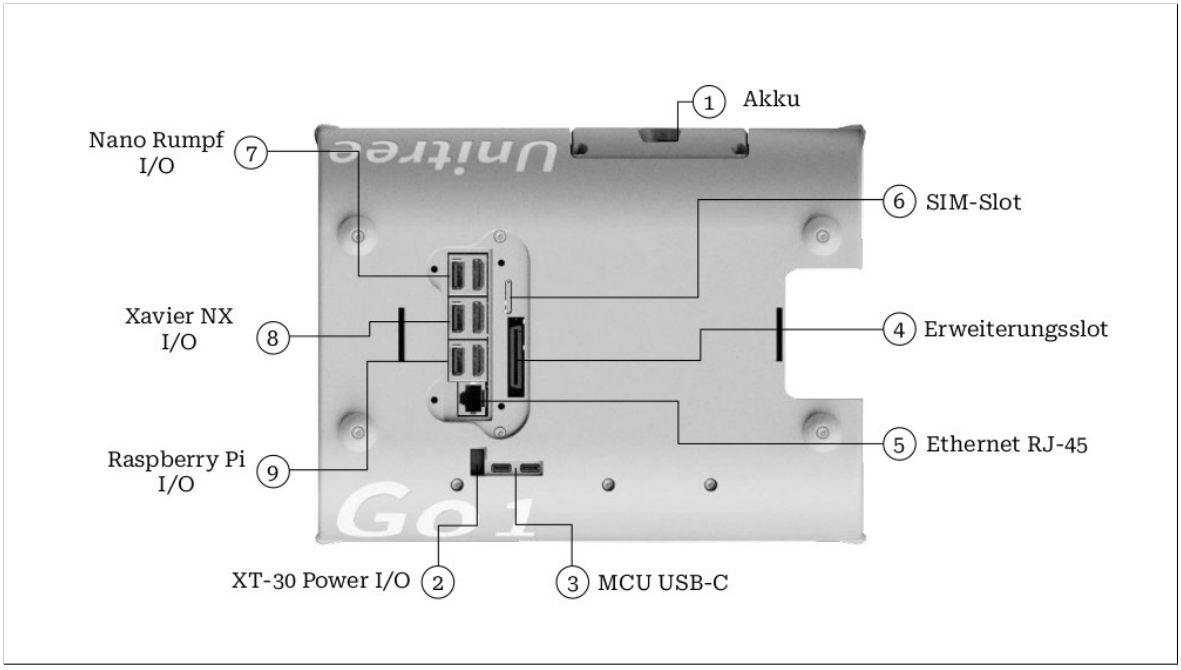


Abbildung 10: Vogelperspektive mit Hardware

werk der eingebaute Steckplatz für *4G/ LTE (Long Term Evolution)*-Karten ⑥ verwendet werden. Dieser ist direkt mit dem verbauten Raspberry Pi verbunden[36].

Die restlichen Verbindungen auf der Rückseite des Rumpfes sind drei Paare mit je einem HDMI (High Definition Multimedia Interface) und einem USB-A Port Für die direkte Verbindung zu den Kleinplatinenrechnern durch Entwickler. Die Verteilung ist wie folgt:

- ⑦ NVIDIA Jetson Nano (Rumpf)
- ⑧ NVIDIA Jetson Xavier NX
- ⑨ Raspberry Pi

Auf den Nano im Kopf des Roboters ist über die Ports keine direkte Verbindung möglich.

## 3.2 Hardware Architektur

Im folgenden Kapitel wird der Aufbau des intern verbauten Hardware-Systems und die Funktion der einzelnen Bauteile dessen dargestellt. Hierfür wird vorerst ein kurzer Überblick über die Komponenten und ihren Zusammenhang geschaffen, was sich in Teilen mit der Ausführung im vorigen Kapitel 3.1.4 überschneidet. Mit dem geschaffenen Überblick werden die Einzelteile des Systems genauer betrachtet und dokumentiert. Dies soll über die

[36] Siehe Kapitel 5.1

einfache Dokumentation des Herstellers deutlich hinaus gehen. Abschließend wird die interne Kommunikation der Bauteile in sich und mit externen Komponenten beschrieben.

### 3.2.1 Überblick

Als Einstieg in den Überblick soll veranschaulichend Abbildung 11 dienen. Sie zeigt die einzelnen Komponenten und ihre Verbindungen untereinander und zu den erweiterten Systemen wie dem BMS oder der Sensorik.

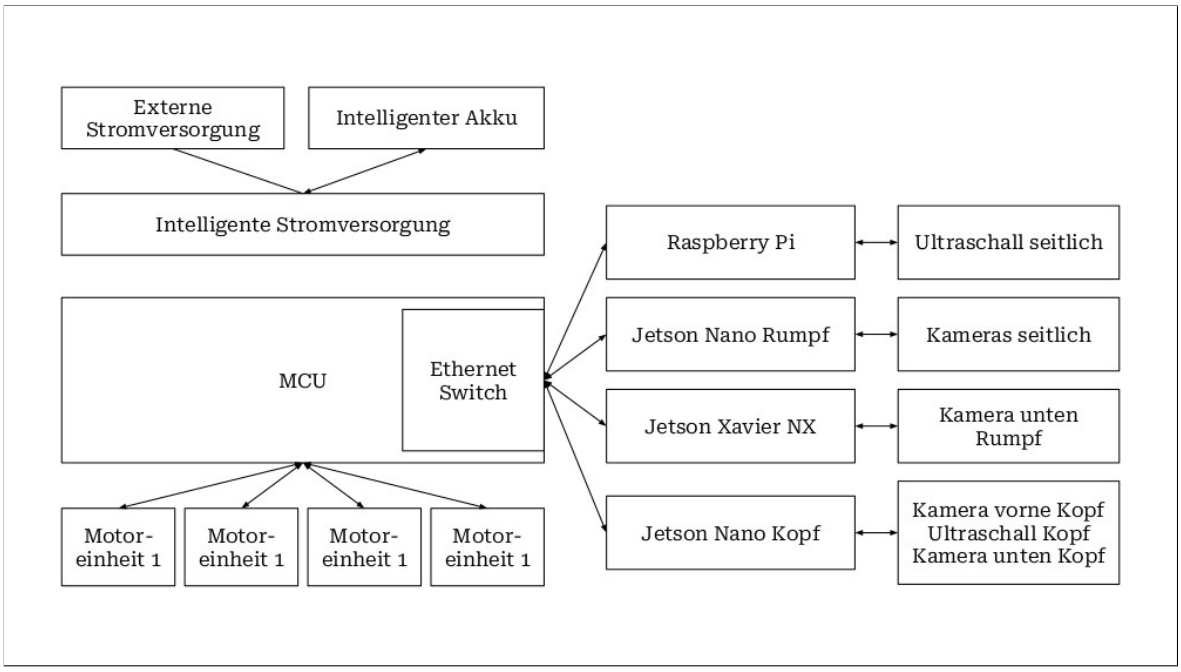


Abbildung 11: Überblick über die interne Architektur des Go1

Als Grundlage für sowohl die interne Hardware als auch die mechanischen Komponenten dienen die MCU und das BMS. Beide werden vom Hersteller nicht für den Zugriff freigeschaltet und können lediglich indirekt verwendet werden. So können beispielsweise die Daten der Motoren und deren Steuerung aktuell nur über Bibliotheken gelesen und manipuliert werden. Auch die Daten des BMS sind nur lesend verfügbar. Zentral zu allen Komponenten steht ein Switch, welcher diese über ein Netzwerk verbindet. Direkt daran angeschlossen sind alle drei *NVIDIA Jetson* Einheiten – zwei Nanos und ein Xavier NX, der *Raspberry Pi*, die *MCU* und der nach außen verfügbar gemachte *RJ-45 Port*. Der Raspberry Pi kann als zentraler Baustein für alle Entwickler am GO1 bezeichnet werden. Bis auf dedizierte Auswertungen oder Zugriffe auf die Kameramodule werden die meisten Prozesse zumindest auf dem Pi verwaltet. Die NVIDIA Einheiten hingegen verarbeiten die ihnen zugeordneten Kameramodule, mit der Ausnahme des Nanos im Kopf des Roboters, der zudem auch die Sensordaten des nach vorne gerichteten Ultraschallsensors abgreift und verfügbar macht. Der NVIDIA Jetson

Xavier NX ist zudem die rechen-stärkste Einheit mit Blick auf Prozessor und Grafikeinheiten und kann deshalb im ML (Machine Learning) Bereich und in der Videoauswertung verwendet werden. Folgende Übersicht zeigt die Verteilung der zugreifbaren Recheneinheiten zu den zu verwaltenden Bausteinen.

| **Raspberry Pi** | **Nano 1 (Kopf)** | **Nano 2 (Rumpf)** | **Xavier NX** |
|---|---|---|---|
| Wifi Modul<br>Webhosting<br>App-Verbindung<br>Monitoring<br>Bibliotheken<br>Ultraschall seitlich | LED-Steuerung<br>Audio-Ausgabe<br>Ultraschall frontal<br>Videoauswertung<br>Kopf vorne/unten | Videoauswertung<br>links + rechts | Videoauswertung<br>Rumpf unten<br>ML Prozesse |

Das nächste Kapitel geht auf Grundlage des Überblicks genauer auf die einzelnen Bausteine der internen Hardware des GO1 ein.

### 3.2.2 Kernelemente

Als Kernelemente des GO1 werden der verbaute *Raspberry Pi*, die zwei verbauten *NVIDIA Jetson Nanos* und der *NVIDIA Jetson Xavier NX* bezeichnet. Grundsätzlich ist die MCU ebenfalls als Kernelement zu bezeichnen, sie ist jedoch nicht für den Zugriff durch den Entwickler freigeschaltet und wird deshalb nicht weiter betrachtet. Zur genaueren Inspektion der Komponenten kann ein externer Rechner per *Ethernet* an den in Kapitel 3.1.4 *RJ-45*-Port angeschlossen werden. Diesem Rechner muss dann eine statische IP (Internet Protocoll)-Adresse im Netz `192.168.123.0/24` vergeben werden. Da die IP-Adresse noch nicht vergeben sein darf, wurde zur Analyse im Rahmen dieser Arbeit die Adresse `192.168.123.51` verwendet.

**Raspberry Pi**

Laut Hersteller-Dokumentation ist im Roboter ein *Raspberry Pi* 4 verbaut. Anhand der Dokumentation erkennt man lediglich die IP-Adresse des Pi – `192.168.123.161`, jedoch keine Informationen zu den Eigenschaften dessen. Zur Prüfung der Eigenschaften des Pis kann sich mit dem Roboter per Ethernet verbunden werden. Über den Standard-Nutzer kann sich laut Dokumentation per SSH (Secure Shell) auf den Roboter verbunden werden.

```
# user: pi, password: 123, root-password:123
ssh pi@192.168.123.161
```

Als Erstes soll das genaue Modell des Raspberry Pi erkannt werden:

```
pi@raspberrypi:~ $ grep Model /proc/cpuinfo
Model           : Raspberry Pi Compute Module 4 Rev 1.0
```

Die Prüfung der verbauten Variante des Compute Model 4 lässt dich durch folgendes Kommando durchführen:

```
pi@raspberrypi:~ $ grep MemTotal /proc/meminfo
MemTotal:        1894664 kB
```

Eine kurze Prüfung der Herstellerwebsite zeigt uns, dass ein `Broadcom BCM2711 quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz` in der Variation mit 2 GB (Gigabyte) Arbeitsspeicher verbaut ist. Die Boot-Partition und der initiale Festplattenspeicher werden bei diversen Kleinplatinenrechner oftmals über eine SD (Secure Digital)-Karte realisiert. Um das auf dem Raspberry Pi zu prüfen kann die Belegung des Dateisystems ausgegeben werden. Die temporären Dateisysteme werden hierbei ausgeschlossen.

```
pi@raspberrypi:~ $ df -HTx tmpfs -x devtmpfs
Filesystem     Type  Size  Used Avail Use% Mounted on
/dev/root      ext4   32G   18G   13G  59% /
/dev/mmcblk0p1 vfat  265M   69M  196M  26% /boot
```

Zu erkennen ist, dass tatsächlich eine SD-Karte als Boot-Partition unter `/dev/mmcblk0p1` eingebunden wurde. Zudem lässt sich die Gesamtgröße des Dateisystems ablesen – 32 GB. Zur Prüfung des verbauten Betriebssystems und des Linux Kernels kann folgendes Kommando verwendet werden:

```
pi@raspberrypi:~ $ grep PRETTY_NAME /etc/os-release
PRETTY_NAME="Debian GNU/Linux 10 (buster)"
pi@raspberrypi:~ $ uname -r
5.4.81-rt45-v8+
```

Zusammenfassend lassen sich die Kerndaten des Pis wie in Tabelle 1 dargestellt.

| | |
|---|---|
| Modell | Raspberry Pi Compute Module 4 Rev 1.0 |
| SoC | Broadcom BCM2711 quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz |
| RAM | 1 894 664 kB (1.8 GB) Arbeitsspeicher |
| Speicher | 32 GB Festplattenspeicher über eine SD-Karte |
| OS | Debian 10 (Buster) |
| Kernel | Linux Kernel 5.4.81-rt45-v8+ |

Tabelle 1: Kenndaten des Raspberry Pi

Wirft man einen Blick auf die zusätzlich zu den im Raspberry Pi integrierten angeschlossenen Komponenten, so erkennt man die Funktion des Pi als Schnittstelle zwischen Entwickler und Roboter gut. Angeschlossene Geräte lassen sich großenteils über die Ausgabe der per USB verbundenen Geräte mit dem Befehl `lsusb` prüfen.

```
pi@raspberrypi:~ $ lsusb
Bus 001 Device 004: ID 0bda:c812 Realtek Semiconductor Corp.
Bus 001 Device 003: ID 2c7c:0125 Quectel Wireless Solutions Co., Ltd.
   ↪  EC25 LTE modem
[...]
```

Nennenswert sind in der Ausgabe besonders der *Realtek* USB-WiFi Adapter und das *Quectel*-Modem zum Einstecken der 4G/ LTE SIM (Subscriber Identity Module)-Karte. Die genaue Funktion der USB Geräte lässt sich oftmals durch eine Suche der Geräteidentifikation links neben dem vollen Namen des Gerätes auf der Herstellerwebsite prüfen. Beide Geräte erlauben es dem Pi, sich neben der auf der Platine verbauten Schnittstellen mit dem Internet oder einem lokalen Netzwerk zu verbinden.

Ein Blick auf die offizielle Dokumentation der Ultraschallsensoren lässt hier erkennen, dass die beiden Sensoren links und rechts am Rumpf des Roboters über den Serial-Port `ttyAMA0` verbunden sind.

```
pi@raspberrypi:~ $ dmesg | grep ttyAMA0
[    1.251673] fe201000.serial: ttyAMA0 at MMIO 0xfe201000 (irq = 14,
   ↪  base_baud = 0) is a PL011 rev2
```

Da die Bibliotheken des Herstellers zur Nutzung der Ultraschallsensoren vorkompiliert und nicht dokumentiert ist konnten über die Verbindungsart und die Nutzung der Sensoren ohne die Bibliothek keine weiteren Informationen gefunden werden. Dies gilt auch für die Sensoren im Kopf des GO1.

### NVIDIA Jetson Nano Kopf

Folgt man der Dokumentation der Hersteller, so erreicht man die Recheneinheit im Kopf des Roboters unter der IP Adresse `192.168.123.13`. Verbinden lässt sich der Rechner über den Nutzer `unitree` und dem Passwort 123.

```
# user: unitree, password: 123, root disabled
ssh unitree@192.168.123.<nano-ip(13|14|15)>
```

Laut Hersteller ist auf allen drei NVIDIA Chips das Betriebssystem Ubuntu installiert, welches auf Debian basiert, aber einige nützliche Funktionen über die Basis von Debian

hinaus mitbringt. So auch den Befehl `lshw`, über den sich eine Zusammenfassung der auf dem System verwendeten Hardware ausgeben lässt.

```
unitree@unitree-desktop:~$ sudo lshw -short
[...] Class       Description
[...] ========================
[...] system      NVIDIA Jetson Nano Developer Kit
[...] memory      3962MiB System memory
[...] bridge      NVIDIA Corporation
[...] multimedia  USB2.0 Camera RGB
[...] multimedia  USB2.0 Camera RGB
[...] generic     CP2102N USB to UART Bridge Controller
[...] multimedia  USB Audio Device
```

Erkennbar ist über die gekürzte Ausgabe, dass im Kopf des GO1 ein *NVIDIA Jetson Nano* mit 4 GB Arbeitsspeicher verbaut ist. Zudem sind per USB vier externe Geräte angeschlossen, ein Lautsprecher im Rücken des Kopfes, ein Bridge-Controller zur Steuerung der beiden LED-Bänder und zwei Kameras. Die nach vorne gerichtete Kamera ist unter `/dev/video1` gemountet, die im Kopf nach unten gerichtete Kamera unter `/dev/video0`. Die Dokumentation der Ultraschallsensoren zeigt ebenfalls, dass sich am Serial-Port `ttyTHS1` der Ultraschall-Sensor am Kopf des Roboters befindet.

```
unitree@unitree-desktop:~$ dmesg| grep ttyTHS1
[    1.099918] 70006040.serial: ttyTHS1 at MMIO 0x70006040 (irq = 64,
   ↪  base_baud = 0) is a TEGRA_UART
```

Unter den Mountpoints der Kameras können diese ausgelesen und als Quelle verwendet werden. Ein Blick auf die Dateisysteme zeigt im Gegensatz zum selben Befehl auf dem Raspberry Pi lediglich die `root`-Partition, ein weiteres Inspizieren zeigt dann jedoch ebenfalls die `boot`-Partition auf der SD-Karte.

```
unitree@unitree-desktop:~$ df -Hx tmpfs -x devtmpfs
Filesystem      Size  Used Avail Use% Mount
/dev/mmcblk0p1   15G   12G  2.5G  83% /

unitree@unitree-desktop:~$ sudo lsblk
NAME          FSTYPE   SIZE MOUNTPOINT
mmcblk0               14.7G
-> mmcblk0p1  ext4      14G /
mmcblk0boot0             4M
mmcblk0boot1             4M
```

Die Prozessorvariante in `/proc/cpuinfo` gibt `ARMv8 Processor rev 1 (v8l)` aus. Ein Blick auf die Herstellerwebsite zeigt, dass mit der Prozessorbezeichnung ein `Quad-Core ARM Cortex-A57 MPCore` Prozessor verbaut ist[37]. Da die NVIDIA Jetson Reihe auf den Einsatz in der Robotik und im ML Bereich optimiert sind, sind neben der CPU (Central Processing Unit) starke Grafikeinheiten verbaut. Laut Hersteller ist eine `NVIDIA Maxwell-GPU` mit 128 Cores verbaut. Die genaue Version des Betriebssystems lässt sich analog zum Vorgehen im Raspberry Pi ermitteln:

```
unitree@unitree-desktop:~$ grep PRETTY_NAME /etc/os-release
PRETTY_NAME="Ubuntu 18.04.5 LTS"
unitree@unitree-desktop:~$ uname -r
4.9.201-tegra
```

Zusammenfassen lassen sich die Kerndaten des NVIDIA Jetson Nanos im Kopf des Roboters wie in Tabelle 2 dargestellt:

| | |
|---|---|
| Modell | NVIDIA Jetson Nano Developer Kit |
| GPU | NVIDIA Maxwell-GPU mit 128 Cores |
| Prozessor | Quad-Core ARM Cortex-A57 MPCore |
| RAM | 3962 MiB (4 GB) Arbeitsspeicher |
| Speicher | 16 GB Festplattenspeicher über eine SD-Karte |
| OS | Ubuntu 18.04.5 LTS |
| Kernel | 4.9.201-tegra |

Tabelle 2: Kenndaten des NVIDIA Jetson Nano im Kopf des Roboters

**NVIDIA Jetson Nano Rumpf**

Der Nano im Rumpf des Roboters is dem Nano im Kopf des Roboters gegenüber bis auf die angeschlossenen Geräte und seiner IP-Adresse `192.168.123.14` identisch. Die Ausgaben für das Betriebssystem in `/etc/os-release`, der Kernel-Version aus `uname -r` und der Informationen zur SD-Karte aus `df -Hx tmpfs -xdevtmpfs` stimmen bei beiden Geräten überein. Die Unterschiede in der verbundenen Hardware gehen aus folgender Ausgabe hervor:

```
unitree@unitree-desktop:~$ sudo lshw -short
[...] Class       Description
[...] ========================
[...] system      NVIDIA Jetson Nano Developer Kit
[...] memory      3964MiB System memory
[...] bridge      NVIDIA Corporation
```

[37] NVIDIA Corporation. *Entwicklerkit und -module für eingebettete Systeme*. 2023. URL: urhttps://www.nvidia.com/de-de/autonomous-machines/embedded-systems/.

```
[...] multimedia  USB2.0 Camera RGB
[...] multimedia  USB2.0 Camera RGB
```

Ähnlich zum Nano im Kopf des GO1 sind zwei Kameras verbaut, eine Kamera links im Rumpf von hinten betrachtet unter dem Mounting-Point `/dev/video1` und eine Kamera rechts im Rumpf unter `/dev/video0`. Als Übersicht zu den Kerndaten des Nanos im Rumpf des Roboters kann Tabelle 2 verwendet werden.

**NVIDIA Jetson Xavier NX**

Die zentrale Recheneinheit mit der höchsten Leistung ist der im Rumpf verbaute *NVIDIA Jetson Xavier NX* mit der internen IP-Adresse `192.168.123.15`. Auf dem NX ist, wie auf beiden Nanos, das Betriebssystem Ubuntu installiert. Auch die Kernel-Version ist identisch zu beiden Nanos.

```
unitree@nx:~$ grep PRETTY_NAME /etc/os-release
PRETTY_NAME="Ubuntu 18.04.5 LTS"
unitree@nx:~$ uname -r
4.9.201-tegra
```

Im Wesentlichen unterscheidet der NX sich in seiner Plattform, der Prozessoreinheit und dem verbauten Arbeitsspeicher von den beiden Nanos. Folgende Übersicht gibt die Hardwareausstattung aus:

```
unitree@nx:~$ sudo lshw -short
[sudo] password for unitree:
[...] Class       Description
[...] ========================
[...] system      NVIDIA Jetson Xavier NX Developer Kit
[...] memory      7773MiB System memory
[...] bridge      NVIDIA Corporation
[...] multimedia  USB2.0 Camera RGB
```

Laut Hersteller ist im NX ein `6-Core NVIDIA Carmel ARM v8.2 64-bit-CPU` verbaut[38]. Es kann festgehalten werden, dass der NX deutlich fähiger ist, als die beiden Nanos in Kopf und Rumpf. Zusätzlich zum besseren Prozessor sind im NX 8 GB Arbeitsspeicher verbaut. Die angeschlossene Kamera ist die im Rumpf nach unten gerichtete Kamera. Ein Blick auf die Festplattenkapazitäten des NX stellt auch klar, warum Unitree diese Einheit als Kernstück der Rechenleistung des GO1 bewirbt.

```
unitree@nx:~$ df -Hx tmpfs -x devtmpfs
```

[38] Corporation, s. Anm. 37.

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/nvme0n1p1  118G   22G   90G  20% /
/dev/mmcblk0p1   15G   84M   14G   1% /media/unitree/cd8bfc0a-0f39-4
   ↪ efa-b376-116833b08f45
```

Die deutlich höhere Speicherkapazität durch das Anschließen einer SSD (Solid State Drive) zusätzlich zur für den Bootvorgang verwendeten SD-Karte ermöglicht dem NX das Auswerten größerer Datenmengen als auf den beiden Nanos mit nur 16 GB Speicherkapazität auf deren SD-Karten. Wie auf den Nanos ist auch auf dem NX die Grafikeinheit deutlich wichtiger als die Recheneinheit. Laut Hersteller ist eine `NVIDIA Volta-GPU` mit 384 Cores und 48 Tensor Cores verbaut. Auch hier ist der NX deutlich besser ausgestattet als die beiden Nanos. Tabelle 3 fasst die Eigenschaften des NVIDIA Jetson Xavier NX kurz zusammen.

| | |
|---|---|
| Modell | NVIDIA Jetson Xavier NX |
| GPU | NVIDIA Volta-GPU mit 384 Cores und 48 Tensor Cores |
| Prozessor | 6-Core NVIDIA Carmel ARM v8.2 64-bit-CPU |
| RAM | 7773 MiB (8 GB) Arbeitsspeicher |
| Speicher | 16 GB Festplattenspeicher über eine SD-Karte<br>120 GB SSD Speicher |
| OS | Ubuntu 18.04.5 LTS |
| Kernel | 4.9.201-tegra |

Tabelle 3: Kenndaten des NVIDIA Jetson Xavier NX

Anzumerken ist, dass in der Hersteller-Dokumentation des Xavier NX eine Möglichkeit gelistet ist, den NX in verschiedenen Leistungsmodi zu starten. Je nach Eingangsleistung und Kühlkapazität haben die CPU und GPU (Graphics Processing Unit) die Möglichkeit, ihre Taktfrequenz zu erhöhen. Diese Funktion wurde jedoch nicht getestet.

## 3.3 Netzwerk

Aufgrund der vielen Komponenten, Funktionen und den möglichen Erweiterungen des GO1 ist eine robuste interne Kommunikation nötig. Die interne Kommunikationsstruktur des Roboters baut großenteils auf Netzwerkstandards wie *Ethernet* und *Wi-Fi* auf, setzt besonders in der Konnektivität mit externen Komponenten jedoch zusätzlich auf weitere Standards wie *Bluetooth* und *WWAN (Wireless Wide Area Network)*.

Das folgende Kapitel erläutert die vorhandene Kommunikation der internen und externen Komponenten des GO1 und analysiert diese auf ihre Stärken und Schwächen. Zudem soll die Methodik der Analyse des Netzwerks und mögliche Problemfeststellungen und -behebungen festgehalten werden.

### 3.3.1 Überblick

Abbildung 12 gilt als Referenz für die folgenden Ausführungen. Zentrale Einheit der Kommunikation sind der verbaute Ethernet Switch und der intern verbaute *Raspberry Pi*. Wie auf Abbildung 12 zu erkennen ist, sind alle fünf Recheneinheiten - der Raspberry Pi, die MCU, die beiden NVIDIA Jetson Nanos und der Jetson Xavier NX - per *Ethernet* mit dem Switch verbunden. Auch der extern zugängliche Ethernet-Port auf dem Rücken des Roboters ist mit dem Switch verbunden.

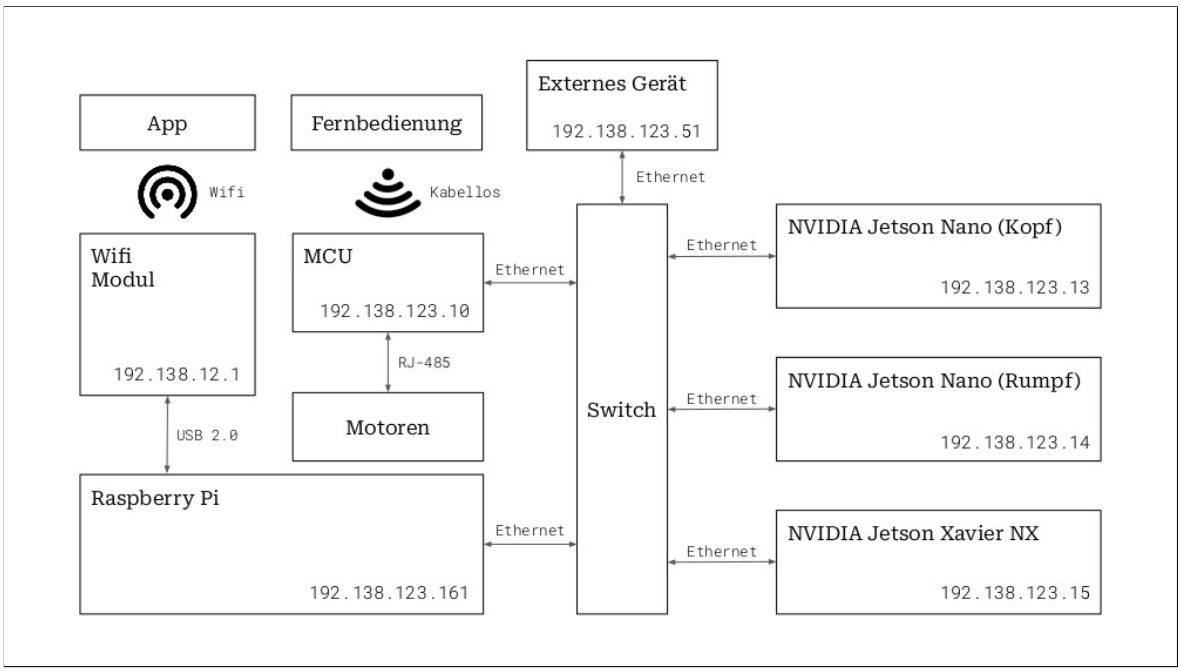


Abbildung 12: Überblick über Netzwerkkonfiguration

Alle geswitchten Komponenten des Netzwerks sind im `192.168.123.0/24`-Netzwerk registriert. Dabei ist die Verteilung der IP-Adressen folgendermaßen vorkonfiguriert:

- **MCU:** `192.168.123.10`
- **Raspberry Pi:** `192.168.123.161`
- **NVIDIA Jetson Nanos:**
  1. Kopf: `192.168.123.13`
  2. Seiten: `192.168.123.14`
- **NVIDIA Jetson Xavier NX:** `192.168.123.15`

Dem Endgerät, das am externen Ethernet-Port an der Oberseite des Roboters angesteckt werden kann, muss eine statische IP-Adresse im Bereich `192.168.123.0/24` vergeben werden, die nicht bereits von einem der oben genannten Geräten verwendet wird.

Der Raspberry Pi hat zusätzlich zu seiner physischen Verbindung zum Switch und der `192.168.123.161`-IP-Addresse noch ein WLAN (Wireless Local Area Network) Modul verbaut, mit welchem er das Netz `192.168.12.0/24` publiziert. Dieses Netz wird ab Werk für die Verbindung der App mit dem System benötigt. Des Weiteren kann dieses Netz genutzt werden, um eine kabellose Verbindung mit dem Gesamtsystem des Roboters herzustellen.

### 3.3.2 Gekabelte Verbindung

Wie bereits in Kapitel 3.1.4 angerissen, besitzt der GO1 auf der Oberseite des Rumpfes eine RJ-45 Schnittstelle zur physischen Verbindung mit den Recheneinheiten. Nachdem eine gekabelte Verbindung hergestellt wurde, muss auf der verbundenen Einheit eine statische IP Adresse konfiguriert werden, die nicht bereits von einer der verbauten Komponenten belegt ist. Im Rahmen dieser Arbeit wird stets die IP Adresse `192.168.123.51` angegeben. Vergeben sind lediglich die Adressen, die im Überblick bereits gelistet wurden. Ist die statische Adresse konfiguriert, können nun alle Systeme, bei denen SSH aktiviert ist, erreicht werden. Zu beachten ist lediglich, dass das Netzwerk nur ein geswitchtes Netzwerk ist, kein geroutetes. Die interne Kommunikation muss also über Routen auf den einzelnen Recheneinheiten konfiguriert werden. Ein Beispiel hierfür ist im folgenden Paragrafen zum Thema *kabellose Verbindung* aufgeführt.

### 3.3.3 Kabellose Verbindung

Zu kabellosen Verbindung mit dem Netzwerk des GO1 ist im Gegensatz zur gekabelten Verbindung keine statische IP Adresskonfiguration nötig. Es muss sich lediglich mit dem WLAN Access Point verbunden werden, den der Raspberry Pi nach Systemstart aufspannt. Die genaue Konfiguration dessen ist in Kapitel 4.2.3 dokumentiert.

Nach Start des Roboters wird ein Access Point mit der Seriennummer des verwendeten GO1 als SSID (Service Set Identifier) aufgespannt. Das Standardpasswort zur Verbindung ist `00000000`. Die IP Konfiguration des Hosts wird dann über DHCP (Dynamic Host Configuration Protocol) geregelt. Der Access Point hat die Addresse `192.168.12.1`, weshalb dem Host auch eine Adresse im Netzwerk `192.168.12.0/24` vergeben wird.

Um die interne Kommunikation des Pis mit dem Rest des Netzwerks besser nachvollziehen zu können, können die Forwarding-Regeln und die IP Routen des Systems ausgegeben werden.

```
pi@raspberrypi:~ $ cat /proc/sys/net/ipv4/ip_forward
1 # IP-Forwarding aktiviert
pi@raspberrypi:~ $ cat /etc/sysctl.conf | grep ip_forward
```

```
net.ipv4.ip_forward=1 # Dauerhafte Aktivierung des Forwarding
pi@raspberrypi:~ $ sudo iptables -L
[...]
Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere
ACCEPT     all  --  anywhere             anywhere
[...]
```

Zu erkennen ist, dass das System alle Forwardingversuche zulässt. Des Weiteren müssen für Kommunikation des `192.168.12.0/24` Netzes mit den Hosts im `192.168.123.0/24` Netz die Routen auf dem Pi geprüft werden.

```
pi@raspberrypi:~ $ ip route | grep default
[...]
default via 192.168.123.1 dev eth0 src 192.168.123.161 metric 202
default via 192.168.12.1 dev wlan1 src 192.168.12.1 metric 305
[...]
```

Zu erkennen ist, dass die Standardroute aller Anfragen an das Gateway im Netz `..12.0/24` über das Interface `wlan1` geleitet werden, worüber sie dann an den verbundenen Host kommen. Umgekehrt werden alle Nachrichten des Host an das Netz `..123.0/24` über das Ethernet Interface `eth0` geleitet, dass als Gateway fungiert und die Pakete an die Empfänger direkt weiterleitet, da diese über den Switch ohne weitere Gateways erreichbar sind.

Auf den verbundenen NVIDIA Jetsons ist hingegen nur eine Standardroute definiert.

```
unitree@unitree-desktop:~$ ip route | grep default
default via 192.168.123.161 dev eth0
default via 192.168.123.1 dev eth0 onlink
```

Hier fungiert jeweils das Ethernet Interface `eth0` als Gateway und leitet alle unbekannten Adressen, beispielsweise die des Netzes `..12.0/24`, über dieses Interface an die Adresse `192.168.123.161`, was die Adresse des Ethernet Interfaces des Raspberry Pis ist.

### 3.3.4 Weitere Netzwerk Funktionen

Innerhalb des Raspberry Pis sind neben den Interfaces `eth0` und `wlan1` noch weitere Netzkarten verbaut. Diese können beispielsweise genutzt werden, um den Pi mit einem WLAN Access Point zu verbinden oder ihn in ein Mobilfunknetz einzuwählen. Diese Funktionen werden genauer in den Kapiteln 5.1 und 5.1 erläutert.

# 4 Analyse des Roboters

Dieses und die folgenden Kapitel beschäftigen sich lediglich mit der Infrastruktur rund um den Roboter und der Hardware und den Funktionen, die bereits im Roboter verbaut sind oder ergänzt werden können. Die Funktionen rund um ML, erweiterte Robotik und Lidar werden nicht behandelt. Dieses Kapitel beschäftigt sich mit dem Ansatz der Analyse des Roboters. Es wird gezeigt, wie die bestehenden Funktionen getestet und genutzt werden können, wie erkannt wird, welche Funktionen bereits aktiviert sind und welche Teile der Soft- oder Hardware nicht aktiviert sind. Zum Abschluss werden die bereits vorhanden Funktionen in dem Umfang, in dem sie ab Werk geliefert wurden, gezeigt und erklärt. Einige der Funktionen werden im späteren Verlauf der Arbeit auch erweitert oder verändert. Die Dokumentation hierzu ist in Kapitel 5 zu finden. Betroffene Funktionen werden hier im Kapitel explizit hervorgehoben.

Als Grundlage für die in diesem Kapitel vorausgesetzten Informationen wird Kapitel 3 referenziert. Dieses sollte vor dem Weiterlesen betrachtet werden, falls noch kein umfangreiches Grundwissen über den Aufbau des GO1 vorhanden ist. Ebenfalls zu beachten ist, dass es sich bei dem analysierten Roboter in den folgenden Kapiteln 4 und 5 um die Version *GO1 Edu* handelt, welche deutlich umfangreichere Ausstattung und Softwaremöglichkeiten mitliefert, als die weniger ausgestatteten Versionen des GO1.

## 4.1 Inbetriebnahme

In diesem Kapitel wird die erste Inbetriebnahme des Roboters beschrieben. Hierzu werden auch die Inhalte und Erweiterungen aufgelistet, welche im Lieferumfang für einen GO1 Edu enthalten sind. Anschließend soll der Betrieb durch den Akku oder ein Netzteil beschrieben werden.

### 4.1.1 Lieferumfang

Abbildung 13 zeigt die im Lieferumfang enthaltene Transportbox aus Styropor und deren Inhalte, welche gleichzeitig der gesamte Lieferumfang des GO1 in der *Edu* Version ist.
Im Lieferumfang enthalten sind folgende Elemente:

1. GO1 inklusive Akku
2. Kompakte Fernbedienung mit *Follow-Me*-Funktion
3. Netzteil für das Akku-Ladegerät

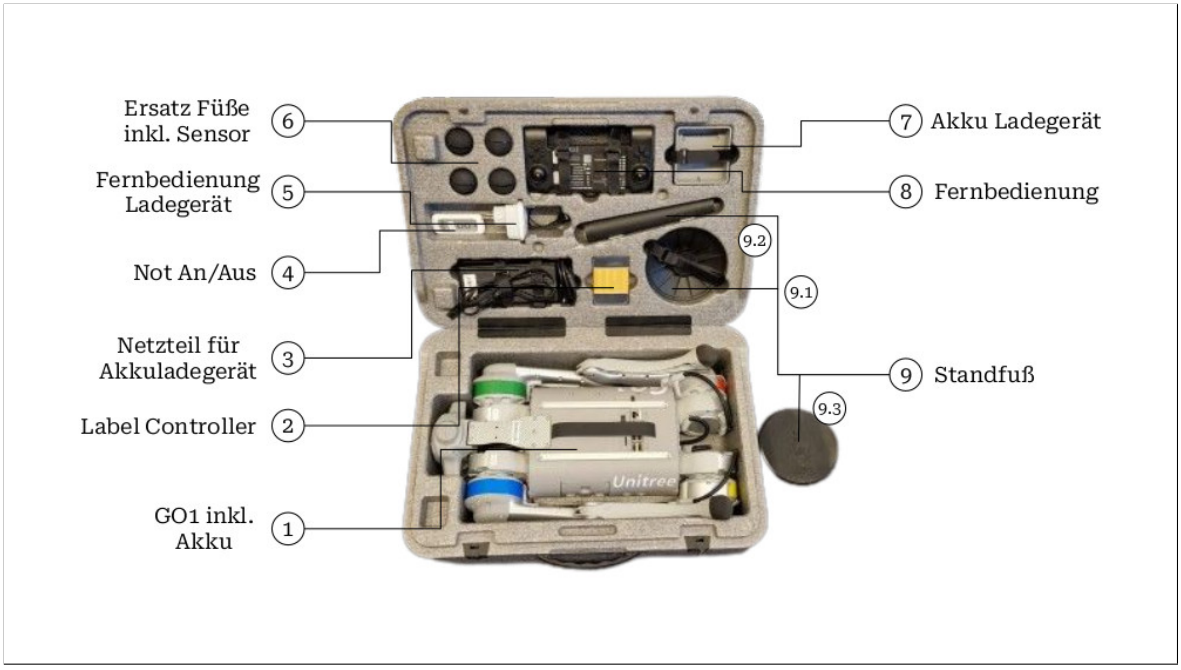


Abbildung 13: Ansicht der Transportbox und des Lieferumfangs

4. Fernbedienung für An-/Ausschaltfunktion
5. Ladegerät für die Fernbedienungen inklusive USB-A auf USB-C Kabel
6. Vier Ersatzfüße inklusive integrierter Drucksensoren
7. Batterieladegerät mit USB-C Anschluss zur Akkuinspektion
8. Fernbedienung
9. Standfuß aus drei Teilen für die Ablage des GO1
   a) Sockel
   b) Verlängerung
   c) Plattform inklusive Einkerbungen zum Ausbalancieren des aufliegenden GO1

Sollte kein weiteres Zubehör konfiguriert sein, so sind an den Montagestellen auf der Oberseite des Roboters Gummienden zum Schutz des Korpus bei Rotationen montiert. Je nach bestelltem Zubehör – wie beispielsweise eines professionellen Lidar Sensors oder eines Roboterarmes am GO1 – sind Schienen an der Oberseite des Rumpfes montiert. In diesem Fall werden die Gummienden sowie eine Schutzabdeckung für die Entwicklerports, die standardmäßig montiert ist, in der Transportbox mitgeliefert.

### 4.1.2 Mobile Inbetriebnahme

Als mobile Inbetriebnahme wird hier die Inbetriebnahme des Roboters mit einer Stromversorgung durch den Akku bezeichnet. Für die Inbetriebnahme auf diese Weise sind folgende Teile nötig:

**Voraussetzungen**

GO1, Akku, Akkuladegerät inklusive Netzteil, Fernbedienung inklusive Ladegerät

Zur Vorbereitung müssen sowohl Akku als auch die Fernbedienung geladen werden. Sind diese ausreichend geladen, kann der Akku eingesteckt und der GO1 in die Ausgangsposition gebracht werden. Hierfür müssen die vier Beine so rotiert werden, dass sowohl Fuß als auch Knie den Boden berühren. Abbildung 14 zeigt die Ausgangsposition des Roboters.

Abbildung 14: Ausgangsposition des Roboters

Durch einfaches Drücken auf dem Knopf über der Ladeanzeige des Akkus wird der Ladestand durch die vier LEDs angezeigt. Durch ein sofortiges weiteres Drücken und Halten des Knopfes startet der Roboter. Dies wird über ein kurzes serielles Aufblinken aller vier LEDs signalisiert. Zudem sind die Lüfter deutlich zu hören. Durch dasselbe Verfahren – Drücken gefolgt von zweitem Drücken und Halten des Knopfes neben der Ladeanzeige auf der Unterseite der Fernbedienung wird auch diese angeschaltet. Ein einmaliges akustisches Signal ertönt beim Anschalten. Die Fernbedienung verbindet sich automatisch mit dem Roboter. Im Werkszustand steht der Roboter nach dem Einschalten nach etwa 70 - 80 Sekunden auf. Der Roboter kann nun über die Fernbedienung gesteuert werden.

Zum Ausschalten des GO1 sollte dieser in eine liegende Position (*Prone*-State) und anschließend in den *Damping*-State gebracht werden. Das wird durch die Tastenkombinationen

`L2+A` und `L2+B` erreicht. Danach kann der Akku wie beim Anschalten durch Drücken und erneutes Drücken und Halten ausgeschalten werden. Die LEDs signalisieren das erfolgreiche Ausschalten und die Lüfter schalten sich aus. Auch die Fernbedienung kann durch dieses Vorgehen ausgeschalten werden. Hier signalisieren drei kurze akustische Signale das erfolgreiche Ausschalten.

### 4.1.3 Stationäre Inbetriebnahme

Im Gegensatz zur mobilen Inbetriebnahme wird hier die Inbetriebnahme des Roboters durch eine stetige Stromversorgung durch ein Netzteil bezeichnet. Hierfür müssen einige Vorbereitungen getroffen werden, da im Lieferumfang kein Netzteil für den Betrieb des GO1 enthalten ist.

**Voraussetzungen**

GO1, Netzteil für Akkuladegerät
**Zusätzlich:** Hohlstecker M 5.5/2.1 mm auf Schraubklemme, XT30-M Stecker mit ausreichend (> 10 cm) Verkabelung. Alternativ: XT30-M auf Hohlstecker M 5.5/2.1 mm Adapter.

Vergleicht man Ausgangsspannung und maximale Stromstärke des verbauten Akkus mit denselben Werten des Netzteils für das Akkuladegerät, so erkennt man, dass das Ladegerät die passende Ausgangsspannung und Leistung vorweist, um den GO1 über den in Kapitel 3.1.4 Abbildung 10 beschriebenen *XT-30* Port zu betreiben. Da das Netzteil lediglich einen Hohlstecker zum Anschluss bietet, muss hierfür jedoch in Eigenarbeit ein Adapter auf XT-30 hergestellt werden. Herbei sind die in den Ressourcen beschriebenen Bauteile nötig. Die Schraubverbindung ist zwar optional und kann durch eine Lötstelle oder ein vorgefertigtes Teil ersetzt werden, sie erleichtert jedoch die Beschaffung und den Zusammenbau des Adapters. Zu beachten ist hierbei nur die korrekte Abmantelung der Kabel für die Schraubklemme und die Orientierung der Kabel in der Klemme. Abbildung 15 zeigt eine beispielhafte Umsetzung des Beschriebenen und die korrekte Orientierung des XT-30 Steckers.

Aus Sicherheitsgründen sollte der sogenannte *Sportmodus* des GO1 deaktiviert werden, da sich dieser sonst nach Start des Roboters aufsteht, was die Verkabelung lösen könnte. Zudem ist die maximale Ausgangsleistung des Netzteils nicht hoch genug, um den Roboter dauerhaft inklusive der Motoren zu betreiben. Der Betrieb wird deshalb stationär genannt. Für das Deaktivieren des Sportmodus muss sich zunächst auf den Raspberry Pi verbunden werden. Im Home-Ordner des Nutzers *pi* befindet sich der Ordner `Unitree/autostart/triggerSport` mit der Datei `triggerSport.sh`. Diese muss lediglich umbenannt werden, beispielsweise zu `triggerSport.disabled.sh`.

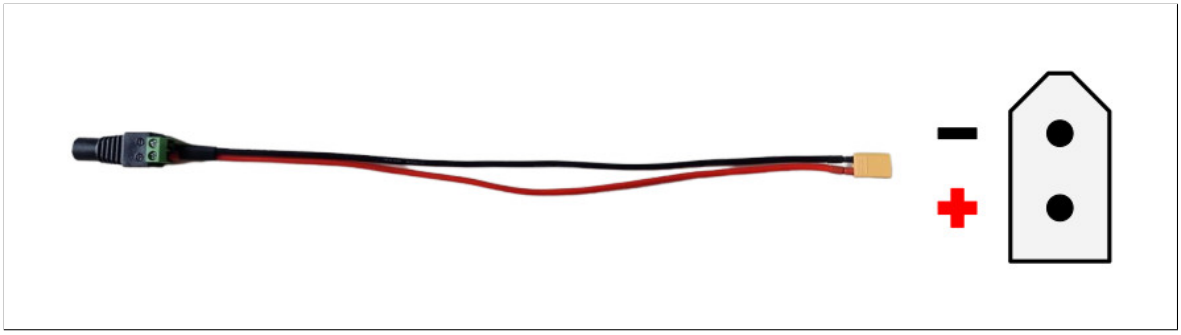

Abbildung 15: XT-30 auf Hohlstecker Verkabelung und XT-30 Orientierung

```
pi@raspberrypi:~ $ cd Unitree/autostart/triggerSport/
pi@raspberrypi:~/Unitree/autostart/triggerSport $ ls
build  log  triggerSport.sh  version.txt
pi@raspberrypi:~/Unitree/autostart/triggerSport $ mv triggerSport.sh
   ↪ triggerSport.disabled.sh
pi@raspberrypi:~/Unitree/autostart/triggerSport $ ls
build  log  triggerSport.disabled.sh  version.txt
```

Beim Neustart des Roboters wird nun nicht automatisch in den Sportmodus geschaltet und die Motoren bewegen sich nicht. Weiteres zur *Autostart*-Funktion des GO1 in Kapitel 4.2.1.

Zuletzt kann das Netzteil inklusive des Adapters in den XT-30 Port auf dem Rücken des Roboters gesteckt werden. Ein erfolgreicher Start des Roboters kann über das akustische Wahrnehmen der Lüfter geprüft werden. Zum Ausschalten muss der Stecker lediglich entfernt und der Roboter somit vom Strom getrennt werden. Hierfür muss nur sichergestellt sein, dass alle manuellen Operationen auf den Recheneinheiten vollendet sind, damit diese nicht korrumpiert werden.

### 4.1.4 Grafische Anwendungen

Nach Inbetriebnahme des GO1 stehen dem Nutzer zwei grafische Anwendungen zur Verfügung, mit denen er Funktionen des Roboters wie Monitoring, Steuerung und Simulationen ausführen kann. Dieses Kapitel soll kurz die erste Einrichtung beziehungsweise das Öffnen der Anwendungen dokumentieren.

#### Webinterface

Der Raspberry Pi des GO1 startet zum Systemstart automatisch einen Webserver mit einer Webseite, die diverse Funktionen zugänglich macht. Für das Aufrufen der Webseite muss man sich im selben Netzwerk wie der GO1 befinden. Weitere Informationen hierzu sind in den Kapiteln 3.3 und 4.2.3 beschrieben. Auf dem HTTP (Hypertext Transfer Protocol)-Port 80

des Pis[39] ist nun die Seite aufrufbar. In Browsern reicht hierfür lediglich die Eingabe der IP Adresse, der Port muss nicht definiert sein. Abbildung 16 zeigt eine Bildschirmaufnahme des Webinterfaces.

Abbildung 16: Screenshot des Webinterfaces

### Mobile App

Unitree Robotics hat als Ergänzung zum Webinterface ebenfalls eine mobile Anwendung für die Plattformen *iOS* und *Android* veröffentlicht. Diese kann fertig gebaut von der Entwicklerwebseite heruntergeladen werden und muss auf den jeweiligen Plattformen als fertige App aus unbekannter Quelle installiert werden[40]. Hierfür sind in der Regel die Entwickleroptionen zu aktivieren. Da die Installationsprozesse für die Plattformen und Endgeräte unterschiedlich sein können, wird in dieser Arbeit nicht darauf eingegangen. [41] Abbildung 17 zeigt zwei Bildschirmaufnahmen der mobilen Anwendung.
Für die Verbindung zum Roboter muss sich lediglich mit dem Netzwerk des Roboters verbunden werden. Die IP Adresse des Roboters muss dann noch in den Einstellungen hinterlegt werden. Danach können die Informationen vom GO1 über die Anwendung eingesehen werden.

[39] Erreichbar auf `192.168.123.161` oder `192.168.12.1` im eigenen WLAN-Hotspot (Siehe 4.2.3)

[40] Unitree Robotics. *Go1 APP Download*. URL: `uhttps://www.unitree.com/app/` (besucht am 17. 08. 2023).

[41] Weitere Informationen zur Installation von Beta-Apps für iOS auf `https://testflight.apple.com/join/KraKgqam` und für Android auf `https://www.heise.de/tipps-tricks/Externe-Apps-APK-Dateien-bei-Android-installieren-so-klappt-s-3714330.html`

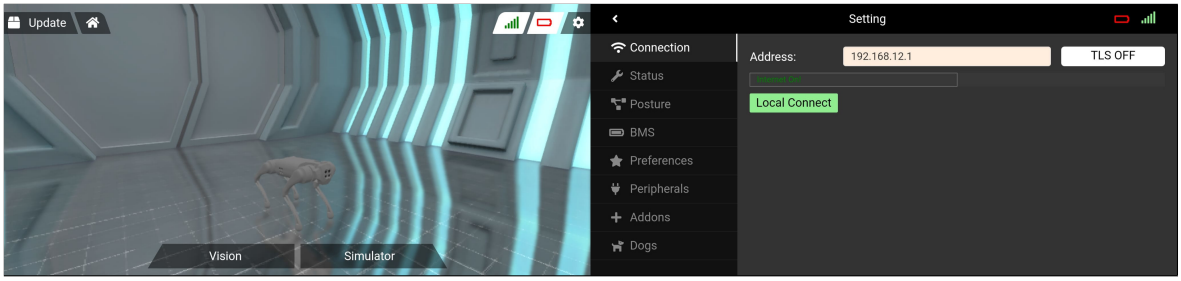


Abbildung 17: Screenshot der mobilen Anwendung

## 4.2 Funktionen

Im folgenden Kapitel werden einige der im Werkszustand mitgelieferten Funktionen des GO1 erklärt und vorgeführt. Einige der Funktionen sind bereits freigeschaltet und bedürfen keiner bis weniger Konfiguration. Andere sind nicht freigeschaltet oder kaum dokumentiert, weshalb diese detaillierter beschrieben werden. Teile der Funktionen sind nicht im Umfang der Arbeit erfasst worden. Alle bekannten nicht erfassten Funktionen werden am Ende des Kapitels aufgezählt.

### 4.2.1 Software Autostart

Unitree Robotics hat bei der Implementierung der Startsequenzen ihrer Softwarekomponenten eine einfache, für den Nutzer zugängliche Autostart Funktion der Betriebssysteme des Raspberry Pis und der NVIDIA Jetsons verwendet. Auf den Ubuntu Systemen wird das Paket `gnome-startup-applications` verwendet. Die Installation des Paketes kann durch den Befehl `apt list -installed | grep gnome-startup-applications` geprüft werden. Anzumerken ist, dass die Funktionalität von der Desktop-Umgebung *Gnome* bereitgestellt wird, nicht vom Betriebssystem der Recheneinheiten. Die auf LXDE (Lightweight X11 Desktop Environment) basierende Desktop-Umgebung PIXEL (Pi Improved Xwindows Environment, Lightweight), die auf dem Raspberry Pi installiert ist, liefert eine andere Möglichkeit der Autostart-Konfiguration. Dennoch ist die Einrichtung identisch zur Bibliothek `gnome-startup-applications`, welche aber nicht auf dem System installiert ist. Eine offizielle Dokumentation für die Autostart-Funktion des Pis ist deshalb nicht verfügbar, es ist lediglich ersichtlich, dass die Funktion dem auf dem Open-Source-Projekt *FreeDesktop* beschriebenen Standard *Autostart Of Applications During Startup* entspricht[42]. Die Funktion wird in diesem Kapitel exemplarisch am Raspberry Pi gezeigt.

#### Implementierung

[42] FreeDesktop.org. *Autostart Of Applications During Startup*. URL: `https://specifications.freedesktop.org/autostart-spec/0.5/ar01s02.html` (besucht am 17. 08. 2023).

Zu Prüfung der Implementierung muss sich vorerst auf den Raspberry Pi verbunden werden. Hier wird zunächst geprüft, welcher Benutzer auf dem System nach einem Boot automatisch eingeloggt wird, damit die Desktop-Umgebung die passenden Programme startet. Erwartet wird hier der Benutzer *pi*, was bei der Prüfung bestätigt wird.

```
pi@raspberrypi:~ $ cat /etc/lightdm/lightdm.conf  | grep autologin-
   ↪ user | grep -v \#
autologin-user=pi
```

Der im FreeDesktop Standard definierte Ordner `/home/pi/.config/autostart/` enthält lediglich eine Datei des Typs `.desktop`. Dateien mit dieser Endung werden von der Desktop-Umgebung verwendet, um Programme zu starten und in der Oberfläche gegebenenfalls weitere Informationen wie Bilder und Beschreibungen anzuzeigen.

```
pi@raspberrypi:~ $ ls /home/pi/.config/autostart/
unitree.desktop
pi@raspberrypi:~ $ cat /home/pi/.config/autostart/unitree.desktop
[Desktop Entry]
Name=unitree
Comment=unitree autostart
Exec=bash /home/pi/UnitreeUpgrade/start.sh
Terminal=false
Type=Application
Categories=System;Utility;Archiving;
StartupNotify=false
NoDisplay=true
```

Der Wert des Feldes `Exec` wird nach dem Boot-Vorgang und dem Einloggen des Benutzers *pi* als Kommandozeilenbefehl interpretiert und ausgeführt. Ein Blick auf die Zeilen 6 und 7 des ausgeführten Skripts verweise lediglich auf ein weiteres Skript.

```
6  cd /home/pi/Unitree/autostart
7  ./update.sh &
```

Das Skript `/home/pi/Unitree/autostart/update.sh` legt eine neue Log-Datei an und initiiert dann den Autostart Prozess.

```
13 for dir in `cat .startlist.sh`
14 do
15   if [[ $dir = \#* ]] ; then
16     echo $dir': skipped' >>/home/unitree/Unitree/autostart/.
         ↪ startlog
17   else
```

```
18      cd $dir
19      echo $dir':'`sed -n '1p' version.txt` >> ${scriptPath}.
           ↪ detailedVersion
20      ./$dir.sh
21      sleep 3
22    fi
23    cd $scriptPath
24  done
```

Der Inhalt der Datei `.startlist.sh` ist eine durch Zeilenumbrüche getrennte Liste aller Ordner, welche zum Autostart nach ausführbaren Skripten durchsucht werden. In Zeile 13 wird über alle diese Ordner iteriert. Zeile 15 prüft, ob der aktuelle Ordner in der Liste auskommentiert wurde und überspringt diesen gegebenenfalls. In Zeile 18 wird andernfalls in den Ordner navigiert und in Zeile 20 jenes Skript in dem Ordner ausgeführt, das denselben Namen wie der Ordner selbst inklusive der Dateiendung `.sh` hat.

Zur Erweiterung der Autostart-Funktion oder dem Hinzufügen eigener Prozesse nach Systemstart muss somit lediglich ein Ordner im Pfad `/home/pi/Unitree/autostart/` angelegt werden, in dem eine ausführbare Datei mit dem Namen des Ordners inklusive der Dateiendung `.sh` liegt. Weitere Informationen zu dieser Vorgehensweise und ein Beispiel der Erweiterung werden in Kapitel 5.2 gezeigt.

### 4.2.2 Fernsteuerung

Der GO1 lässt sich auf vier verschiedene Arten fernsteuern:

- Fernbedienung
- App
- Webinterface
- Folgefunktion

Dieses Kapitel beschreibt alle vier Möglichkeiten kurz und zeigt diverse Limitierungen der einzelnen Umsetzungen auf. Für alle vier Steuermöglichkeiten muss der Roboter angeschaltet und der Sportmodus aktiviert sein. Sollte der Sportmodus nicht aktiviert sein und eine Verbindung auf den Raspberry Pi nicht möglich oder erwünscht sein, so kann dieser über die gekoppelte Fernbedienung und der Tastenkombination `L2+START` aktiviert werden. Der GO1 muss sich hierfür in der Ausgangsposition wie in Kapitel 4.1.2 beschrieben befinden.

**Fernbedienung**

Der Lieferumfang des GO1 umfasst zwei physische Fernbedienungen, mit denen der Roboter gesteuert werden kann. Die Hauptfernbedienung besitzt zwei sogenannte Joysticks, welche die Position und Bewegung Roboter in verschiedenen Achsen manipulieren können. Die zweite Fernbedienung, von Unitree *Label Controller* genannt, besitzt lediglich ein Joystick, welches die Bewegung nach vorne, hinten, links und rechts steuern kann. Sie dient ebenfalls als Sender für die *Folge*-Funktion. Abbildung 18 zeigt links die Hauptfernbedienung und rechts den Label-Controller.

Abbildung 18: Hauptfernbedienung (links) und Label-Controller (rechts)

Die einfache Bedienung der Hauptfernbedienung ist bereits nach dem Anschalten des Roboters nach Kapitel 4.1.2 möglich. Hierbei koppelt sich die Fernbedienung automatisch mit dem Roboter. Dokumentation über die Art der Verbindung ist nicht auffindbar, es wird jedoch vermutet, dass dies nicht über Bluetooth, sondern über ein anderes Protokoll geschieht und sich die Fernbedienung mit der MCU statt mit dem Raspberry Pi verbindet, welcher auch über Bluetooth verfügen würde. Es beseht jedoch die Möglichkeit, die Fernbedienung via Bluetooth mit einem Smartphone zu koppeln. Hierfür muss zur Verifikation der Kopplung der Pin `1234` verwendet werden. Der Kopplungsprozess ist unter den verschiedenen Herstellern unterschiedlich und wird deshalb hier nicht weiter dokumentiert. Nach erfolgreicher Kopplung kann über die mobile App[43] die Fernbedienung verbunden werden. Über die Einstellungen und den Menüpunkt `Peripherals > Bluetooth Gamepad` kann in der Liste `Gamepad List` die Fernbedienung mit der übereinstimmenden Seriennummer ausgewählt werden. Diese ist ab Werk auf den Fernbedienungen gelabelt. Abbildung 19 zeigt die Bildschirmaufnahmen der relevanten Menüpunkte der App.

Nun besteht die Möglichkeit, den GO1 über die Fernbedienung fernzusteuern, egal in welchem Netzwerk er sich befindet. Es ist lediglich notwendig, dass sich der Roboter und das Handy im

[43] Siehe Kapitel 4.1.4

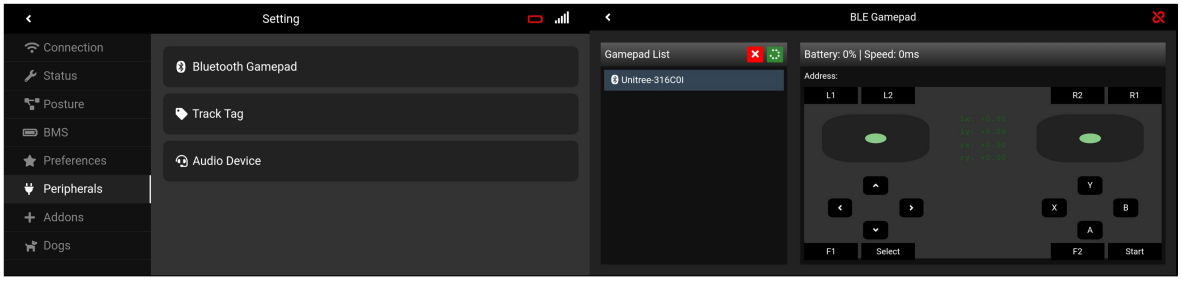


Abbildung 19: App-Menüpunkte `Peripherals > Bluetooth Gamepad` und `Gamepad List`

selben Netzwerk befinden. Weitere Informationen zur Netzwerkerweiterung sind in Kapitel 5 dokumentiert.

### App

Der GO1 lässt sich ebenfalls durch die mobile Anwendung steuern, ohne sie vorher mit der Hauptfernbedienung verbunden zu haben. Hierfür muss der Hauptmenüpunkt `Vision` ausgewählt werden. Danach muss im rechten oberen Eck die Steuerung in der Ansicht der Kameras aktiviert werden. Wie auf Abbildung 20 gezeigt, kann dann der Modus im rechten unteren Eck des Bildschirmes gewählt werden. Wählt man hier einen der Laufmodi aus, so lässt sich der GO1 mit den beiden dargestellten Steuereinheiten bewegen. Links stellt die linke Steuereinheit der Hauptfernbedienung dar, rechts die rechte Einheit.

Abbildung 20: Bildschirmaufnahme des App-Controllers

### Webinterface

Die Webseite, die auf dem Raspberry Pi gehostet wird, bieter im Menü `Vision` die Möglichkeit, im rechten oberen Eck des Bildschirmes die Steuerung des Roboters zu aktivieren. Nach Aktivierung werden dem Nutzer, wie auf Abbildung 21 dargestellt, zwei Steuerelemente dargestellt. Diese können links mit den Tasten `W-A-S-D` und rechts mit den Tasten ↑-←-↓-→ gesteuert. Die linke Steuereinheit entspricht der linken Seite der Hauptfernbedienung, die rechte dementsprechend der rechten Seite.

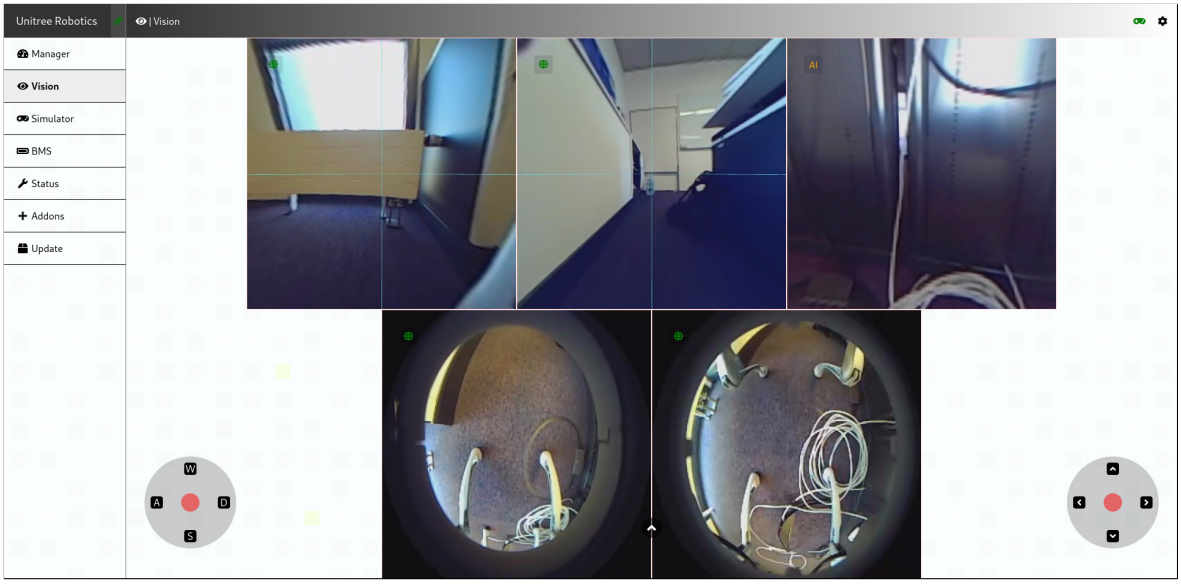


Abbildung 21: Bildschirmaufnahme des Web-Controllers

**Folgefunktion**

Der sogenannte Label-Controller lässt sich genauso wie die Hauptfernbedienung über die Tastenkombination Drücken und erneutes Drücken und Halten des `POW`-Knopfes anschalten. Mit dem Joystick lässt sich der Roboter vereinfacht steuern. Die eigentliche Funktion des Label-Controllers ist jedoch die von Unitree Robotics beworbene *Follow Me* Funktion. Der Controller kommuniziert ständig mit dem Roboter und tauscht Informationen zur angenäherten Position des GO1 relativ zum Label-Controller aus. Dies lässt sich in der mobilen App beobachten. Hierfür muss über die Einstellungen auf den Pfad `Peripherals > Track Tag` navigiert werden. Abbildung 22 zeigt die beiden Bildschirmaufnahmen.

Um den Roboter zum Folgen zu bringen, muss der Wert der Option `Follow` angetippt werden. Dieser sollte nun von `OFF` auf `ON` wechseln. Danach bewegt sich der Hund immer relativ zum Label-Controller. Durch die Tasten des Label-Controllers lässt sich das verhalten minimal anpassen. Tabelle 4 zeigt fasst die Tastenfunktionen zusammen.
Anzumerken ist, dass der Label-Controller nicht per Bluetooth mit dem Handy verbunden werden kann.

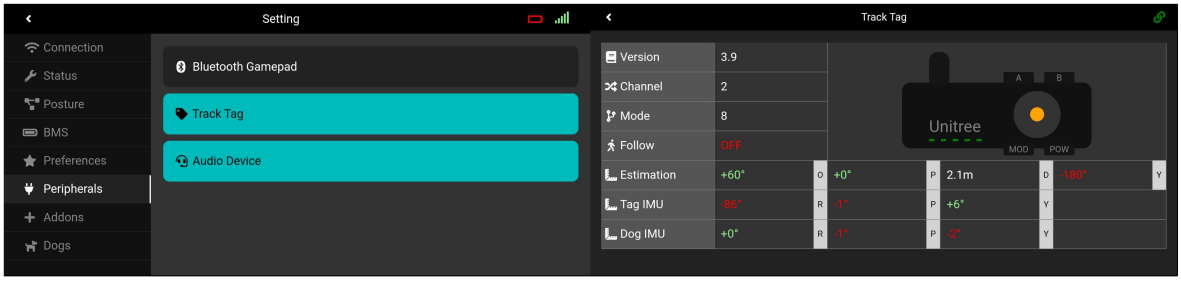


Abbildung 22: App-Menüpunkte `Peripherals > Track Tag` und die Tracking-Übersicht

| Taste | Funktion |
|---|---|
| POW | 1 Sekunde Drücken → Aufrichten nach Sturz<br>2 mal kurz Drücken → Wechsel der Standmodi<br>Stehen → Legen → Motoren deaktivieren → Stehen |
| MOD | Kurzes Drücken → Folgen deaktivieren<br>2 mal kurz Drücken → Wechsel der Modi<br>Langsames Folgen (1.5 m/s) → Schnelles Folgen (3 m/s) |
| A | Wechsel Ausweichmodus (nach Test nicht Funktionsfähig) |
| B | Kurzes Drücken → Rotation gegen Uhrzeigersinn um etwa 6°<br>2 mal kurz Drücken → Reset der Rotation auf Standardwert |

Tabelle 4: Zusammenfassung der Tastenfunktionen des Label Controllers

### 4.2.3 Lokales Netzwerk

Der Raspberry Pi des GO1 nutzt eines seiner Netzwerk-Schnittstellen, um ein WLAN zu publizieren. Ein Blick auf das Autostartmodul `configNetwork` zeigt, dass hierfür das Interface `wlan1` verwendet wird.

```
35  # configure WIFI
36  sudo ifconfig wlan1 192.168.12.1/24
37  sudo ifconfig wlan1 up
38  sudo hostapd /etc/hostapd/hostapd.conf&
```

Um weitere Informationen Konfiguration des WLAN zu finden, muss die hostapd (Host Access Point Daemon) Konfiguration in `/etc/hostapd/hostapd.conf` betrachtet werden. Der Service `hostapd` ist ein für Nutzer eines Systems verfügbarer Service für diverse Access Points und Authentifizierungsserver[44].

```
18  wpa=2
19  wpa_key_mgmt=WPA-PSK
20  rsn_pairwise=CCMP
21  ssid=Unitree_Go501075A
```

[44] Jouni Malinen. *hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator*. 12. Jan. 2013. URL: `https://w1.fi/hostapd/` (besucht am 18. 08. 2023).

```
22  wpa_passphrase=00000000
```

Hier ist ersichtlich, dass die SSID des WLAN Access Points der Seriennummer des jeweiligen GO1 entspricht. Diese ist auf diversen Teilen des Roboters gekennzeichnet. Das Standardpasswort ist `00000000` und kann in der Konfigurationsdatei ebenfalls geändert werden. Nach Neustart des Interfaces `wlan1` tritt dieses neue Passwort in Kraft.

Ebenfalls hilfreich kann die Funktion einer versteckten SSID sein, da der Access Point des GO1 sonst in unmittelbarer Nähe zum Roboter für jedermann einsichtig wäre, was ein potenzielles Sicherheitsrisiko im regulären Betrieb darstellt. Hierfür muss lediglich die Zeile `ignore_broadcast_ssid=1` am Ende der Datei hinzugefügt werden. Die Einstellung bewirkt, dass die Publizierung des WLAN keine SSID bekannt gibt und Verbindungsanfragen ohne eine Angabe der vollständigen SSID ignoriert werden.

### 4.2.4 Audio Interfaces

Im Kopf des GO1 ist wie in Abschnitt 3.2.2 beschrieben ein Lautsprecher verbaut. Ein Blick auf die Hierarchie der verbundenen USB Geräte zeigt, dass das Gerät mit der ID `Dev 6` die Klasse Audio hat.

```
unitree@unitree-desktop:~$ lsusb -t | grep Class=Audio
    S|__ Port 4: Dev 6, If 0, Class=Audio, Driver=snd-usb-audio, 12M
    S|__ Port 4: Dev 6, If 1, Class=Audio, Driver=snd-usb-audio, 12M
    S|__ Port 4: Dev 6, If 2, Class=Audio, Driver=snd-usb-audio, 12M
unitree@unitree-desktop:~$ lsusb | grep "Device 006"
Bus 001 Device 006: ID 0d8c:0012 C-Media Electronics, Inc.
```

Der Mount-Point `/dev/snd/controlC2` des Gerätes lässt sich über den Sym-Link im Ordner `/dev/snd/by-id/` auslesen.

```
unitree@unitree-desktop:~$ ls -l /dev/snd/by-id/
total 0
lrwxrwxrwx 1 root root 12 1x  28 23:58 usb-C-Media_Electronics_Inc.
   ↪ _USB_Audio_Device-00 -> ../controlC2
```

Im Folgenden werden zwei Möglichkeiten gezeigt, um den Lautsprecher des Roboters ohne weitere Konfiguration zu nutzen.

#### Sprachdurchgabe über App

Die mobile App für den Roboter ermöglicht es, Mikrofon-Aufnahmen des Gerätes, auf dem die App installiert ist, an den GO1 zu senden und dann über den Lautsprecher auf der

Rückseite des Kopfes abzuspielen. Hierfür muss die Anwendung zuerst mit dem Roboter verbunden werden. Danach kann über den Menüpunkt `Peripherals > Audio Device` auf die Oberfläche zur Konfiguration der Übertragung zugegriffen werden. Abbildung 23 zeigt die Übersicht der Audioübertragung.

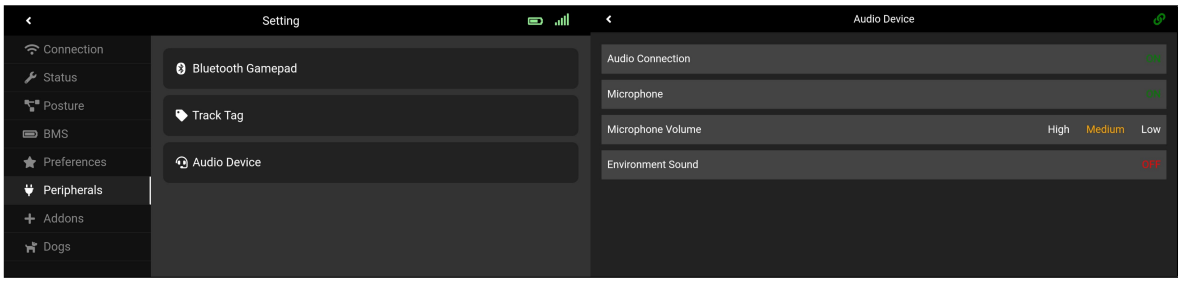


Abbildung 23: Überblick über Netzwerkkonfiguration

Das Icon im rechten oberen Bildrand zeigt an, ob das Audiointerface im Kopf des Hundes erfolgreich verbunden wurde. Hierfür muss der Autostartprozess `wsaudio` auf dem Nano noch in Betrieb sein. Das kann über folgenden Befehl geprüft werden.

```
unitree@unitree-desktop:~$ ps -aux | grep wsaudio
unitree    8205 95.0  0.4 466312 18360 ?        Rl   00:00   0:23 ./
   ↪ build/wsaudio
```

Sollte dieser Prozess nicht aktiv sein, so kann das Autostartskript manuell ausgeführt werden.

```
unitree@unitree-desktop:~$ cd /home/unitree/Unitree/autostart/wsaudio
   ↪ /
unitree@unitree-desktop:~/Unitree/autostart/wsaudio$ ./wsaudio.sh &
[1] 9179
[wsaudio] starting ...
unitree@unitree-desktop:~/Unitree/autostart/wsaudio$ ps -ax | grep
   ↪ wsaudio
 9179 pts/0    Rl     3:15 ./build/wsaudio
```

Wichtig ist bei der manuellen Ausführung von Autostartskripten, dass diese aus ihrem Ordner heraus gestartet werden, da die Skripte selbst oftmals mit relativen Pfaden arbeiten.

In der mobilen Anwendung kann nun der Wert `Audio Connection` gewählt werden, um diesen von `OFF` auf `ON` zu schalten. Sobald dies auch für den Wert `Microphone` erledigt wurde, kann über das eingebaute Mikrofon des Handys, auf dem die Anwendung läuft, Audio aufgenommen werden. Das Aufgenommene wird dann mit Netzwerk-bedingter Latenz auf dem Lautsprecher abgespielt.

**Abspielen von Audiodateien**

Für das Abspielen von Audiodateien müssen diese erst auf den Nano im Kopf des Roboters kopiert werden. Hierfür kann die auf den meisten Linux-basierten Betriebssystemen installierte Funktion `scp` verwendet werden. Ist der Rechner mit der Datei im Netzwerk des GO1, so kann folgender Befehl verwendet werden, um die Datei zu kopieren:

```
sshpass -p 123 scp beispiel.wav unitree@192.168.123.13:~/Music
```

Um den Lautsprecher nutzen zu können, müssen alle Prozesse, die diesen blockieren erst beendet werden. Ab Werk ist das nur der im vorigen Paragraf beschriebene `wsaudio`-Prozess, der mit dem Kommando `pkill -f wsaudio` beendet werden kann. Danach kann mit dem Befehl `aplay` die Datei wiedergegeben werden. Hierfür wird der Gerätename benötigt, welcher mit dem Befehl `aplay -L`, in dessen Ausgabe nach dem Gerät mit der Kennung 2 gesucht wird, ausgelesen werden kann. Die Kennung wurde am Anfang des Kapitels über den Mount-Point erfasst. Die Ausgabe der Datei `/proc/asound/cards` zeigt den USB Lautsprecher unter dem Index 2.

```
unitree@unitree-desktop:~$ cat /proc/asound/cards
[...]
 2 [Device          ]: USB-Audio - USB Audio Device
                       C-Media Electronics Inc. USB Audio Device at
                          ↪ usb-70090000.xusb-3.4, full speed
```

Die Ausgabe der Datei `/proc/asound/card2/pcm0p/info` zeigt, dass die Karte mit dem Index 2 nur einen Kanal hat, der für das Abspielen verwendet werden kann [45].

```
unitree@unitree-desktop:~$ cat /proc/asound/card2/pcm0p/info
card: 2
device: 0
subdevice: 0
stream: PLAYBACK
id: USB Audio
name: USB Audio
subname: subdevice #0
class: 0
subclass: 0
subdevices_count: 1
subdevices_avail: 1
```

[45] `card2` steht hier für den Index 2 der `/proc/asound/cards`-Liste, `pcm0p` für das erste `PLAYBACK` Gerät zur Ausgabe

Dadurch ergibt sich der direkte Hardwarename des Lautsprechers `plughw:2,0` [46], welcher für folgenden Befehl benötigt wird:

```
unitree@unitree-desktop:~$ aplay -D plughw:2,0  ~/Music/beispiel.wav
Playing WAVE '/home/unitree/Music/beispiel.wav' : Signed 16 bit
   ↪ Little Endian, Rate 44100 Hz, Stereo
```

Zur einfacheren Handhabung von einem externen Gerät können auch folgende Befehle verwendet werden, insofern das Gerät mit dem Netzwerk des GO1 verbunden ist:

```
sshpass -p 123 ssh unitree@192.168.123.13 'pkill -f wsaudio'
sshpass -p 123 ssh unitree@192.168.123.13 'aplay -D plughw:2,0 ~/
   ↪ Music/beispiel.wav'
```

Um die Lautstärke des Lautsprechers anzupassen, stellt die Bibliothek, die auch `aplay` enthält, ebenfalls die Funktion `amixer` zur Verfügung. Durch den Befehl `amixer -c 2 set Speaker <0-100>%`. Die Option `-c 2` verweist hier wieder auf die Gerätekennung.[47]

### 4.2.5 Kopfbeleuchtung

Die LED-Reihen an den beiden Außenseiten des Kopfes sind, wie in Kapitel 3 beschrieben, am Jetson Nano des Kopfes angeschlossen. Prüft man dort die Funktionen, die über den Autostart gesteuert werden, so fallen zwei Ordner auf – `faceLightServer/` und `faceLightMqtt/`. Leider sind die Funktionalitäten beider Skripte als Binärdateien abgelegt und somit nicht ohne weiteres auslesbar. Auch die Dokumentation des Herstellers weist keinerlei Informationen zu den LED-Reihen auf. Der Name des zweiten Ordners weist jedoch auf eine mögliche Funktionalität der Steuerung über MQTT hin. Um dies zu bestätigen, lässt sich ein MQTT-Explorer verwenden. Dieser registriert sich als Client beim MQTT-Broker und schreibt alle Nachrichten und veröffentlichten Topics mit. Mehr zum Thema MQTT gibt es auf der offiziellen Dokumentation des Standards[48], Beispiele zur Nutzung des Standards am Roboter in Kapitel 5.

Zur Nutzung des MQTT Explorers wird die IP Adresse des Brokers und der Port, auf dem dieser veröffentlicht wird, benötigt. Für die Adresse kommen nur die IPs `192.168.123.13`, `...14`, `...15` und `...161` infrage. Es kann mit der Bibliothek *Nmap* geprüft werden, ob der MQTT-Standardport `1883` auf einer der registrierten IPs im Netz `192.168.123.0/24` geöffnet ist.

[46] `plug` steht hier für gesteckte Hardware `hw` (z.B. USB), 2 für den Index und 0 für den Kanal (subdevice)

[47] Alsa-Projekt.org. *Advanced Linux Sound Architecture (ALSA) project homepage*. 20. Okt. 2020. URL: `https://www.alsa-project.org/wiki/Main_Page` (besucht am 18. 08. 2023).

[48] https://mqtt.org/

```
nmap -sS -O -p1883 192.168.123.0/24
```

Die Ausgabe zeigt, dass die beiden IPs `192.168.123.15` und `192.168.123.161` den MQTT Port offen haben. Über den MQTT Explorer wird zunächst der NVIDIA Jetson Xavier NX als Broker getestet. Die Verbindung funktioniert, jedoch werden weder verfügbare Topics noch Messages ausgegeben. Ein Test auf dem Raspberry Pi mit der IP `192.168.123.161` zeigt die Ausgabe wie in Abbildung 24 dargestellt.

Abbildung 24: Ausgabe eines MQTT Explorers in Verbindung mit dem Raspberry Pi als Broker

In den angezeigten Topics sind aktuell noch keine Informationen zu den LEDs zu erkennen. Dies kann daran liegen, dass das zu dem Topic noch keine Nachrichten versendet wurden, was eine plausible Schlussfolgerung ist, da die Kopflichter des GO1 standardmäßig ausgeschalten sind. Ändert man dies nun durch die in der mobilen Anwendung gegebenen Funktion unter dem Menüpunkt `Preferences`, so sieht man auch das Topic `face_light/color` im MQTT-Explorer, wie in Abbildung 25 dargestellt.

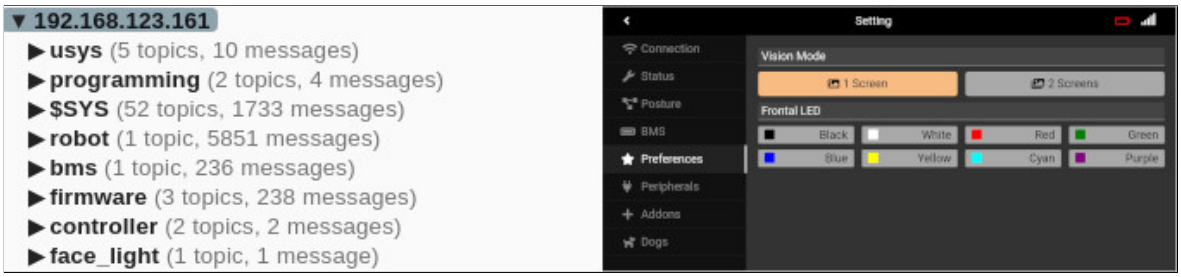


Abbildung 25: MQTT Explorer mit Topic `face_light/color` (links) und App-Funktion (rechts)

Die Message Payload kann beispielsweise über das Commandline-Tool *Mosquitto-Client* ausgegeben werden. Hierfür muss folgender Befehl genutzt werden, um die binäre Payload lesbar auszugeben. Der genutzte Rechner muss sich hierfür im Netzwerk des GO1 befinden.

```
mosquitto_sub -h 192.168.123.161 -t face_light/color -F %x
ff0000
00ff00
0000ff
```

Die drei Werte in den letzten der Zeilen der Ausgabe sind formatierte Message-Payloads des Topics `face_light/color` und wurden ausgegeben, als in der mobilen Anwendung die drei Farben *Rot*, *Grün* und *Blau* in eben dieser Reihenfolge eingestellt wurden. Somit ist herleitbar, dass die LEDs des Roboters über den *RGB*-Farbraum konfigurierbar sind. Die Mischung aus Rot, Grün und Blau kann hier pro Farbe mit einem Wert von 0 - 255 eingestellt werden.

Über den Befehl `mosquitto_pub` kann nun auch die Farbe des Roboters geändert werden. Hierfür muss nur das Versenden der Message-Payload in binärer Form beachtet werden. Folgender Befehl stellt das Kopflicht des GO1 auf Rot um.

```
echo -ne "\xFF\x00\x00" | mosquitto_pub -h 192.168.123.161 -t
   ↪ face_light/color -s
```

Es ist ebenfalls denkbar die LEDs des Roboters direkt über die `CP210x UART Bridge` zu steuern, die in Kapitel 3.2.2 erwähnt wurde. Dies wurde im Rahmen dieser Arbeit jedoch nicht umgesetzt und kann in Zukunft noch dokumentiert werden.

### 4.2.6 Video Streaming

Der GO1 bietet mit seinen fünf Kameras die Möglichkeit, Bilder seiner Umgebung zu übertragen und es den Nutzern so zu ermöglichen, den Roboter aus der Entfernung zu steuern. Die Positionierung, Verteilung und die Mounting-Points der Kameras innerhalb des Roboters, den Recheneinheiten und den Betriebssystemen wurde bereits in Kapitel 3.2 geschildert. Kurz zusammengefasst sind im Kopf des Roboters zwei Kameras positioniert, nach vorne und nach unten gerichtet. Beide sind am Jetson Nano innerhalb des Kopfes verbunden. Die beiden Außenseiten des Rumpfes sind mit zwei Kameras bestückt, die mit dem Jetson Nano m Rumpf des GO1 verbunden sind. Die letzte Kamera an der Unterseite des Rumpfes ist mit dem Jetson Xavier NX verbunden. Am Beispiel der nach vorne gerichteten Kamera im Kopf des GO1 soll in diesem Kapitel kurz erläutert werden, wie auf die Kameras zugegriffen werden kann und wie man von einem verbundenen Rechner außerhalb des Roboters auf die Bilder zugreifen kann. Die dargestellte Anleitung ist für alle anderen Kameras bis auf etwaige Mounting-Points und IP Adressen identisch.

#### Zugriff auf Kamerabilder

Um die Kameras des GO1 nutzen zu können, müssen zuerst alle Prozesse gestoppt werden, die die Geräte selbst blockieren. Geprüft werden kann dies über den Befehl `fuser -vm /dev/video1`, hier muss nach der Ausgabe nach den Zeilen gesucht werden, die als `ACCESS`-Flag den Wert `m` für `memory mapped files` haben. Die Prozesse können dann mit deren Namen beendet werden.

```
pkill -f point_cloud_nod
pkill -f example_point
```

Für den Zugriff auf die Daten der Kamera über den Mount-Point `/dev/video1` kann das Paket `ffmpeg` genutzt werden, das auf allen Ubuntu Systemen des GO1 vorinstalliert ist. Folgender Befehl inklusive Erläuterung zu den Optionen kann genutzt werden, um per `ffmpeg` einen Videostream über RTSP (Real-Time Streaming Protocoll) auf einen Streaming-Server zu starten.

```
ffmpeg -nostdin \                    # Keine Interaktion
    -f video4linux2 \                # Input Format
    -i /dev/video1 \                 # Input-URL
    -vcodec libx264 \                # Video Kodierung (h264)
    -preset:v ultrafast \            # Kodierungsgeschwindigkeit
    -tune zerolatency \              # Keine H264 B-Frames
    -framerate 5 \                   # Framerate
    -f rtsp \                        # Output Format
    rtsp://<ip:8554|port>/<stream> # Output File/URL
```

Weitere Details zur Ausführung und dem Streaming über einen Server werden in Kapitel 5 behandelt. Das Kamerabild ist beim Streaming über `ffmpeg` in keiner Form verarbeitet und sieht wie in Abbildung 26 dargestellt aus.

Eine Verarbeitung und Verbindung der beiden *Fischaugen* der Kamerabilder ist im Roboter bereits mit der Bibliothek *OpenCV* umgesetzt. Auch die in Kapitel 2.2.3 genannte *Unitree-Camera-SDK* nutzt OpenCV in der Implementierung. Im Rahmen dieser Arbeit wird jedoch nicht auf die erweiterten Funktionen durch den Einsatz von OpenCV eingegangen. Hierfür kann die Arbeit „Entwicklung eines Intelligenten lidarbasierten 3D Navigationssystems für den Unitree GO1“[49] zu Rate gezogen werden.

### 4.2.7 Batterie Management

Der Herstellerdokumentation und Werbung ist an einigen Stellen zu entnehmen, dass im GO1 ein intelligentes BMS verbaut ist. Dieses ermöglicht es Nutzern, in Echtzeit Informationen

[49] Kemnitzer, s. Anm. 1.

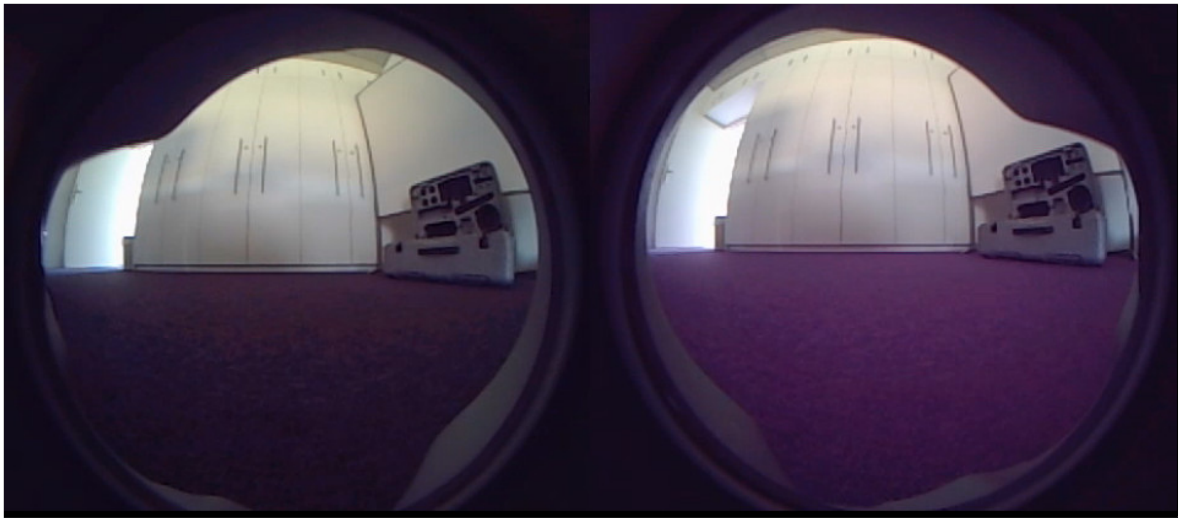

Abbildung 26: Kamerabild des GO1

zum Stand der Batterie abzugreifen und gegebenenfalls auf die Informationen zu reagieren. In der Herstellerdokumentation des Roboters ist nicht dokumentiert, wie die Daten erfasst oder interpretiert werden können, es wird lediglich auf die beiden Übersichten in der mobilen Anwendung und der Webseite verwiesen, die in Abbildung 27 dargestellt werden.

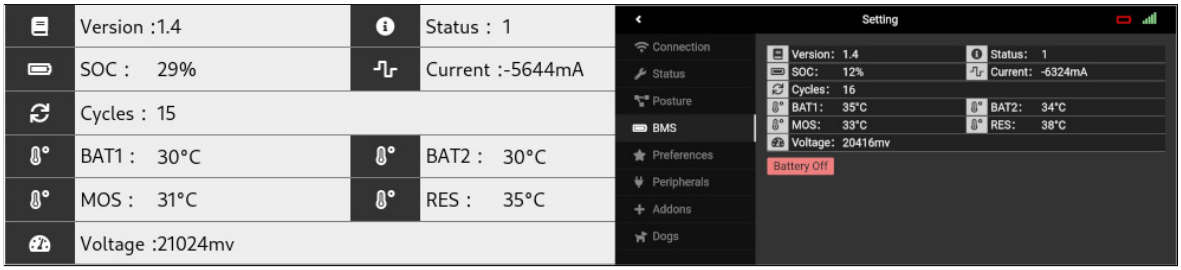


Abbildung 27: Batterieinformationen in der Webseite (links) und App (rechts)

Die Ausgabe des MQTT-Explorers aus Kapitel 4.2.5 beinhaltet ein Topic namens `bms/state`. Auch die Prüfung der Webseite über die Entwicklertools innerhalb moderner Browser weist auf die Bereitstellung der BMS Daten über MQTT hin. Der Dateipfad `src/plugins/mqtt/receivers/bmsReceivers.ts` und das konfigurierte MQTT-Topic `bms/state` bestätigen dies. Gibt man nun die Message-Payloads des Topics aus, so erhält man folgendes Ausgabeformat.

```
mosquitto_sub -h 192.168.123.161 -t bms/state -F %x
0104011bf5e8ffff0f001e1e1f23800da00d20002000a00da00da00d20002000800d
```

Man kann über die Entwicklertools die Struktur der Daten zurückverfolgen. Zeile 14 der Datei `bmsReceivers.ts` zeigt die Umwandlung der Message-Payload aus einem `Byte Buffer` in ein `Uint8Array`. Laut der JavaScript-Dokumentation ist ein `Uint8Array` ein Ar-

ray aus 8-bit unsigned little Endian Integer[50]. Somit können je zwei Ziffern der hexadezimalen Ausgabe des BMS als ein Wert des Arrays interpretiert werden. Die Zeilen 14 bis 17 und Zeile 20 in Listing 4 zeigen die Umwandlungen der 8-bit Integer.

```
14 const uint8s = new Uint8Array(message);
15 data.bms.version = uint8s[0] + "." + uint8s[1];
16 data.bms.status = uint8s[2];
17 data.bms.soc = uint8s[3];
18 data.bms.current = dataView.getInt32(4, true);
19 data.bms.cycle = dataView.getUint16(8, true);
20 data.bms.temps = [uint8s[10],uint8s[11],uint8s[12],uint8s[13]];
21 for (let i = 0; i < 10; i++) {
22   data.bms.cellVoltages[i] = dataView.getUint16(14+i*2, true);
23 }
24 data.bms.voltage = data.bms.cellVoltages.reduce((a,c) => a+c);
```

Listing 4: Inhalt der Webpack Datei `bmsReceivers.ts`

Die Dokumentation der Klasse `DataView` zeigt, das die Funktionen `getInt32()` und `getUint16()` folgenden Syntax haben[51].

```
getInt32(byteOffset, littleEndian)
getUint16(byteOffset, littleEndian)
```

Folglich wird in Zeile 18 ein 4-Byte Integer ab dem fünften Byte der Message-Payload ausgelesen, in den Zeile 19 ein 2-Byte Integer ab dem neunten Byte und zehn weitere ab dem fünfzehnten Byte der Payload gelesen. Die zehn letzten 2-Byte Integer werden zu einer Gesamtzahl addiert. Durch die Auswertungen der Webseite ergibt sich folgender Überblick über das Format der Payload. Das verwendete Beispiel ist die oben gezeigte Ausgabe des Befehls `mosquitto_sub`.

Im Kapitel 5 wird gezeigt, wie die Informationen des BMS sinnvoll ausgelesen und verwertet werden können.

[50] MDN contributors. *Uint8Array*. 14. Aug. 2013. URL: `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Uint8Array` (besucht am 22. 08. 2023).

[51] MDN contributors. *DataView*. 21. Aug. 2013. URL: `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/DataView?retiredLocale=de` (besucht am 22. 08. 2023).

| Byte | Format | Inhalt | Beispiel | Konvertierung | |
|---|---|---|---|---|---|
| 0-1 | 2 x uint8 | Version | 01, 04 | v1.4 | |
| 2 | uint8 | Status | 01 | 1 | |
| 3 | uint8 | State of Charge | 1b | 27 % | |
| 4-7 | int32 | Strom | f5e8ffff | -5899 mA | |
| 8-9 | uint16 | Zyklus | 0f00 | 15 | |
| 10-13 | 4 x uint8 | Temperaturen | 1e, 1e, 1f, 23 | 30 °C, 30 °C 31 °C, 35 °C | |
| 14-33 | 10 x uint16 | Zell-Spannung | 800d, a00d, 2000, 2000, a00d, a00d, a00d, 2000, 2000, 800d | 3456 mV, 3488 mV, 32 mV, 32 mV, 3488 mV, 3488 mV, 3488 mV, 32 mV, 32 mV, 3456 mV | **Summe:** 20,992 V |

Tabelle 5: Übersicht der BMS Message Payload

## 4.3 Weitere Funktionen

Neben den in dieser Arbeit dokumentierten Funktionen bringt der GO1 weitere Funktionen mit, die hier kurz aufgelistet und erläutert werden sollen.

- **ROS:** Auf einigen der Recheneinheiten des Roboters ist ROS installiert, was für umfangreiche Funktionserweiterungen im Bereich der Robotik genutzt werden kann.
- **SLAM (Simultaneous Localization and Mapping):** Der Roboter verfügt über die mobile Anwendung und einige vorinstallierte Bibliotheken die Funktion, durch die verbaute Sensorik ein Bild der Umgebung zu mappen und sich in diesem zu lokalisieren.
- **OpenCV Kameraauswertung:** Die mitgelieferte Bibliothek *Unitree-Camera-SDK* nutzt OpenCV zur Auswertung der Kamerabilder. Durch OpenCV können die Fischaugen der Kameras kombiniert in ein Bild zusammengefasst, die Tiefenmessung der Kamera genutzt und die Bilder ausgewertet werden.
- **Ultraschallsensorik:** Die Ultraschallsensoren können ebenfalls über eine offizielle Bibliothek ausgelesen werden um auf Bewegungen und Hürden im Umfeld des Roboters zu reagieren.
- **Bewegungssteuerung:** Die Motoren des GO1 können in zwei Modi programmatisch gesteuert werden, den High und Low Level Modi.

Die Funktionen ROS, SLAM, OpenCV und die Nutzung der Ultraschallsensoren werden in der Arbeit „Entwicklung eines Intelligenten lidarbasierten 3D Navigationssystems für den Unitree GO1“ aufgegriffen und dokumentiert[52].

[52]Kemnitzer, s. Anm. 1.

# 5 Funktionserweiterungen

Folgendes Kapitel beschäftigt sich mit diversen Erweiterungen, die besonders im langfristigen Umgang mit dem Roboter von Nutzen sein sollen. Hierunter fallen Punkte wie die stetige Verbindung mit dem Internet, eine mögliche Verbindung über weite Entfernungen inklusive der Fernsteuerung des Roboters sowie der Übertragung der Kamerabilder zu diversen Zwecken, beispielsweise des Steuerns des Roboters aus der Entfernung.

## 5.1 Konnektivität

In Kapitel 3.3 wurde bereits erläutert, wie das interne Netzwerk des GO1 und all seiner Rechenkomponenten aufgebaut ist. In Kapitel 3.2 wurde gezeigt, wie sich auf die einzelnen Rechenkomponenten verbunden werden kann. Im folgenden Kapitel sollen weitere Möglichkeiten erarbeitet werden, um den GO1 über diverse Netzwerke zugänglich zu machen. Alle folgenden Erweiterungen der Konnektivität wurden getestet und dementsprechend dokumentiert.

### Wifi

Der Raspberry Pi des GO1 besitzt drei WLAN-Interfaces, von denen lediglich eines für das Spannen des eigenen Access-Points verwendet wird. Das freie Interface `wlan2` kann verwendet werden, um den Roboter innerhalb eines bereits bestehenden Netzwerkes kabellos mit dem Internet zu verbinden. Hierfür muss die `configNetwork` Autostart-Funktion auf dem Raspberry Pi angepasst werden. Diese ist, wie in Kapitel 4.2.1 dokumentiert, im Pfad `/home/pi/Unitree/autostart/configNetwork` zu finden. Im Skript `configNetwork.sh` werden zu Beginn alle Interfaces abgeschaltet, worauf nur die Interfaces `eth0` und `wlan1` wieder aktiviert und konfiguriert werden. Das Interface `wlan2` muss deshalb zunächst wieder aktiviert werden.

```
pi@raspberrypi:~ $ sudo ifconfig wlan2 up
```

### Verbindung durch den wpa_supplicant

Das installierte Betriebssystem des Pis, Debian (Raspbian) 10, verwendet zur Konfiguration der kabellosen Netzwerkverbindungen das Paket `wpa_supplicant`. Um die sensiblen Netzwerkkonfigurationen nicht zu korrumpieren, wird hierfür eine neue Konfigurationsdatei zum Verbinden mit einem Access-Point erstellt. Hierfür wird im Verzeichnis `/etc/wpa_supplicant/` ein neues Verzeichnis für eigene Verbindungen angelegt.

```
pi@raspberrypi:~ $ mkdir /etc/wpa_supplicant/config.d
```

Danach werden die nötigen Konfigurationen in eine Datei names `wlan2.conf` geschrieben. Wichtig ist hierbei das Setzen der korrekten Landeskürzung, um dem Betriebssystem klarzustellen, welche rechtlichen Grundlagen für die kabellose Verbindung über WLAN eingehalten werden müssen.

```
pi@raspberrypi:~ $ echo "\
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=DE \n" \
> wlan2.conf
```

Diese Länderkürzung muss auch noch global im System hinterlegt werden.

```
iw region set DE
```

Daraufhin können die Zugangsdaten zum Access-Point verschlüsselt an die oben erstellte `wlan2.conf`-Datei angehängt werden.

```
pi@raspberrypi:~ $ wpa_passphrase Beispiel-SSID passwort >> /etc/
   ↪ wpa_supplicant/conf.d/wlan2.conf
```

Die Datei sollte nun wie folgt aussehen:

```
1 ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
2 update_config=1
3 country=DE
4 network={
5   ssid="Beispiel-SSID"
6   #psk="passwort"
7   psk=6d2c8604ecb1f4825d410b859ed0f
       ↪ a19621bea7ffa0b1c9b8bdda995c7135c20
8 }
```

Zeile 6 – `psk` – sollte hierbei zur Geheimhaltung des Passworts entfernt werden. Nun kann das Interface mit der erstellten Konfiguration mit dem gewünschten Access-Point verbunden werden.

```
pi@raspberrypi:~ $ wpa_supplicant -B -i wlan2 -c /etc/wpa_supplicant/
   ↪ conf.d/wlan2.conf
```

Sollte bereits eine laufende Konfiguration für das Interface vorhanden sein, so muss diese zuerst mit folgendem Befehl entfernt werden.

```
pi@raspberrypi:~ $ sudo rm /var/run/wpa_supplicant/wlan2
```

Der Raspberry Pi des Roboters ist nun mit dem Access Point verbunden. Ist dieses ebenfalls am Internet angebunden, so hat der Raspberry Pi eine funktionierende Netzwerkverbindung, solange er sich in Funknähe des Access-Points befindet.

**Automatisierung der Verbindung**

Zur automatischen Verbindung des Pis mit dem Internet nach Systemstart kann die Autostart-Funktion `configNetwork` angepasst werden. Hierfür sollte ein weiteres Skript erstellt werden, welches die neue Logik enthält. Dieses kann wie folgt aussehen:

```
1  #!/bin/bash
2  # /home/pi/Unitree/autostart/configNetwork/connectWlan2.sh
3
4  sleep 1
5  echo "[connectWlan2] Start initializing wlan2 interface"
      ↪ $toStartlog
6  sudo ifconfig wlan2 up
7  sudo rm /var/run/wpa_supplicant/wlan2
8  echo "[connectWlan2] Connecting wlan2 interface" $toStartlog
9  sudo wpa_supplicant -B -i wlan2 -c /etc/wpa_supplicant/config.d/
      ↪ wpa_wlan2.conf
10 echo "[connectWlan2] Done connecting wlan2 interface via
      ↪ wpa_supplicant" $toStartlog
11 sleep 3
```

Das Skript kann jetzt am Ende des bereits bestehenden Skripts `configNetwork.sh` aufgerufen werden.

```
42 sudo ./connectWifi2.sh&
```

Der Raspberry Pi verbindet sich jetzt während dem Systemstart mit dem konfigurierten Netzwerk.

**Mobilfunk**

Die Modelle GO1 *MAX* und *EDU* haben laut Herstellerwerbung ein 4G/LTE Modem verbaut, welches genutzt werden kann, um den Roboter dauerhaft mit dem Internet zu verbinden oder ihn aus der Ferne zu steuern. Die Dokumentation des Herstellers gibt jedoch keine Hinweise

zur Konfiguration oder Nutzung des Modems. Ein Blick auf die verbundenen USB Geräte gibt einen Hinweis auf das genaue Modem.

```
pi@raspberrypi:~ $ lsusb | grep Quectel
Bus 001 Device 003: ID 2c7c:0125 Quectel Wireless Solutions Co., Ltd.
   ↪  EC25 LTE modem
```

Verbaut ist ein *Quectel EC25 LTE Modem*, welches in Linuxkreisen häufig genutzt wird und deshalb auch genutzt werden kann, ohne neue Treiber nachinstallieren zu müssen.

**Vorbereitung** Zur Verbindung des Modems mit dem Mobilfunknetz ist eine SIM-Karte zwingend vorausgesetzt. Diese kann in den dafür vorgesehenen Slot auf der Oberseite des Rumpfes eingesteckt werden. Genaueres zur Positionierung des Slots kann in Kapitel 3.1.4 nachgelesen werden. Der Slot ist für Micro-SIM-Karten der Größe 12 mm auf 15 mm geeignet.

Der Großteil der gängigen SIM-Karten benötigt zur Entsperrung eine PIN (Personal Identification Number). Dieser sollte für die vereinfachte Nutzung im Roboter deaktiviert werden. Da diese Funktion für alle Kartenanbieter variieren kann, sollte hierfür die Dokumentation des Anbieters konsultiert werden. Oftmals kann die PIN Funktion auch durch Endgeräte wie Smartphones deaktiviert werden. Auch hier unterscheiden sich die Vorgehensweisen jedoch stark, weshalb nicht weiter auf die Deaktivierung der PIN eingegangen wird. Nach Deaktivierung kann die SIM-Karte in den Slot eingefügt und der Roboter eingeschaltet werden.

**Analyse des Modems**

LTE Modems können in der Regel in verschiedenen Modi betrieben werden. Je nach Modus unterscheiden sich auch die verwendeten Treiber des Gerätes. Um mit der Konfiguration des Modems fortzufahren, muss zuerst der Modus bestimmt werden, in dem das Gerät am Pi betrieben wird. Hierfür kann die `lsusb` Device-Nummer genutzt werden, die folgendermaßen bestimmt werden kann.

```
pi@raspberrypi:~ $ lsusb | grep Quectel
Bus 001 Device 003: ID 2c7c:0125 Quectel Wireless Solutions Co., Ltd.
   ↪  EC25 LTE modem
pi@raspberrypi:~ $ lsusb -t | grep "Dev 3"
        |__ Port 3: Dev 3, If 0, Class=Vendor Specific Class, Driver=
           ↪ option, 480M
        |__ Port 3: Dev 3, If 1, Class=Vendor Specific Class, Driver=
           ↪ option, 480M
```

```
        |__ Port 3: Dev 3, If 2, Class=Vendor Specific Class, Driver=
           ↪ option, 480M
        |__ Port 3: Dev 3, If 3, Class=Vendor Specific Class, Driver=
           ↪ option, 480M
        |__ Port 3: Dev 3, If 4, Class=Vendor Specific Class, Driver=
           ↪ qmi_wwan, 480M
        |__ Port 3: Dev 3, If 5, Class=Audio, Driver=snd-usb-audio,
           ↪ 480M
        |__ Port 3: Dev 3, If 6, Class=Audio, Driver=snd-usb-audio,
           ↪ 480M
        |__ Port 3: Dev 3, If 7, Class=Audio, Driver=snd-usb-audio,
           ↪ 480M
```

Zu erkennen ist, dass das Gerät mit dem Treiber `qmi_wwan` betrieben wird. In diesem Modus wird das Modem als `wwan`-Interface gelistet.

```
pi@raspberrypi:~ $ ifconfig -a | grep wwan
wwan0: flags=4098<BROADCAST,MULTICAST>  mtu 1500
```

Für die vereinfachte Nutzung des Modems im `qmi` Modus wird das Paket `libqmi-utils` nachinstalliert. Hierfür muss der Raspberry Pi mit dem Internet verbunden sein[53].

```
pi@raspberrypi:~ $ sudo apt update && sudo apt install libqmi-utils
```

Für die Nutzung des Werkzeugs `qmicli` des Pakets `libqmi-utils` muss zuerst noch der Gerätepfad im Dateisystem bestimmt werden. Hierfür können die Kernel-Nachrichten zurate gezogen werden.

```
pi@raspberrypi:~ $ dmesg | grep qmi
[    7.606540] qmi_wwan 1-1.3:1.4: cdc-wdm0: USB WDM device
[    7.682555] qmi_wwan 1-1.3:1.4 wwan0: register 'qmi_wwan' at usb-
   ↪ fe980000.usb-1.3, WWAN/QMI device, 8e:c2:07:55:9c:c3
[    7.686460] usbcore: registered new interface driver qmi_wwan
```

Der Gerätename ist in der ersten Zeile der Ausgabe als `cdc-wdm0` angegeben. Das kann über eine Ausgabe der möglichen Geräte im Verzeichnis `/dev/` bestätigt werden.

```
pi@raspberrypi:~ $ ls /dev/cdc-*
/dev/cdc-wdm0
```

### Vorbereitung des Modems

[53] Siehe Kapitel 5.1

Nun müssen einige Vorbereitungen getroffen werden, um das Modem mit dem Mobilfunknetz verbinden zu können. Zuerst muss geprüft werden, ob das Modem *online* ist.

```
pi@raspberrypi:~ $ sudo qmicli -d /dev/cdc-wdm0 --dms-get-operating-
   ↪ mode
error: couldn't create client for the 'dms' service: CID allocation
   ↪ failed in the CTL client: Transaction timed out
```

Sollte wie hier die Fehlermeldung auftreten, so ist das Gerät durch einen anderen Prozess oder Service blockiert. Ein gängiger Service, der auf das Modem zugreift, ist oft der `ModemManager.service`. Folgender Befehl prüft, ob dieser aktiv ist.

```
pi@raspberrypi:~ $ systemctl | grep Modem
ModemManager.service   loaded active running   Modem Manager
```

Folgender Befehl stoppt diesen. Danach kann das Modem noch einmal per `qmicli` abgefragt werden.

```
pi@raspberrypi:~ $ sudo systemctl stop ModemManager
pi@raspberrypi:~ $ sudo qmicli -d /dev/cdc-wdm0 --dms-get-operating-
   ↪ mode
[/dev/cdc-wdm0] Operating mode retrieved:
        Mode: 'online'
        HW restricted: 'no'
```

Sollte der `qmicli` Befehl immer noch fehlschlagen, so muss geprüft werden, ob ein weiterer Prozess desselben Pakets das Modem blockiert. Dies kann über die Abfrage `ps aux | grep qmi` erledigt werden. Ein möglicher blockierender Prozess muss dann über seine Prozess-ID und den Befehl `pkill` beendet werden. Ist der angezeigte `Mode` des `qmicli` Befehls nicht `online`, so muss dieser Wert manuell gesetzt werden.

```
pi@raspberrypi:~ $ sudo qmicli -d /dev/cdc-wdm0 --dms-set-operating-
   ↪ mode='online'
[/dev/cdc-wdm0] Operating mode set successfully
```

**Verbindung mit Mobilfunknetz**

Als Nächstes kann das Modem in das Mobilfunknetz eingewählt werden. Hierfür kann folgender Befehl genutzt werden.

```
pi@raspberrypi:~ $ sudo qmicli \
    -p \ # Request to use the 'qmi-proxy' proxy
    -d /dev/cdc-wdm0 \ # Device Path
```

```
    --device-open-net='net-raw-ip|net-no-qos-header' \ # Open device
       ↪ with specific link protocol and QoS flags
    --wds-start-network="apn='internet',ip-type=4" \ # IPv4 und APN
    --client-no-release-cid # Do not release the CID when exiting
[/dev/cdc-wdm0] Network started
Packet data handle: '2269312560'
[/dev/cdc-wdm0] Client ID not released:
Service: 'wds'
CID: '17'
```

Als Wert für den APN (Access Point Name) muss hier beim Anbieter direkt Auskunft eingeholt werden, da dies je nach Kartenart und Netzwerk unterschiedlich sein kann. Im Test war die Angabe des APN nicht relevant, dies kann jedoch eine mögliche Fehlerquelle sein. Ein weiterer möglicher Fehler kann sein, dass das Modem nicht für das Protokoll `raw-ip` eingestellt wurde. In diesem Fall muss das Netzwerk-Interface gestoppt werden, das Modem konfiguriert und daraufhin wieder aktiviert werden. Anschließend kann der `operating-mode` nochmals geprüft werden.

```
pi@raspberrypi:~ $ sudo ip link set wwan0 down
pi@raspberrypi:~ $ echo 'Y' | sudo tee /sys/class/net/wwan0/qmi/
   ↪ raw_ip
Y
pi@raspberrypi:~ $ sudo ip link set wwan0 up
pi@raspberrypi:~ $ sudo qmicli -d /dev/cdc-wdm0 --wda-get-data-format
[/dev/cdc-wdm0] Successfully got data format
                    QoS flow header: no
               Link layer protocol: 'raw-ip'
    [...]
```

Die Ausgabe zeigt ebenfalls, dass kein `QoS`-Header gesetzt werden sollte. Der Befehl zu Verbindung mit dem Netzwerk kann nun erneut ausgeführt werden.

Als Letztes muss dem Modem im Mobilfunknetz noch eine IP Adresse zugewiesen werden. Sobald das Modem im Mobilfunknetz eingewählt ist, kann es über DHCP eine IP Adresse zugewiesen bekommen. Hierfür wird ein weiteres Paket installiert, das eine Konfiguration sucht und, falls keine gefunden wird, eine Anfrage im Netz zur Zuweisung einer IP stellt.

```
pi@raspberrypi:~ $ sudo apt install udhcpc
pi@raspberrypi:~ $ sudo udhcpc -q -f -i wwan0
udhcpc: started, v1.30.1
No resolv.conf for interface wwan0.udhcpc
udhcpc: sending discover
udhcpc: sending select for 10.114.75.205
```

```
udhcpc: lease of 10.114.75.205 obtained, lease time 7200
```

Das Modem ist jetzt mit dem Mobilfunknetz und somit mit dem Internet verbunden.

**Automatische Verbindung**

Mit der Konfiguration des Modems ist der Raspberry Pi bis zum nächsten Systemstart mit dem Internet verbunden. Startet man den Pi jedoch neu, so muss die Prozedur seit Anfang des Kapitels noch einmal durchführen. Zur vereinfachten automatisierten Verbindung mit dem Mobilfunknetz kann die in Kapitel 4.2.1 beschriebene Autostart-Funktion verwendet werden. Genauer kann hier die `configNetwork` Funktion erweitert werden, wie bereits in Kapitel 5.1 dokumentiert. Hierfür muss lediglich ein weiteres Skript am Ende der `configNetwork.sh` Datei aufgerufen werden.

```
43  sudo ./connectWwan0.sh&
```

Der Inhalt der Datei spiegelt die Verbindung des Modems mit dem Netzwerk und das Abgreifen der IP über DHCP wieder und sieht folgendermaßen aus.

```
1  #!/bin/bash
2  # /home/pi/Unitree/autostart/configNetwork/connectWwan0.sh
3
4  sleep 1
5  echo "[connectWwan0] Preparing environment for wwan0 interface"
      ↪ $toStartlog
6  sudo systemctl stop ModemManager
7  echo "[connectWwan0] Start initializing wwan0 interface"
      ↪ $toStartlog
8  sudo ifconfig wwan2 up
9  sudo qmicli -p -d /dev/cdc-wdm0 --device-open-net='net-raw-ip|net
      ↪ -no-qos-header' --wds-start-network="apn='internet',ip-type
      ↪ =4" --client-no-release-cid
10 echo "[connectWwan0] Getting IP for wwan0 interface" $toStartlog
11 sudo udhcpc -q -f -i wwan0
12 echo "[connectWwan0] Connected wwan0 to internet" $toStartlog
13 sleep 3
```

Der Roboter verbindet sich nun bei Systemstart per Modem mit dem Mobilfunknetz des Anbieters, vorausgesetzt, es befindet sich eine gültige und entsperrte SIM-Karte im Slot auf der Oberseite des Rumpfes.

**Ausblick**

Die Verbindung kann in Zukunft genutzt werden, um den Roboter aus weiter Entfernung steuern zu können. Hierfür kann ein sogenanntes VPN (Virtual Private Network) verwendet werden. Der Raspberry Pi des Roboters müsste hierfür als *Client* im VPN registriert werden. Verbindet man sich nun mit einem weiteren Client mit dem VPN, so sind beide Geräte im gleichen Netzwerk, unabhängig davon, über welches Netzwerk sie mit dem Internet verbunden sind.

## 5.2 BMS

Nutzt man den GO1 im Batterie-Betrieb, so sollte man stets darauf achten, dass die Ladung des Akkus nicht unter 3 % fällt, da die Motoren bei diesem Ladestand zur Sicherheit abgeschaltet werden. Die Umsetzung des Entwicklerteams der Firma Unitree hat in diese Sicherheitsfunktion jedoch wenig Arbeit investiert. So ist die einzige ständig sichtbare Anzeige der Akkuladung außen am Akku angebracht und nur von einer Seite einsehbar. Zudem kann die Anzeige die Ladung nur in acht Schritten anzeigen und ist somit nicht genau genug. Auch das abrupte Abschalten der Motoren ist potenziell schädlich für den Roboter, da dieser nicht zuerst in einen liegenden Zustand gebracht wird, was zur Folge haben kann, dass er in jeder erdenkbaren Position zusammensackt. Die einzelnen Glieder der vier Beine und der Rumpf fallen dann ungeschützt und prallen auf dem Boden auf.

Um diesem Mangel entgegenzusetzen und den Roboter somit resilienter gegen vermeidbare Schäden zu machen, wird in diesem Kapitel ein Batterie-Monitoring-System mit aktiver Warnung und vorsichtiger Motorabschaltung implementiert. Ziel ist es, bei einer niedrigen Akkuladung von 5 % eine Warnung über die LEDs des Kopfes darzustellen und den GO1 dann sanft zu Boden zu lassen, bevor die eingebauten Mechanismen bei 3 % Ladung greifen und die Motoren abschalten. Hierfür werden die bereits verbauten Funktionen MQTT und Autostart verwendet. Dokumentation hierfür ist im Kapitel 4.2 zu finden.

### Entwurf

Abbildung 28 zeigt das Sequenzdiagramm zur Darstellung des Batterie-Monitorings.
Die beteiligten Komponenten des Roboters sind die Autostart Funktion, die Softwarekomponenten BMS-Sniffer, LED-Control und Movement-Control, der MQTT-Broker, die LEDs des Kopfes, die MCU und die zwölf Motoren. Tatsächlich interagiert wird lediglich mit der Autostart-Funktion und dem MQTT-Broker. Die drei Software-Komponenten müssen noch entwickelt werden.

Das Autostart-Modul auf dem Raspberry Pi soll um die neue Funktionalität `bmsMonitoring` erweitert werden. Hierfür wird ein Ordner mit Skript angelegt, das alle weiteren Softwarekom-

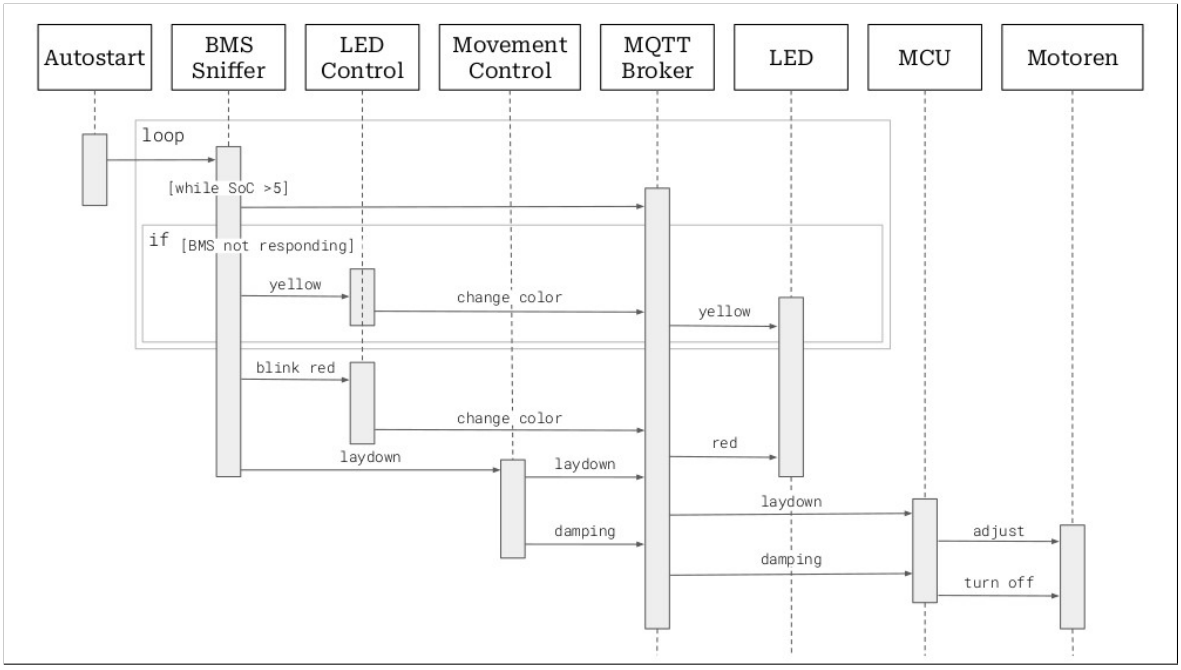


Abbildung 28: Sequenzdiagramm der Batterie-Monitoring Komponenten

ponenten auslöst. Das Modul BMS-Sniffer soll dauerhaft die BMS Daten über den MQTT-Broker abfragen. Ist das BMS nicht erreichbar, beispielsweise weil der Roboter im Netzbetrieb ohne Akku gestartet wurde, so wird die Komponente LED-Control aufgerufen, welche die LEDs des Roboters dauerhaft gelb leuchten lässt. Sollte das BMS erreichbar sein und die SoC (State of Charge) auf 5 % oder weniger fallen, so wird zuerst das LED-Control-Modul aufgerufen, um die LEDs des GO1 rot blinken zu lassen, worauf das Movement-Control-Modul aufgerufen wird, um den Roboter hinzulegen und danach die Motoren zu entspannen (Damping-State). Sowohl das LED-Control-Modul als auch das Movement-Control-Modul nutzen den MQTT-Broker als Schnittstelle zu den zu steuernden Hardwarekomponenten.

**Umsetzung**

Für die Umsetzung des BMS-Monitors muss zuerst die Autostart-Funktion ergänzt werden. Hierfür muss der Ordner `/home/pi/Unitree/autostart/bmsMonitor` erstellt werden. In diesem wird das Skript `bmsMonitor.sh` angelegt, welches lediglich ein weiteres Skript namens `run.sh` als Hintergrundprozess startet. Dieser Schritt ist nötig, um die Monitoring-Funktion vom Autostart-Prozess zu entkoppeln. Das `run.sh` Skript startet nun alle Loggingaktivitäten und die nötigen Software-Komponenten. Als letzter Schritt muss die neue Funktion noch in die Startup-Liste `/home/pi/Unitree/autostart/.startlist.sh` angehängt werden.

Um die Entwicklung und Anpassung der Autostart-Funktion zu vereinfachen, kann der Quellcode in einem Ordner an anderer Stelle über die Versionsverwaltung *Git* aktuell gehalten werden. Die einzelnen Dateien und Ordner können über ein Installationsskript in die Autostart-Funktion integriert werden. Folgendes Listing zeigt die relevanten Teile des Installationsskriptes. Die vollständigen Dateien sind im Anhang zu finden.

```
33 # Ordner loeschen
34 rm -r "$DIR/bmsMonitor" &> /dev/null
35 # Symbolische Links zum Repository erstellen
36 cp -rs "$SCRIPT_DIR/bmsMonitor" "$DIR"
37 # Skript kopieren um Pfade zu erhalten
38 cp -f "$SCRIPT_DIR/bmsMonitor/bmsMonitor.template.sh" "$DIR/
      ↪ bmsMonitor/bmsMonitor.sh"
39 # Skripte ausfuehrbar machen
40 chmod +x "$DIR/bmsMonitor/bmsMonitor.sh" "$SCRIPT_DIR/bmsMonitor/
      ↪ run.sh"
```

Folgende Übersicht zeigt den Aufbau des Quellcode-Ordners zur Einordnung des Installationsskripts und der weiteren Erklärungen.

```
go1-bmsMonitor
├── bmsMonitor
│   ├── bmsMonitor.sh
│   ├── constants.sh
│   ├── led_control.py
│   ├── requirements.txt
│   ├── run.sh
│   └── sniff_bms.py
└── install.sh
```

Der eigentliche Einstieg des `bmsMonitor` ist die Datei `run.sh`, die als Hintergrundprozess durch den Autostart gestartet wird. Diese installiert vorerst alle benötigten *Python*-Bibliotheken, die in der `requirements.txt` Datei hinterlegt sind. Daraufhin wird das Skript `sniff_bms.py` gestartet, welches den BMS-Monitor enthält.

```
8 python3 -m pip install -r /home/pi/Unitree/autostart/bmsMonitor/
      ↪ requirements.txt
9 python3 /home/pi/Unitree/autostart/bmsMonitor/snif_bms.py
```

Die Zeilen 13 bis 23 des Skripts `sniff_bms.py` zeigen die Dekodierung der binären Message-Payload bei neuen MQTT-Nachrichten und die weitere Auswertung der SoC.

```
13 def on_message(c, userdata, msg):
14     print("message")
```

```
15      [ver0, ver1, status, soc, current, cycle, temp0, temp1,
16       temp2, temp3, cell_v0, cell_v1, cell_v2, cell_v3,
17       cell_v4, cell_v5, cell_v6, cell_v7, cell_v8, cell_v9] \
18          = struct.unpack('BBBBiHBBBBHHHHHHHHHH', msg.payload)
19      if not bms_responding(msg.payload):
20          const_led(c, 10, 10, 0)
21      elif 5 >= soc:
22          alert_led(c, 255, 0, 0)
23          lay_down()
```

Die Methode `struct.unpack(...)` zeigt die in Kapitel 4.2.7 erarbeitete Dekodierung der Message-Payload des Topics `bms/state`. In Zeile 19 wird geprüft, ob das BMS aktiv ist. Ist das BMS inaktiv, so besteht die Message-Payload nur aus binären Nullen. In diesem Fall wird die Funktion `const_led(c,r,g,b)` aus dem Modul LED-Control in der Datei `led_control.py` aufgerufen.

```
14  def const_led(client, r, g, b):
15      client.publish(FACE_LIGHT_TOPIC, struct.pack('BBB', r, g, b),
           ↪  qos=QOS)
16      time.sleep(1)
```

Die Übergabeparameter `r`, `g` und `b` sind die RGB-Werte, die die LEDs am Kopf des Roboters annehmen sollen. Der Parameter `c` ist die MQTT-Client-Instanz, die zur Kommunikation mit dem Broker genutzt wird. Das Format des MQTT-Topics `face_light/color` ist in Kapitel 4.2.5 beschrieben. Die RGB-Werte `r=10`, `g=10` und `b=0` stellen die LEDs auf ein stark gedimmtes Gelb ein.

In Zeile 21 der Datei `sniff_bms.py` prüft den Wert des SoC des Akkus, ist dieser größer als 5, so endet die Funktion und es wird auf die nächste MQTT-Nachricht gewartet. Andernfalls werden die LEDs in der `alert_led()` Funktion der Datei `led_control.py` auf Rot-blinkend konfiguriert.

```
 7  def alert_led(client, r, g, b):
 8      client.publish(FACE_LIGHT_TOPIC, struct.pack('BBB', r, g, b),
           ↪  qos=QOS)
 9      time.sleep(1)
10      client.publish(FACE_LIGHT_TOPIC, struct.pack('BBB', 0, 0, 0),
           ↪  qos=QOS)
11      time.sleep(1)
```

Die Übergabeparameter sind hier die gleichen wie die der Funktion `const_led(c,r,g,b)`. Nur den Wert `r=255` zu setzen hat zur Folge, dass die LEDs in der maximalen Helligkeit

Rot leuchten. Die Funktion `lay_down()` des Moduls Movement-Control ist der Einfachheit wegen in der Datei `sniff_bms.py` formuliert.

```
31 def lay_down():
32     client.publish(CONTROLLER_ACTION_TOPIC, payload="standDown")
33     client.publish(CONTROLLER_ACTION_TOPIC, payload="damping")
```

Die Funktion sendet lediglich zwei Befehle an das MQTT-Topic `controller/action`. Die Aktion `standDown` bewirkt, dass der Roboter alle vier Beine anwinkelt und somit den Rumpf nah an den Boden bewegt. Die Aktion `damping` bewirkt, dass der Roboter die Spannung aus den Gelenken nimmt und die Motoren deaktiviert. Das hat zur Folge, dass der Roboter leicht zusammensackt, weshalb er vorher in einen liegenden Zustand gebracht werden sollte. Die gesamte Aktion der LED Konfiguration und des Hinlegens ist kurz genug, dass die SoC des Akkus nicht auf 3 % fällt. Sobald dies passiert werden die Motoren abgeschaltet und können nicht mehr verwendet werden.

Die Konstanten der gesamten Python-Umgebung sind in der Datei `constants.py` hinterlegt.

```
1 HOSTNAME = "192.168.12.1"
2 PORT = 1883
3 TIMEOUT = 60
4 FACE_LIGHT_TOPIC = "face_light/color"
5 BMS_STATE_TOPIC = "bms/state"
6 CONTROLLER_ACTION_TOPIC = "controller/action"
7 QOS = 2  # exactly once
```

Zur finalen Installation der Erweiterung kann wie folgt vorgegangen werden.

```
pi@raspberrypi:~ $ mkdir ~/Extensions/ && cd ~/Extensions
pi@raspberrypi:~/Extensions $ git clone <Repository Link>
pi@raspberrypi:~/Extensions $ cd go1-bmsMonitor/
pi@raspberrypi:~/Extensions/go1-bmsMonitor $ sh install.sh
```

Das Installationsskript erstellt einen Link im Autostart-Ordner zum Verzeichnis des geklonten Git-Repositories und konfiguriert die Skripte, sodass sie ausführbar sind. Bei einem erneuten Systemstart des Raspberry Pi wird die Funktion am Ende des Autostart-Prozess' gestartet. Der gesamte Quellcode kann in dem zur Versionsverwaltung angelegten Git-Repository genauer untersucht werden[54].

[54] Noah Lehmann. *go1-bmsMonitor*. 1. Juli 2023. URL: `https://github.com/noahlehmann/go1-bmsMonitor` (besucht am 24. 08. 2023).

## 5.3 Remote Video Streaming

Wie in Kapitel 4.2.6 gezeigt, kann der GO1 die Bilder der fünf verbauten Kameras über ein Netzwerk streamen. Dieses Kapitel baut auf den Erkenntnissen des Kapitels 4.2.6 auf und erweitert die dort gezeigten Funktionen um den Streaming-Server und eine Darstellung des gestreamten Kamerabildes.

### Vorbereitung

Zur Vorbereitung des Remote Videostreaming muss zuerst ein Streaming-Server vorbereitet werden, der über eines der am Roboter verbundenen Netzwerke erreichbar ist. In diesem Kapitel wird zum einfachen Hosting des Servers die Virtualisierungsumgebung *Docker* verwendet. Ein vorhandenes Grundwissen hierüber wird vorausgesetzt, jedoch ist die Anwendung sehr einfach gehalten. Docker hat hier den Vorteil, dass das gezeigte Beispiel auf den meisten Rechnern nachstellbar ist. Der eigentlich verwendete Server ist somit nicht relevant, insofern Docker auf ihm installiert werden kann.

Die Open-Source-Bibliothekssammlung *bluenvironment* enthält eine auf der Programmiersprache *GO* besierte Implementierung eines Medienservers mit Unterstützung diverser Videoprotokolle wie beispielsweise RTSP und WebRTC (Web Real Time Communications). Diese werden auch im folgenden Beispiel verwendet. Die Implementierung des Medienservers ist zur Nutzung als Docker-Image verfügbar, welches über folgenden Befehl genutzt werden kann.

```
docker run --rm -it \
    -e MTX_PROTOCOLS=tcp \
    -p 8554:8554 \
    -p 8889:8889 \
```

Führt man diesen Befehl auf dem zu nutzenden externen Server aus, so startet das Image einen Container mit integriertem Medienserver, welcher für das Streaming über das Protokoll RTSP den Port 8554 und für das Protokoll WebRTC den Port 8889 öffnet und auf dem Host verfügbar macht. Für das Empfangen der Kamera-Streams des GO1 wird in diesem Fall RTSP verwendet, für das Senden eines Streams hingegen WebRTC. Sobald der Container gestartet ist, wartet der Medienserver auf Verbindungen.

### Streaming

Das Streaming der Kamerabilder vom Roboter auf den Server verläuft analog zu der Vorgehensweise in Kapitel 4.2.6.

```
ffmpeg -nostdin \                    # Keine Interaktion
```

```
    -f video4linux2 \                # Input Format
    -i /dev/video1 \                 # Input-URL
    -vcodec libx264 \                # Video Kodierung (h264)
    -preset:v ultrafast \            # Kodierungsgeschwindigkeit
    -tune zerolatency \              # Keine H264 B-Frames
    -framerate 5 \                   # Framerate
    -f rtsp \                        # Output Format
    rtsp://<ip:8554|port>/<stream> # Output File/URL
```

Hier ist nun die Server IP aus dem vorigen Paragrafen in der letzten Zeile des Befehls einzusetzen, gefolgt vom Port 8554 für das RTSP. Als Stream Name kann ein selbst gewählter Begriff genutzt werden. Streamt man mehrere Kameras gleichzeitig, so bietet es sich an, die Namen nach Ausrichtung der Kamera zu benennen. Für den Stream der nach vorne gerichteten Kameras des GO1 bietet sich beispielsweise folgende Server-URL (Uniform Resource Locator) an:

```
rtsp://192.168.123.70:8554/head
```

Nach erfolgreicher Verbindung wird auf dem Server folgende Meldung ausgegeben.

```
2023/08/25 10:22:02 INF [RTSP] [conn 192.168.123.13:35428] opened
2023/08/25 10:22:02 INF [RTSP] [session f35fae28] created by
   ↪ 192.168.123.13:35428
2023/08/25 10:22:02 INF [RTSP] [session f35fae28] is publishing to
   ↪ path 'head', with TCP, 1 track (H264)
```

Auf dem Nano im Kopf des Roboters wird folgende Meldung ausgegeben.

```
Output #0, rtsp, to 'rtsp://192.168.123.51:8554/head':
  Metadata:
    encoder         : Lavf57.83.100
    Stream #0:0: Video: h264 (libx264), yuvj422p(pc), 928x400, q
       ↪ =-1--1, 100 fps, 90k tbn, 100 tbc
    Metadata:
      encoder         : Lavc57.107.100 libx264
    Side data:
      cpb: bitrate max/min/avg: 0/0/0 buffer size: 0 vbv_delay: -1
frame=15476 fps=101 q=24.0 size=N/A time=00:02:34.75 bitrate=N/A dup
   ↪ =11652 drop=0 speed=1.01x
```

Die Kamerabilder des Kopfes werden erfolgreich übertragen. Diese Vorgehensweise kann für alle anderen Kameras wiederholt werden, es muss lediglich ein neuer Stream unter einem anderen Namen an den Server gesendet werden.

### Anzeigen des Streams

Das Protokoll WebRTC hat den Vorteil, dass es in den meisten modernen Browsern bereits unterstützt wird und keinerlei Konfiguration bedarf. Zum Anzeigen der Kamerabilder kann einfach die Server URL inklusive des Ports und des Stream-Namens in die Adressleiste des Browsers eingegeben werden. Das Kamerabild im Browser wird wie in Abbildung 29 gezeigt dargestellt.

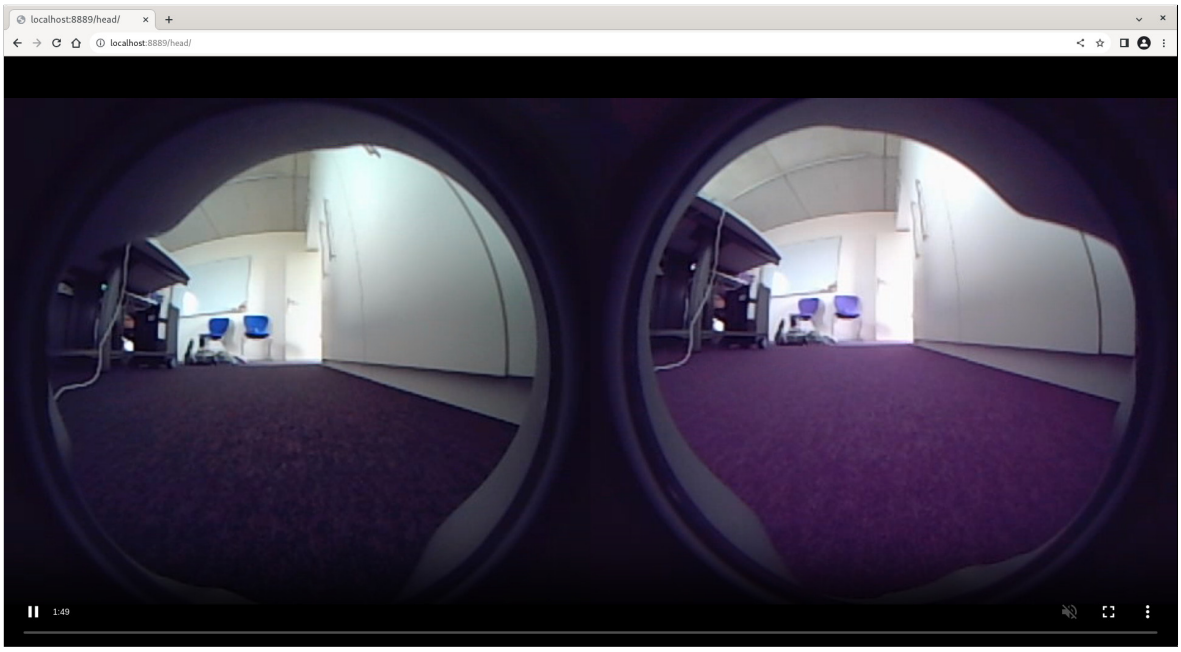


Abbildung 29: Das Kamerabild des Kopfes in der Browseransicht

Getestet wurde für dieses Beispiel auf dem Browser *Chromium* in der Version `116.0.5845.96 (Official Build) built on Debian 12.1, running on Debian 12.1 (64-bit)`.

## 5.4 Fernsteuerung

Der letzte Schritt zur vollständigen Steuerung des GO1 aus der Ferne ist die Fernsteuerung der Bewegungen des Roboters. Hierfür muss geprüft werden, inwiefern sich die Befehle der Fernbedienung aus Kapitel 5.4 über ein Netzwerk replizieren lassen. Die Ergebnisse aus den Kapiteln 4.2.5 und 4.2.7 lassen vermuten, dass sich die Bewegung über MQTT steuern lässt.

### Analyse der Webseite

Eine Inspektion des Quellcodes der Webseite zeigt folgenden Ausschnitt der Datei `webpack:///src/views/Vision.vue`.

```
172  const handleStickData = (evt: {
173      lx: number;
```

```
174      ly: number;
175      rx: number;
176      ry: number;
177  }) => {
178      const floats = new Float32Array(4);
179      floats[0] = evt.lx;
180      floats[1] = evt.rx;
181      floats[2] = evt.ry;
182      floats[3] = evt.ly;
183      mqtt.publish("controller/stick", floats.buffer, { qos: 0 });
184  };
```

Listing 15: Ausschnitt der Datei `Vision.vue`

Die Ausgabe der Zeilen 173 bis 176 lässt vermuten, dass die beiden Joysticks der Fernbedienung emuliert werden, in dem $x$ und $y$ Werte angegeben werden, um die Position des Sticks relativ zum Nullpunkt – der Mitte – zu errechnen. Über das Bewegen des linken Sticks der auf der Webseite angezeigten Steuerungen auf ein oberes Maximum lässt sich über das Einhängen eines Debuggers folgender Wert der vier Koordinaten ausgeben.

```
evt = {lx: 0,ly: 1,rx: 0,ry: 0}
```

Die möglichen Float-Werte der Koordinaten können also zwischen $-1$ und 1 liegen. Die Dokumentation der Implementierung der JavaScript-Klasse `Float32Array` zeigt, dass das Array `floats` in Zeile 178 vier Floats mit der Länge 32 bit speichert. Die Funktion `floats.buffer` in Zeile 183 schreibt die vier Werte in binärer Representation in den Message Payload des MQTT Topics `controller/stick`. Um nun die Werte `lx=0`, `ly=1`, `rx=0` und `ry=0` manuell an den MQTT Broker des GO1 zu senden, müssen die Werte in ihre binäre Form und die richtige Reihenfolge gebracht werden. Die Zeilen 179 bis 182 zeigen, dass die korrekte Reihenfolge der binären Koordinaten der Steuerung `lx →rx →ry →ly` ist.

Zum Test der Ergebnisse kann ein Befehl mit dem Paket `mosquitto_pub` an den Broker gesendet werden. Hierfür ist die hexadezimale Darstellung der Werte hilfreich. Die hexadezimale Representation des binären Wertes eines `float32` mit dem Wert 1 ist Folgende.

```
Binaer   = 00111111100000000000000000000000
Hex      = 3F800000
```

Der Befehl lautet wie folgt.

```
echo -ne "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x3f\x80\
    ↪ x00\x00" | mosquitto_pub -h 192.168.123.161 -t controller/stick
    ↪  -s
```

Da man die Stellung der zwei Sticks dauerhaft konfiguriert, sollte man die Werte zum Test schnellstmöglich wieder auf null zurücksetzen, da der Roboter im Beispiel oben sonst dauerhaft vorwärtsläuft.

**Aufbau der Fernsteuerung**

Da die Komplexität einer gesamten Anwendung für dieses Beispiel über den Rahmen der Arbeit hinaus geht, soll hier nur kurz ein Diagramm der nötigen Komponenten für eine minimale Anwendung zur Fernsteuerung des Roboters dargestellt werden.

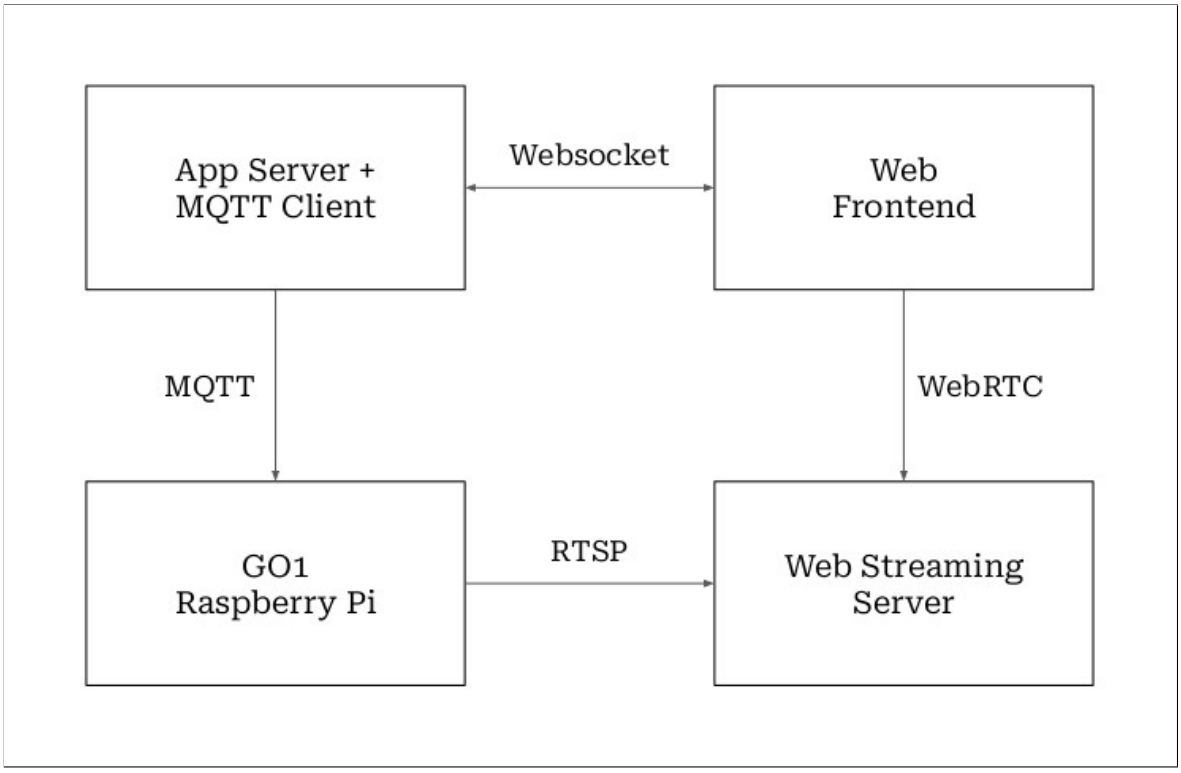


Abbildung 30: Diagramm der Komponenten einer beispielhaften Fernsteuerungssoftware

Als Grundlage für die Anwendung dient ein Appserver, welcher sowohl eine Websocket mit MQTT-Anbindung bereitstellt, als auch die Webseite hostet, welche die Steuerung und einen optionalen Videostream der Kamerabilder des Roboters enthält. Die Webseite sendet per Websocket die Nutzereingaben an den Server, wobei dieser die Eingaben in MQTT-Messages umwandelt und diese an den Broker im Raspberry Pi des GO1 sendet.

**Integration des Videostreams**

HTML5 (Hypertext Markup Language V.5) unterstützt in den meisten modernen Browsern nativ die Übertragung von WebRTC Streams. Dies wurde bereits in Kapitel 5.3 gezeigt. Auch das native HTML5 Tag `<video/>` kann diese Streams einbetten. Hierfür muss lediglich das Attribut `src` mit der URL des Streams auf dem Streaming-Server gesetzt werden.

Wie in Abbildung 30 gezeigt, kann nun die Webseite mit dem eingebetteten Stream über die Verwendung des nativen HTML5 Tags `<video/>` direkt auf den Streaming-Server zugreifen und somit nicht nur die Steuerung, sondern auch die Kamerabilder anzeigen. Für die Funktion der Fernsteuerung und der integrierten Videoübertragung müssen alle in Abbildung 30 gezeigten Komponenten im selben Netzwerk sein oder über mehrere Netzwerke hinweg miteinander kommunizieren können. Die Vorbereitungen aus den Kapiteln Wifi und Mobilfunk können hierfür hilfreich sein.

# 6 Fazit

Dieses Kapitel dient als Abschluss und Zusammenfassung der erarbeiteten Erkenntnisse dieser Arbeit. Zu Beginn werden die Ergebnisse dieser Arbeit zusammengefasst und bewertet. Danach soll die Vorgehensweise und das Arbeitsmittel in Form des GO1 bewertet werden. In diesem Zug wird auch ein Fazit zur Integration dessen in ein Hochschulökosystem geschlossen. Abschließend soll ein Ausblick gestaltet werden, der die potenziell nächsten Schritte der Arbeit am Roboter in Aussicht stellt.

## 6.1 Rückblick

Diese Arbeit hatte besonders das Ziel, als Grundlage für die Arbeit mit dem GO1 angesehen zu werden, um ihn dann in einem Hochschulumfeld in der Forschung, Lehre und anderen Gebieten einsetzen zu können. Hierfür wurde der Roboter in ein vereinfachtes Feld der Serviceroboter eingeordnet, wobei ebenfalls erläutert wurde, welche weiteren Merkmale zutreffend sind. Anschließend wurde der Aufbau und Umfang des GO1 in der Edu Version genau dokumentiert, sodass dieser Teil als Referenz für die Einarbeitung in den Roboter verwendet werden kann. Im Architekturbereich wurden die internen Komponenten genau analysiert. Hierzu gehören auch die genauen Kenndaten der verbauten Recheneinheiten, deren Funktionen und besonders die Kommunikation der Einheiten untereinander.

Anhand der gewonnenen Erkenntnisse über den Roboter konnte mit der Analyse der Funktionen begonnen werden. Nach Dokumentation des Lieferumfangs und der diversen Möglichkeiten der Inbetriebnahme wurden diverse Grundfunktionen sowie vereinzelt auch fortgeschrittene Funktionen des GO1 dokumentiert und bewertet. Viele dieser Funktionen werden zwar vom Hersteller beworben, leider jedoch nicht dokumentiert, weshalb die Arbeit an dieser Stelle als detaillierte und besonders umfangreiche alternative Dokumentation angesehen werden kann. Anzumerken ist hier der hohe Detailgrad der Untersuchungen bei Funktionen, die nicht oder nur unzureichend dokumentiert oder nicht vollständig funktionsfähig sind. Da nicht alle Funktionen des Roboters getestet wurden, wurde anschließend eine Liste der wichtigsten nicht bearbeiteten Funktionen angefertigt, in der auch Referenzen zu einer anderen Arbeit gegeben sind, die sich mit großen Teilen dieser Funktionen beschäftigt.

Abschließend wurde auf dem Wissen des Aufbaus, der Komponenten und der Funktionen des GO1 eine Reihe neuer Funktionen und Erweiterungen erarbeitet, welche im allgemeinen alle das Ziel verfolgen, den Roboter aus der Ferne steuern zu können und somit deutlich flexibler einsetzbar zu machen. Ein weiteres Ziel der Funktionen ist es, die Grundlage für eine

autonome Nutzung des Roboters zu schaffen, in dem man es steuernder Software ermöglicht, auf die Daten des Roboters zuzugreifen, ohne direkt verbunden zu sein.

Zuletzt wird die Arbeit noch bewertet und ein Ausblick geschaffen, welcher in den folgenden Abschnitten erarbeitet wird.

## 6.2 Einschätzung

Die Arbeit am GO1 ist eine meist ertragreiche, jedoch oftmals auch mühsame. Einerseits sind die Präzision der Bauteile, besonders der mechanischen Bauteile der Motoren und Gelenkstrukturen, die Qualität verbauten Komponenten, wie der drei NVIDIA Jetsons und der Raspberry Pi, sowie der Funktionsumfang von Teilen der gelieferten Software beeindruckend, besonders in Relation zum Einstiegspreis des Roboters. Jedoch zeigt die Ausstattung und Umsetzung besonders des Edu Models des GO1 klare Defizite, möglicherweise durch die kurze Entwicklungszeit, welche sich anhand der verwendeten open-source Software und der Bekanntgabe des Verkaufs annähernd auf zwei bis drei Jahre schätzen lässt. Dieses Model ist auch bedeutend teurer in der Anschaffung, was die initiale Erwartung deutlich erhöht.

Positiv anzumerken sind die Öffnung aller Systeme für die Erweiterung durch neue Funktionen, die ausreichende Rechenkraft der verbauten Einplatinencomputer und die umfangreiche Ausstattung an Sensorik. Besonders die Verwendung von Open-Source Komponenten wie den Betriebssystemen, verwendeten nativen Bibliotheken und Treibern für verbaute Zusatzkomponenten erleichtern die Untersuchung und Erweiterung der Funktionen, die der GO1 bereits ab Werk liefert.

Negativ anzumerken sind hingegen die unzureichende Dokumentation der Roboter, besonders der Unitree-Bibliotheken, sowie das Verbergen der Implementierungen durch die ausschließliche Verwendung von Binärcode für native Funktionen. Das macht es besonders bei der Suche nach Fehlern bei zu testenden Funktionen nahezu unmöglich, die Ursachen zu identifizieren, ohne die Bestandssoftware detailliert zu untersuchen und teilweise zu Reverse-Engineering zurückzugreifen.

Eine einfache Integration des GO1 in ein Hochschulökosystem ist somit ohne angemessenen Aufwand nicht möglich. Die initiale Dokumentation der Funktionen, das Testen und Bewerten dieser muss als Grundlage dieser Arbeit gesehen werden. Um den Roboter dann in realen Situation an der Hochschule einsetzen zu können, kann dann auf die erarbeiteten Erkenntnisse aufgebaut werden. Für die Integration in die Lehre ist der Roboter hingegen ein wertvolles Gut. Das Testen der Funktionen und das Erweitern dieser durch eigene Entwicklungen kann in verschiedenen Kenntnisständen der Robotik durchgeführt werden. So ist dieser Roboter durch die Nutzung bekannter Komponenten wie der NVIDIA Karten und des Raspberry Pis eine

solide Plattform für Anfänger auf dem Gebiet. Arbeitet man jedoch intensiver mit dem Gerät, so Bedarf es eines weiten Spektrums an Kenntnissen, unter anderen in den Bereichen der Grundlageninformatik, hardwarenaher Programmierung und der Programmierung allgemein, Netzwerktechnik, Physik, Elektrotechnik, Systemadministration und besonders der KI. Richtig eingesetzt ist der GO1, sowie ähnliche Geräte, eine sinnvolle Investition in eine Hochschule und der Lehre dieser.

## 6.3 Ausblick

Zuletzt soll diese Arbeit noch als Referenz für Zukünftige Projekte am GO1 dienen. Die Ausarbeitung des Funktionsumfangs des Roboters kann in mehreren Schritten erfolgen, welche in ihrer Komplexität steigen und aufeinander aufbauen. Die Erweiterungen haben das Ziel, den Roboter möglichste autonom verwenden zu können, ihm also eine Aufgabe zuzuweisen, die er dann in allen Belangen ohne menschliche Unterstützung erledigen kann.

Der erste Schritt ist die dauerhafte Erreichbarkeit des Roboters für Entwickler und externe Server, unabhängig des Standortes dessen. Hierfür können die Erkenntnisse aus den Kapiteln Wifi und 5.1 verwendet werden, um den Roboter per VPN an einen Server zu verbinden, über den er nun dauerhaft und unabhängig der Art der Verbindung zum Internet erreichbar ist.

Über diese Verbindung kann im zweiten Schritt eine Art Austausch der Logs und aktuellen Daten zur Telemetrie, des Standortes und des BMS zwischen dem Roboter und einem fest installierten Server implementiert werden. Dies ist nicht zwingend trivial, da hierfür das Format der Daten, das Intervall des Austauschs und die Art der Daten festgelegt werden müssen. Zudem muss das Protokoll zur Kommunikation und das Format der zu speichernden Daten erarbeitet werden. Ein besonderes Augenmerk muss hierfür auf die Latenz der Übertragung gelegt werden, da diese im Laufe dieser Arbeit oftmals ein Problem in der Effizienz der genutzten Funktionen darstellte.

Nach Austausch der relevanten Daten in einem effizienten, robusten und festgelegten Format kann an der Arbeit der manuellen Fernsteuerung des Systems begonnen werden. Auch hier muss der Fokus auf der Übertragungsgeschwindigkeit der Daten und der Zuverlässigkeit der Verbindung liegen. Im Rahmen der Fernsteuerung kann ebenfalls an der effizienten Videoübertragung gearbeitet werden, da diese zur sicheren Steuerung des Roboters ohne Sichtkontakt zu diesem zwingend nötig ist. Auch die Nutzung der Ultraschalldaten kann hier in Betracht gezogen werden.

Als letzter Schritt zur Autonomie ist abschließend die Steuerung zu bearbeiten. Denkbar sind hier zwei Szenarien, die besonders von den Erkenntnissen zur Qualität der Netzwerkverbindung aus den vorigen Schritten abhängig sind. ISt die Übertragung der Daten unzuverlässig,

so sollte die Übertragungsmenge verringert werden und der Roboter über eingebettete Software gesteuert werden. Hier ist der Fokus auf die Effizienz der Implementierung zu legen, da die Recheneinheiten des Roboters nur eine begrenzte Leistung liefern können. Ist die Übertragungsrate ausreichend, so kann die Steuerung des Roboters auf ein externes System ausgelagert werden. Hier ist der Fokus auf die Übertragung der Daten gelegt.

Abschließend kann der GO1 anhand der Erkenntnisse dieser Arbeit und der möglichen Erweiterungen auf mögliche Einsatzzwecke erforscht werden, um den Übergang der akademischen Nutzung zu einer kommerziellen Nutzung zu schaffen, die der Gesellschaft potenziell Nutzen bringt, was im Endeffekt das ultimative Ziel aller akademischen Arbeiten ist, diese Arbeit mit dem Titel *Integration eines Unitree Go1 Quadruped Roboters in ein Hochschul-Ökosystem* eingeschlossen.

# Anhang A Listings

/home/pi/UnitreeUpgrade/start.sh

```
#!/bin/bash
cd /home/pi/UnitreeUpgrade
python3 ./startup_manager.py  &
python3 ./startup_uploader.py &

cd /home/pi/Unitree/autostart
./update.sh &
```

/home/pi/autostart/Unitree/configNetwork/configNetwork.sh

```
#!/bin/bash
eval echo "[configNetwork] starting... " $toStartlog
# configure 4G
sudo ifconfig wlan0 down
sudo ifconfig wlan1 down
sudo ifconfig wlan2 down
sudo ifconfig wwan0 down

_EC25=$(lsusb | sed -n '/EC/P' | wc -c)
if [[ _EC25 -gt 10 ]]; then
  eval echo "[configNetwork] 4G module detected" $toStartlog
  sudo ifconfig eth0 down
  sudo ifconfig eth1 down
  sudo ./ppp/quectel-pppd.sh&
  _ppp0Count=0
  sleep 1
  while [[ $_ppp0Count -lt 6 ]]
  do
    sleep 1
    ((_ppp0Count++))
    eval echo "[configNetwork] waiting for ppp0" $toStartlog
    _ppp0=`ifconfig | grep 'ppp0' | wc -c`
    sleep 1
    if [[ $_ppp0 -gt 10 ]]; then
      eval echo "[configNetwork] ppp0 obtained" $toStartlog
      break
    fi
    sleep 1
  done
  sleep 1
  sudo ifconfig eth0 up
  sudo ifconfig eth1 up
fi

# configure WIFI
sudo ifconfig wlan1 192.168.12.1/24
sudo ifconfig wlan1 up
sudo hostapd /etc/hostapd/hostapd.conf&
eval echo "[configNetwork] config WIFI" $toStartlog
```

```
sleep 5

sudo ./connectWifi2.sh&
```

hostapd.conf

```
interface=wlan1
driver=nl80211
hw_mode=a
ieee80211n=1
ieee80211ac=1
ieee80211d=1
ieee80211h=1
require_ht=1
require_vht=1
wmm_enabled=1
country_code=US

vht_oper_chwidth=1
channel=165
vht_oper_centr_freq_seg0_idx=155
ht_capab=[HT40-][HT40+][SHORT-GI-40][DSSS_CCK-40]

wpa=2
wpa_key_mgmt=WPA-PSK
rsn_pairwise=CCMP
ssid=Unitree_Go501075A
wpa_passphrase=00000000
```

webpack://../bmyReceivers.ts

```
// webpack:///src/plugins/mqtt/receivers/bmsReceivers.ts

import { MqttData } from "@/plugins/mqtt/data";
import { BmsSubTopic } from "@/plugins/mqtt/topics";

type Receivers = {
    [key in BmsSubTopic]: (
        data: MqttData,
        message: Buffer,
        dataView: DataView
    ) => void;
};

const receivers: Receivers = {
    "bms/state": (data, message, dataView) => {
        const uint8s = new Uint8Array(message);
        data.bms.version = uint8s[0] + "." + uint8s[1];
        data.bms.status = uint8s[2];
        data.bms.soc = uint8s[3];
        data.bms.current = dataView.getInt32(4, true);
        data.bms.cycle = dataView.getUint16(8, true);
```

```
        data.bms.temps = [uint8s[10], uint8s[11], uint8s[12], uint8s[13]];
        for (let i = 0; i < 10; i++) {
            data.bms.cellVoltages[i] = dataView.getUint16(14 + i * 2, true);
        }
        data.bms.voltage = data.bms.cellVoltages.reduce((a, c) => a + c);
    }
};

export default receivers;
```

/home/pi/autostart/Unitree/configNetwork/connectWlan2.sh

```
#!/bin/bash
# /home/pi/Unitree/autostart/configNetwork/connectWlan2.sh

sleep 1
echo "[connectWlan2] Start initializing wlan2 interface" $toStartlog
sudo ifconfig wlan2 up
sudo rm /var/run/wpa_supplicant/wlan2
echo "[connectWlan2] Connecting wlan2 interface" $toStartlog
sudo wpa_supplicant -B -i wlan2 -c /etc/wpa_supplicant/config.d/wpa_wlan2.conf
echo "[connectWlan2] Done connecting wlan2 interface via wpa_supplicant" $toStartlog
sleep 3
```

/home/pi/autostart/Unitree/configNetwork/connectWwan0.sh

```
#!/bin/bash
# /home/pi/Unitree/autostart/configNetwork/connectWwan0.sh

sleep 1
echo "[connectWwan0] Preparing environment for wwan0 interface" $toStartlog
sudo systemctl stop ModemManager
echo "[connectWwan0] Start initializing wwan0 interface" $toStartlog
sudo ifconfig wwan2 up
sudo qmicli -p -d /dev/cdc-wdm0 --device-open-net='net-raw-ip|net-no-qos-header' --wds-start-network="apn='internet',ip-type=4" --client-no-release-cid
echo "[connectWwan0] Getting IP for wwan0 interface" $toStartlog
sudo udhcpc -q -f -i wwan0
echo "[connectWwan0] Connected wwan0 to internet" $toStartlog
sleep 3
```

/home/pi/Extensions/go1-bmsMonitor/install.sh

```
#!/bin/bash

DIR="/home/pi/Unitree/autostart/"
STARTUP_FILE=".startlist.sh"
SCRIPT_DIR=$( cd -- "$( dirname -- "${BASH_SOURCE[0]}" )" &> /dev
    ↪ /null && pwd )

if ! command -v python &> /dev/null
then
  echo "python could not be found, exiting..."
  exit 1
fi

if ! [ -d "$DIR" ];
then
  echo "$DIR was not found, exiting..."
  exit 1
fi

if [ -f "$STARTUP_FILE" ];
then
  echo "$STARTUP_FILE was not found, exiting..."
  exit 1
fi

if grep -q "bmsMonitor" "$DIR$STARTUP_FILE" ;
then
  echo "bmsMonitor already in $STARTUP_FILE";
else
  echo "bmsMonitor" >> "$DIR$STARTUP_FILE"
fi


# Ordner loeschen
rm -r "$DIR/bmsMonitor" &> /dev/null
# Symbolische Links zum Repository erstellen
cp -rs "$SCRIPT_DIR/bmsMonitor" "$DIR"
# Skript kopieren um Pfade zu erhalten
cp -f "$SCRIPT_DIR/bmsMonitor/bmsMonitor.template.sh" "$DIR/
    ↪ bmsMonitor/bmsMonitor.sh"
# Skripte ausfuehrbar machen
chmod +x "$DIR/bmsMonitor/bmsMonitor.sh" "$SCRIPT_DIR/bmsMonitor/
    ↪ run.sh"
```

/home/pi/Extensions/go1-bmsMonitor/bmsMonitor/run.sh

```
#!/bin/bash
myNow=$(date +"%T")
eval echo "$myNow [diagnosis] starting... " $toStartlog
sleep 30
myNow=$(date +"%T")
eval echo "$myNow [diagnosis] starting the monitor" $toStartlog

```

```
python3 -m pip install -r /home/pi/Unitree/autostart/bmsMonitor/requirements.txt
python3 /home/pi/Unitree/autostart/bmsMonitor/snif_bms.py
```

/home/pi/Extensions/go1-bmsMonitor/bmsMonitor/sniff_bms.py

```
import struct

import paho.mqtt.client as mqtt

from constants import HOSTNAME, PORT, TIMEOUT, BMS_STATE_TOPIC, CONTROLLER_ACTION_TOPIC
from led_control import alert_led, const_led


def on_connect(c, userdata, flags, rc):
    c.subscribe(BMS_STATE_TOPIC)


def on_message(c, userdata, msg):
    print("message")
    [ver0, ver1, status, soc, current, cycle, temp0, temp1,
     temp2, temp3, cell_v0, cell_v1, cell_v2, cell_v3,
     cell_v4, cell_v5, cell_v6, cell_v7, cell_v8, cell_v9] \
        = struct.unpack('BBBBiHBBBBHHHHHHHHHH', msg.payload)
    if not bms_responding(msg.payload):
        const_led(c, 10, 10, 0)
    elif 5 >= soc:
        alert_led(c, 255, 0, 0)
        lay_down()


def bms_responding(payload):
    # if payload only zeros, return false;
    return 0 != int.from_bytes(payload, byteorder='big')


def lay_down():
    client.publish(CONTROLLER_ACTION_TOPIC, payload="standDown")
    client.publish(CONTROLLER_ACTION_TOPIC, payload="damping")


client = mqtt.Client("Stick Data")
client.on_connect = on_connect
client.on_message = on_message

client.connect(HOSTNAME, PORT, TIMEOUT)

try:
    client.loop_forever()
except:
    client.disconnect()
```

/home/pi/Extensions/go1-bmsMonitor/bmsMonitor/led_control.py

```
1  import struct
2  import time
3
4  from constants import FACE_LIGHT_TOPIC, QOS
5
6
7  def alert_led(client, r, g, b):
8      client.publish(FACE_LIGHT_TOPIC, struct.pack('BBB', r, g, b),
          ↪  qos=QOS)
9      time.sleep(1)
10     client.publish(FACE_LIGHT_TOPIC, struct.pack('BBB', 0, 0, 0),
          ↪  qos=QOS)
11     time.sleep(1)
12
13
14 def const_led(client, r, g, b):
15     client.publish(FACE_LIGHT_TOPIC, struct.pack('BBB', r, g, b),
          ↪  qos=QOS)
16     time.sleep(1)
```

/home/pi/Extensions/go1-bmsMonitor/bmsMonitor/constants.py

```
1  HOSTNAME = "192.168.12.1"
2  PORT = 1883
3  TIMEOUT = 60
4  FACE_LIGHT_TOPIC = "face_light/color"
5  BMS_STATE_TOPIC = "bms/state"
6  CONTROLLER_ACTION_TOPIC = "controller/action"
7  QOS = 2  # exactly once
```

Ausschnitt von webpack:///src/views/Vision.vue

```
172 const  handleStickData = (evt: {
173     lx: number;
174     ly: number;
175     rx: number;
176     ry: number;
177 }) => {
178     const  floats = new  Float32Array (4);
179     floats [0] = evt.lx;
180     floats [1] = evt.rx;
181     floats [2] = evt.ry;
182     floats [3] = evt.ly;
183     mqtt.publish("controller/stick", floats.buffer , { qos: 0 });
184 };
```

# Literatur


[1] Evan Ackerman. „Boston Dynamics' Spot Robot Dog Now Available for $74,500. For the price of a luxury car, you can now get a very smart, very capable, very yellow robotic dog". In: *IEEE Spectrum* (16. Juni 2020).

[2] Defense Advanced Research Projects Agency. *Big Dog*. URL: urhttps://www.darpa.mil/about-us/timeline/big-dog (besucht am 07. 08. 2023).

[3] Defense Advanced Research Projects Agency. *Maximum Mobility and Manipulation (M3)*. URL: https://www.darpa.mil/program/maximum-mobility-and-manipulation (besucht am 07. 08. 2023).

[4] Alsa-Projekt.org. *Advanced Linux Sound Architecture (ALSA) project homepage*. 20. Okt. 2020. URL: https://www.alsa-project.org/wiki/Main_Page (besucht am 18. 08. 2023).

[5] Gerardo Bledt u. a. „MIT Cheetah 3: Design and Control of a Robust, Dynamic Quadruped Robot". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, S. 2245–2252. DOI: 10.1109/IROS.2018.8593885.

[6] MDN contributors. *DataView*. 21. Aug. 2013. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/DataView?retiredLocale=de (besucht am 22. 08. 2023).

[7] MDN contributors. *Uint8Array*. 14. Aug. 2013. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Uint8Array (besucht am 22. 08. 2023).

[8] NVIDIA Corporation. *Entwicklerkit und -module für eingebettete Systeme*. 2023. URL: urhttps://www.nvidia.com/de-de/autonomous-machines/embedded-systems/.

[9] *Development and use of Go1 binocular fisheye camera*. Unitree Robotics.

[10] *Development and use of Go1 ultrasonic module*. Unitree Robotics.

[11] Boston Dynamics. *Foto eines BigDog*. URL: https://www.bostondynamics.com/wp-content/uploads/2023/07/bigdog-1-1.jpg (besucht am 28. 08. 2023).

[12] Boston Dynamics. *Foto eines Spot*. URL: https://bostondynamics.com/wp-content/uploads/2023/05/spot-explorer-web-2-2048x1152.jpg (besucht am 28. 08. 2023).

[13] Boston Dynamics. *Legacy Robots*. URL: https://bostondynamics.com/legacy/ (besucht am 08. 08. 2023).

[14] Boston Dynamics. „Robotics' Role in Public Safety. How robots like Boston Dynamics' Spot are keeping people safe." In: (2023).

[15] Jason Falconer. „MIT Cheetah Robot Runs Fast, and Efficiently. It's now the second fastest legged robot in the world". In: *IEEE Spectrum* (14. Mai 2013).

[16] Sophie Fischer. *Robotics – Market data analysis & forecasts*. Statista Technology Market Outlook. Statista, Aug. 2022. URL: https://de.statista.com/statistik/studie/id/116785/dokument/robotics-report/ (besucht am 04. 07. 2023).

[17] FreeDesktop.org. *Autostart Of Applications During Startup*. URL: https://specifications.freedesktop.org/autostart-spec/0.5/ar01s02.html (besucht am 17. 08. 2023).

[18] Srijeet Halder u. a. „Real-Time and Remote Construction Progress Monitoring with a Quadruped Robot Using Augmented Reality“. In: *Buildings* (Dez. 2022). URL: https://doi.org/10.3390/buildings12112027.

[19] Devan Joseph. „MIT reveals how its military-funded Cheetah robot can now jump over obstacles on its own“. In: *Business Insider* (3. Juni 2015).

[20] Jonas Kemnitzer. „Entwicklung eines Intelligenten lidarbasierten 3D Navigationssystems für den Unitree GO1“. Masterarbeit. Hochschule für Angewandte Wissenschaften Hof, 2023.

[21] Noah Lehmann. *go1-bmsMonitor*. 1. Juli 2023. URL: https://github.com/noahlehmann/go1-bmsMonitor (besucht am 24. 08. 2023).

[22] Jouni Malinen. *hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator*. 12. Jan. 2013. URL: https://w1.fi/hostapd/ (besucht am 18. 08. 2023).

[23] Lee Milburn, Juan Gamba und Claudio Semini. *Towards Computer-Vision Based Vineyard Navigation for Quadruped Robots*. Dynamic Legged Systems Lab - Istituto Italiano di Tecnologia Genova, Italy, 2. Jan. 2023.

[24] MIT. *Foto eines Cheetah 3*. URL: https://news.mit.edu/sites/default/files/styles/news_article__image_gallery/public/images/201903/MIT-Mini-Cheetah-01_0.jpg (besucht am 28. 08. 2023).

[25] Jeremy Moses und Geoffrey Ford. *The Rise of the Robot Quadrupeds*. 11. Dez. 2020. URL: https://mappinglaws.net/rise-robot-quadrupeds.html (besucht am 07. 08. 2023).

[26] VDI-Gesellschaft Produktionstechnik. *VDI 2860/ Montage- und Handhabungstechnik. Handhabungsfunktionen, Handhabungseinrichtungen; Begriffe, Definitionen, Symbol*. Beuthe Verlag, 1990.

[27] Marc Raibert u. a. *BigDog, the Rough-Terrain Quaduped Robot*. Boston Dynamics, 8. Apr. 2008.

[28] Unitree Robotics. *About Unitree Robotics*. URL: https://www.unitree.com/en/about/ (besucht am 28. 08. 2023).

[29] Unitree Robotics. *GO-M8010-6 Motor - Unitree Robotics*. URL: https://shop.unitree.com/products/go1-motor (besucht am 07. 07. 2023).

[30] Unitree Robotics. *Go1 APP Download*. URL: uhttps://www.unitree.com/app/ (besucht am 17. 08. 2023).

[31] Unitree Robotics. *GO1 Manuals - GO1 Tutorials 1.0.0 documentation*. URL: https://www.docs.quadruped.de/projects/go1/html/index.html (besucht am 08. 08. 2023).

[32] Unitree Robotics. *Unitree Go1*. URL: https://shop.unitree.com/cdn/shop/products/75_540x.jpg?v=1668073774 (besucht am 28. 08. 2023).

[33] Spektrum. *Lexikon der Biologie*. Spektrum Akademischer Verlag Heidelberg, 1999.