HAW HAMBURG

BACHELOR THESIS
Jannes Bahr

Entwicklung und Simulation einer quadrupeden Roboterplattform zur Untersuchung von Gangarten

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Jannes Bahr

Entwicklung und Simulation einer quadrupeden Roboterplattform zur Untersuchung von Gangarten

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Informatik Technischer Systeme*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Tim Tiedemann
Zweitgutachter: Prof. Dr. Thomas Lehmann

Eingereicht am: 18.03.2025

Jannes Bahr

Thema der Arbeit

Entwicklung und Simulation einer quadrupeden Roboterplattform zur Untersuchung von Gangarten

Stichworte

Robotik, Simulation, Quadruped, ROS, Gazebo, Gangarten

Kurzzusammenfassung

Diese Arbeit widmet sich der Weiterentwicklung und Simulation einer quadrupeden Roboterplattform. Hierbei werden von der Ansteuerung der Hardware und Simulation über die Kinematik bis hin zur Entwicklung einfacher Gangarten viele Aspekte der Robotik behandelt. Mehrere Reihen von Experimenten bewerten die Genauigkeit von Bewegungen des Roboters und der Simulation. In der Simulation wird nachgewiesen, dass der Roboter in der Lage ist, einfache Gangarten zu nutzen. Zudem wird anhand der Gangarten Schritt und Trott gezeigt, dass eine Bewertung der Qualität von Gangarten anhand von drei beispielhaften Metriken möglich ist.

Jannes Bahr

Title of Thesis

Development and simulation of a quadruped robot platform for researching gaits

Keywords

Robotics, Simulation, Quadruped, ROS, Gazebo, Gaits

Abstract

This thesis is dedicated to the development and simulation of a quadruped robot platform. It covers many aspects of robotics, from hardware control and simulation, to kinematics and the development of simple gaits. Several series of experiments show the accuracy of the robot and the simulation. Using the simulation, it is shown that the robot is able to use simple gaits like walk and trot. An evaluation of the quality of gaits based on three exemplary metrics is also shown.

Inhaltsverzeichnis

Abbildungsverzeichnis **viii**

Tabellenverzeichnis **xi**

Abkürzungsverzeichnis **xii**

1 Einleitung **1**
1.1 Motivation . . . 1
1.2 Zielsetzung . . . 1
1.3 Verwandte Arbeiten . . . 2
1.4 Übersicht . . . 4

2 Grundlagen **6**
2.1 Gangarten . . . 6
2.2 ROS . . . 7
2.3 Gazebo . . . 9
2.4 Xacro . . . 9
2.5 Entwurfsmuster . . . 10
2.6 Denavit-Hartenberg-Konvention . . . 10
2.7 Stabilität und stabile Zyklen . . . 11

3 Hardware und Hardwareansteuerung **13**
3.1 Übersicht über den Roboter . . . 13
3.2 Sicherheitsanmerkungen . . . 14
3.3 Bauteile und Komponenten . . . 14
3.3.1 Boardcomputer . . . 14
3.3.2 Tracking Kamera . . . 16
3.3.3 Encoder . . . 16
3.3.4 Umrichter . . . 16
3.3.5 Motoren . . . 17

3.3.6 Motor Hubs . . . 17
3.3.7 Energieversorgung . . . 19
3.3.8 Beine . . . 20
3.3.9 Nutzereingaben . . . 21
3.4 Motoransteuerung . . . 22
3.5 Motorparametrierung . . . 24
3.5.1 Retrospektive auf einen Fehler in der Parametrierung . . . 25
3.6 Untersuchung des Motorverhaltens . . . 26
3.6.1 Versuchsaufbau . . . 26
3.6.2 Ergebnisse . . . 27

4 Simulation von Motoren 29
4.1 Simulierte Motoren in Gazebo . . . 29
4.2 Parametrierung simulierter Motoren . . . 30
4.3 Untersuchung simulierter Motoren . . . 30
4.3.1 Versuchsaufbau . . . 30
4.3.2 Ergebnisse . . . 31

5 Bewegungskoordination 32
5.1 Berechnung der Achspositionen . . . 32
5.2 Kalibrierung der Achspositionen . . . 34
5.3 Berechnung der Kinematik . . . 34
5.4 Begrenzung der möglichen Konfigurationen . . . 36
5.5 Untersuchung der Linearität von Beinbewegungen . . . 38
5.5.1 Versuchsaufbau . . . 38
5.5.2 Ergebnisse . . . 40
5.6 Untersuchung der Bewegungslinearität unter Last . . . 40
5.6.1 Versuchsaufbau . . . 40
5.6.2 Ergebnisse . . . 41
5.7 Untersuchung der Bewegungslinearität bei ruckartigen Bewegungen . . . 42
5.7.1 Versuchsaufbau . . . 42
5.7.2 Ergebnisse . . . 43

6 Simulation des Gesamtsystems 44
6.1 Beschreibung des Roboters . . . 44
6.1.1 Geometrie . . . 44
6.1.2 3D-Modelle . . . 45

6.1.3 Trägheit und Gewicht . . . 46
6.2 Untersuchung der Abweichung von Beinbewegungen in der Simulation . . 47
6.2.1 Versuchsaufbau . . . 47
6.2.2 Ergebnisse . . . 48
6.3 Untersuchung der Abweichung von langsamen Körperbewegungen in der Simulation . . . 49
6.3.1 Versuchsaufbau . . . 49
6.3.2 Ergebnisse . . . 49
6.4 Untersuchung der Abweichung von ruckartigen Körperbewegungen in der Simulation . . . 51
6.4.1 Versuchsaufbau . . . 51
6.4.2 Ergebnisse . . . 51
6.5 Festgestellte Probleme in der Gazebo Simulation . . . 52

7 Robotersteuerung 53
7.1 Klassen für Gangarten (Gait) . . . 53
7.2 Klassen für die Repositionierung (Reposition) . . . 53
7.3 Klassen für Nutzereingaben (UserInput) . . . 54
7.4 Klasse für die Beinansteuerung (LegControl) . . . 55
7.5 Konzept für das Wechseln von Gangarten (GaitSelector) . . . 55
7.6 Übersicht über die Klassen . . . 57

8 Gangarten und erste Schritte 59
8.1 Gangarten . . . 59
8.1.1 Gangart Schritt . . . 59
8.1.2 Gangart Trott . . . 60
8.2 Untersuchung der Geschwindigkeit und Stabilität . . . 62
8.2.1 Testmethodik . . . 62
8.2.2 Ergebnisse . . . 63
8.3 Untersuchung der Bewegungsruhe . . . 64
8.3.1 Testmethodik . . . 64
8.3.2 Ergebnisse . . . 64

9 Fazit 66
9.1 Zusammenfassung der Evaluationen . . . 66
9.1.1 Motoren . . . 66
9.1.2 Beine . . . 66

9.1.3 Bewegung 67
9.2 Diskussion 67
9.2.1 Realer Roboter 67
9.2.2 Simulierter Roboter 68
9.3 Schluss 68
9.4 Ausblick 69

Literaturverzeichnis **71**

A Anhang **74**
A.1 Verwendete Hilfsmittel 74

Selbstständigkeitserklärung **75**

Abbildungsverzeichnis

1.1 Fotos von MIT "Mini Cheetah", Boston Dynamics "Spot", "Unitree A1" und ETH Zürich "ANYmal" aus den jeweiligen Veröffendlichungen 3

2.1 Beispielgangarten für Hunde (Schritt, Trott, Passgang, Galopp) 7
2.2 Beispiel einer Publish-Subscribe-Beziehung zweier Nodes aus der ROS Dokumentation [Open Robotics, 2025a]. 8
2.3 Darstellung und DH-Parameter für die Kinematik des Quadruped-Beines aus "Inverse Kinematic Analysis of a Quadruped Robot" [Sen u. a., 2017] . 11
2.4 Darstellung des Stützpolygons und Schwerpunktes eines sechsbeinigen Roboters aus "Handbook of Robotics" [Siciliano und Khatib, 2008, p.378] . . 11
2.5 Darstellung eines Grenzzyklus in einem Phasenraum aus "Nichtlineare Regelung" [Adamy, 2009, p.16] 12

3.1 Foto des Roboters in einer Halterung mit der Plattform für Tests mit Bodenkontakt 13
3.2 Fotos unterschiedlicher im Roboter verbauter Komponenten 15
3.3 Querschnitt eines Motor Hubs mit Beschriftung der Komponenten 18
3.4 Fotos eines Motor Hub mit verbautem Teensy Mikrocontroller 19
3.5 Übersicht über die Versorgung des Roboters 20
3.6 Übersicht über den Aufbau des Beines FL mit besonderem Augenmerk auf die Kraftübertragung von den Motoren zu den Gelenken 20
3.7 Foto eines Hub mit angebautem Bein 21
3.8 Xbox One Controller und 3Dconnexion SpaceMouse 22
3.9 `odrive_ros_bridge` Target und Feedback Messages je Motor 23
3.10 Kaskadierende Reglerstruktur des ODrives aus der ODrive Dokumentation [Odrive Robotics, 2021] 25
3.11 Testaufbau für Motortests unter Last mit einem Gewicht von $5\,kg$ 27
3.12 Sprungantwort des Reglers bei Sprungweiten von 30°, 60°, 90° und 120°. . 27
3.13 Genauigkeit des Reglers beim Abfangen unter unterschiedlichen Lasten . . 28

4.1 Sprungantwort mit Toleranz des Reglers bei Sprungweiten von 30°, 60°, 90° und 120° 31

5.1 Aufbau der Kinematik-Berechnung aus der Ansicht von hinten (links im Bild) und seitlich (rechts im Bild) 35
5.2 Seitenansicht der Hindernis-Geometrie (blau) und des durch die Hindernis-Geometrie eingeschränkten Konfigurationsraums (grün) 37
5.3 Darstellung einer diagonalen Bewegung bei maximal 2000 mm/s (links) und 3000 mm/s (rechts) mit einer Toleranz von 20 mm 39
5.4 Ergebnisse der Linearitätstests für diagonale (links) und vertikale (rechts) Bewegungen 40
5.5 Darstellung einer vertikalen und horizontalen Sinusbewegung der Beine unter der Last des Roboterchassis 41
5.6 Darstellung einer ruckartigen vertikalen Bewegung des Roboters unter Last 43

6.1 Darstellung der Links des Roboters mit dem Koordinatenursprung und der Orientierung der Achsen 45
6.2 Ausschnitt aus dem TF-Tree des Roboters 45
6.3 3D-Modell des Roboters ohne Anbauteile in Autodesk Fusion 360 46
6.4 Visualisierungs- und Kollisionsmodell des Roboters in Gazebo 46
6.5 Darstellung unterschiedlicher Bewegungen in der Simulation und der Realität zur Veranschaulichung der Abweichung der Simulation 48
6.6 Darstellung zweier Sinus-Bewegungen des Roboters in der Simulation und der Realität zur Veranschaulichung der Abweichung der Simulation 50
6.7 Darstellung einer ruckartigen Bewegung des Roboters in der Simulation und der Realität zur Veranschaulichung der Abweichung der Simulation . 51
6.8 Visualisierung des Drifts bei problematischen Positionen in Gazebo anhand der gelben Pfadlinie 52

7.1 Zustandsautomat für das Wechseln von Gangarten durch die Gangsteuerung 56
7.2 Vereinfachtes Klassendiagramm für das Wechseln von Gangarten durch die Gangsteuerung 57
7.3 Informelle Darstellung einer Beispielkonstruktion der in Abbildung 7.2 gezeigten Klassen, sowie die Richtung der ausgetauschten Informationen . . 58

8.1 Gegenüberstellung der Vorwärtsbewegung eines Beines in den beiden Trott-Versionen 61

8.2 Der Roboter beim prädiktiven Trott unter der Einwirkung von $12.5\,kN$ (links) und $15\,kN$ (rechts) 62
8.3 Krafteinwirkung auf den Roboter bei verschiedenen Geschwindigkeiten für die unterschiedlichen Gangarten 63
8.4 Darstellung der Zyklen verschiedener Gangarten als Versatz zur Zielbewegung (je Farbe wird ein gemessener Zyklus dargestellt) 65

Tabellenverzeichnis

3.1 Übersicht über wichtige ODrive Parameter 24

5.1 Mechanische Limits der Gelenke am Beispiel des Beines vorne rechts . . . 33

8.1 Parameter für die Gangart Schritt 60
8.2 Parameter der Gangart Trott 62
8.3 Maximaler Versatz zum Ursprung der verschiedenen Gangarten 65

A.1 Verwendete Hilfsmittel und Werkzeuge 74

Abkürzungsverzeichnis

BLDC Brushless Direct Current
CAD Computer Aided Design
CoM Center of Mass
DH Denavit-Hartenberg
DoF Degree of Freedom
IMU Inertial Measurement Unit
LED Light Emitting Diode
LiDAR Light Detection and Ranging
NUC Next Unit of Computing
P Proportional
PCB Printed Circuit Board
PI Proportional-Integral
PID Proportional-Integral-Differential
ROS Robot Operating System
RQT ROS-Qt
STAIR Stanford Artificial Intelligence Robot
TBD To Be Determined
TDP Thermal Design Power
TF Transformation
UART Universal Asynchronous Receiver-Transmitter
URDF Unified Robot Description Format
USB Universal Serial Bus
VESC Vedder Electronic Speed Controller
Xacro XML Macros
XML Extensible Markup Language
ZMP Zero Moment Point

1 Einleitung

1.1 Motivation

Quadrupede Roboterplattformen haben in den letzten Jahren immer mehr an Bedeutung gewonnen. Da laufende Roboter in der Lage sind, sich sowohl in für Menschen gemachten, als auch in natürlichen Umgebungen zu bewegen, sind sie für viele Anwendungen interessant. Von einfachen Inspektionen in Fabriken über Rettungseinsätze in gefährlichen Umgebungen, bis hin zur potenziellen Erkundung von Planeten oder Monden.

Bei Inspektionsaufgaben haben quadrupede Roboter den Vorteil, dass sie im Gegensatz zu fahrenden Plattformen auch höhere Hindernisse oder Treppen überwinden können. Bei Einsätzen in Katastrophengebieten oder beispielsweise auf dem Mars haben sie mehr Möglichkeiten, sich auf unebenem oder instabilem Gelände zu bewegen. Alle diese Anwendungsfälle erfordern eine extrem hohe Zuverlässigkeit sowohl in der Hardware als auch in der Software.

Die Steuerung quadrupeder Roboter ist eine große Herausforderung. Es existieren sehr viele Freiheitsgrade und ein stabiler Stand des Roboters ist nicht garantiert. Dies macht das Entwickeln von Gangarten und Verhaltensweisen für quadrupede Roboter sehr komplex. Im Falle von Programmfehlern kann es zu Schäden an der Hardware kommen. Reparaturen können aufgrund der meist sehr komplexen Mechanik sehr teuer und aufwendig sein.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist die Weiterentwicklung der Hardware, sowie der Entwurf und die Implementierung der grundlegenden Software und Simulation eines quadrupeden Roboters, als Testplattform für Gangarten. Die Hardware ist dabei nicht das Hauptaugenmerk der Arbeit. Es sollen jedoch einige Entscheidungen im Entwurf der Mechanik und

Elektronik beleuchtet werden, die für ein Verständnis der Funktionsweise des Roboters notwendig sind.

Als Ergebnis der Arbeit soll die erstellte Plattform in der Lage sein, die Gangarten Schritt und Trott zu nutzen und der Nachweis erbracht werden, dass der Vergleich der Gangarten durch die erhobenen Daten aus der Steuerung möglich ist. Eine Einschränkung in dieser Arbeit ist, dass als Untergrund nur ebene Flächen ohne Hindernisse betrachtet werden. Auf diese Weise müssen keine zusätzlichen Sensoren für die Erkennung des Untergrundes betrachtet werden.

Als Zielvorgabe für die Plattform soll bei voller Geschwindigkeit eine maximale Abweichung von Körperbewegungen von 5 % der Schulterhöhe des Roboters erreicht werden. Dieser Wert scheint aufgrund des Gewichts des Roboters und der mechanischen Konstruktion der Beine als realistisch. Durch diese Toleranz ergibt sich eine maximale Abweichung der Fußpositionen von $20\,mm$. Weiterhin wird für die Simulation der Antriebe eine Toleranz von 1 % des durchschnittlichen Arbeitsbereichs der Antriebe, sowie $20\,ms$ an zeitlichem Versatz angenommen. 1 % des Arbeitsbereichs entspricht 2.5 °. Diese Werte beziehen sich auf Abweichungen der Simulation zum realen Roboter.

Durch diese Ziele ergeben sich die folgenden Fragen, die in dieser Arbeit beantwortet werden sollen:

- Lassen sich die vorgegebenen Anforderungen an die Genauigkeit des Roboters mit den verbauten Hardware und der in dieser Arbeit entwickelten Software erfüllen?
- Ist Gazebo geeignet um einen quadrupeden Roboter innerhalb der gesetzten Toleranzen abzubilden?
- Lassen sich Gangarten Schritt und Trott auf Basis der entwickelten Plattform vergleichen und bewerten?

1.3 Verwandte Arbeiten

Der Bereich der quadrupeden Robotik ist, wie bereits erwähnt, ein sehr aktives Forschungsfeld. In den letzten Jahren sind viele Arbeiten zu unterschiedlichen Aspekten von vierbeinigen Robotern erschienen. So beschäftigt sich die Arbeit "Mini Cheetah: A Platform for Pushing the Limits of Dynamic Quadruped Control" [Katz u. a., 2019] mit dem Design eines vierbeinigen Roboters, der auf Agilität ausgelegt ist.

Ein größerer Roboter mit mehr Tragkraft wird in "ANYmal: A Highly Mobile and Dynamic Quadrupedal Robot" [Hutter u. a., 2016] vorgestellt.

Auch in der Industrie werden Quadrupeds mittlerweile verwendet. "Spot" von Boston Dynamics wird in einigen Bereichen als Inspektionsroboter eingesetzt. Unitree Robotics hat mit der A- und GO-Reihe eine für Forschungszwecke deutlich kostengünstigere Alternative zu den oben genannten Robotern auf den Markt gebracht.

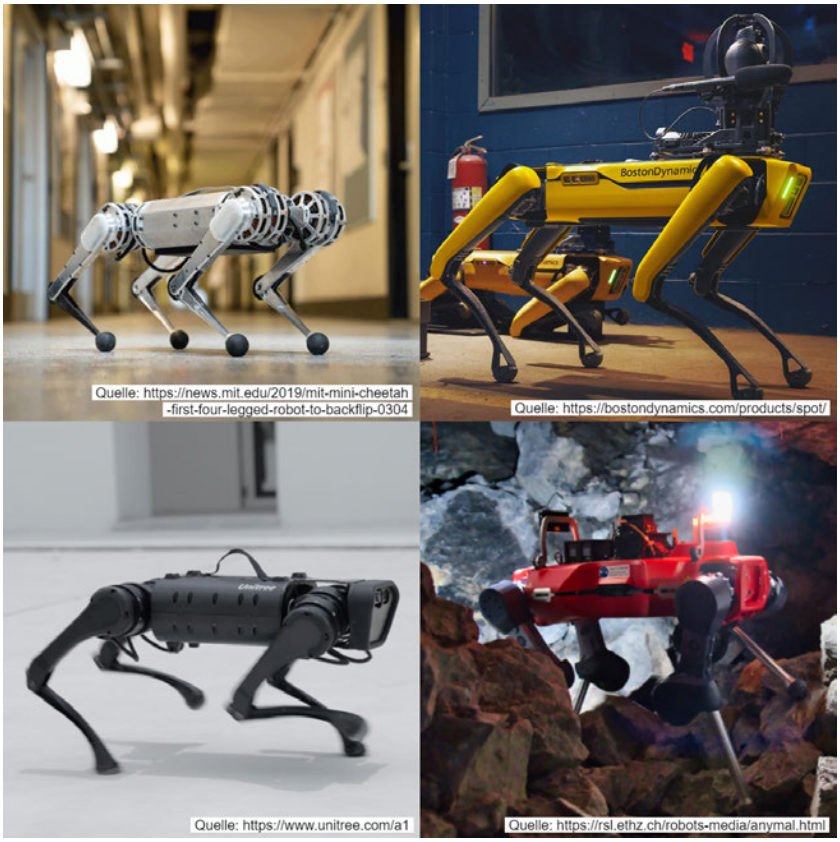


Abbildung 1.1: Fotos von MIT "Mini Cheetah", Boston Dynamics "Spot", "Unitree A1" und ETH Zürich "ANYmal" aus den jeweiligen Veröffendlichungen

Neben der zum Beispiel beim MIT "Mini Cheetah" oder "ANYmal" verwendeten Anordnung von Achsen gibt es noch andere Geometrien, an denen geforscht wird. So nutzt der Roboter aus "System Design and Testing of the Hominid Robot Charlie" [Kuehn u. a., 2016] neben einer beweglichen Wirbelsäule auch eine größere Anzahl an Achsen in den Extremitäten.

Die ETH Zürich hat neben "ANYmal" auch den Roboter "SpaceBok" in der Arbeit "SpaceBok: A Dynamic Legged Robot for Space Exploration" [Arm u. a., 2019] entwickelt. Dieser ist auf Fortbewegung durch Sprünge ausgelegt und dient als Versuchsplattform für potenzielle Anwendungen von ähnlichen Robotern auf anderen Planeten.

Eine komplett andere Herangehensweise an die verbauten Aktuatoren wird in "A soft quadruped robot enabled by continuum actuators" [Muralidharan u. a., 2021] verwendet. Durch die Verwendung von elektrisch betriebenen biegsamen Aktuatoren sind die Beine des Roboters nicht starr.

Neben den mechanischen Eigenschaften der Roboter gibt es auch viele Arbeiten, die sich mit dem Laufen von vierbeinigen Robotern beschäftigen. "Extreme Parkour with Legged Robots" [Cheng u. a., 2023] nutzt zum Beispiel einen rein auf neuronalen Netzen basierenden Ansatz für die Steuerung von Bewegungen, um einen Unitree A1 über unterschiedlichste Hindernisse laufen zu lassen.

Die Übergänge zwischen unterschiedlichen Gangarten werden in "Gaits and gait transitions for legged robots" [Haynes und Rizzi, 2006] untersucht. Hierbei werden anhand einer sechsbeinigen Roboterplattform Übergänge zwischen Gangarten gesucht, ohne dass der Roboter stehenbleibt.

Die zusammenfassende Arbeit "A survey of the development of quadruped robots: Joint configuration, dynamic locomotion control method and mobile manipulation approach" [Chai u. a., 2022] schafft einen Überblick über unterschiedliche aktuelle Ansätze in der quadrupeden Robotik. Hierunter sind Themen wie die historische Entwicklung, Gelenkkonfigurationen, Gangarten, Steuerungsmethoden und Manipulatoren auf quadrupeden Robotern.

1.4 Übersicht

Für diese Arbeit wird ein kaskadierender Ansatz gewählt, um die Bearbeitung der einzelnen Themen zu strukturieren und verständlicher zu machen. Entsprechend wird in einigen Kapiteln eine Evaluation der Zwischenergebnisse der jeweils durchgeführten Versuche vorgenommen. Am Ende der Arbeit werden die Zwischenergebnisse zusammengefasst bewertet.

Um neben dem Inhaltsverzeichnis eine Übersicht über die Arbeit zu geben, folgt eine kurze Zusammenfassung des Inhalts der einzelnen Kapitel:

- **Grundlagen** enthält eine Übersicht über das Basiswissen, das für das Verständnis der Arbeit notwendig ist, sowie eine kurze Vorstellung einer alternativen Methode zur Berechnung der Kinematik.
- **Hardware und Hardwareansteuerung** beschäftigt sich mit dem für diese Arbeit entwickelten Roboter. Es beinhaltet eine kurze Vorstellung der Mechanik und der verbauten Teile, sowie die Programmierung der Hardwareabstraktion für die Motoren. Zudem wird die Einstellung der Umrichter behandelt und experimentell mit und ohne Last überprüft.
- **Simulation von Motoren** beinhaltet eine kurze Übersicht über die Simulation von Motoren. Durch Vergleiche mit einem realen Motor wird die Genauigkeit der Simulation sichergestellt.
- **Bewegungskoordination** behandelt die Berechnungen von Motorpositionen zu Gelenkwinkeln, die Kalibrierung der Beine sowie die Kinematiken. Zudem werden Begrenzungen im Konfigurationsraum des Roboters festgelegt. Über mehrere Experimente wird die Genauigkeit der Beinbewegungen mit und ohne Last, sowie die maximale Geschwindigkeit der Beine überprüft.
- **Simulation des Gesamtsystems** beschäftigt sich mit der Simulation des gesamten Roboters. Es werden die auftretenden Unterschiede zwischen der Simulation und dem realen Roboter durch mehrere Versuche untersucht. So das sichergestellt ist, dass die Simulation den Roboter innerhalb der gesetzten Toleranzen abbilden kann.
- **Robotersteuerung** gibt eine grobe Übersicht über die Softwarearchitektur, die die Ansteuerung des Roboters verwaltet. Hierfür werden die wichtigsten Komponenten dieser Software beschrieben.
- **Gangarten und erste Schritte** beschreibt die für diese Arbeit implementierten Gangarten. Zudem werden diese Gangarten auf ihre Maximalgeschwindigkeit, ihre Stabilität sowie ihre Laufruhe untersucht.
- **Fazit** beinhaltet eine Zusammenfassung, sowie die Bewertung der Ergebnisse im Hinblick auf die Zielsetzung der Arbeit. Zudem wird ein Ausblick auf mögliche Weiterentwicklungen gegeben.

2 Grundlagen

In den nachfolgenden Abschnitten werden Grundlagen und Programme beschrieben, die für das Verständnis der Arbeit notwendig sind. Diese Beschreibungen sollen zunächst nur einen generellen kurzen Einblick in die Themen geben, genauere Informationen können den Quellen entnommen werden.

2.1 Gangarten

Nach "Gaits and gait transitions for legged robots" [Haynes und Rizzi, 2006] ist ein Gait oder auf deutsch eine Gangart oder ein Gang, ein zyklisches Bewegungsmuster, das zu einer Fortbewegung führt. Für verschiedene Anzahlen von Beinen und Geschwindigkeiten ergeben sich naturgemäß unterschiedliche Gangarten. Abbildung 2.1 zeigt Gangarten, die sich speziell auf Hunde beziehen. Die Analyse der Gangarten stammt aus "Canine Gait Analysis" [Carr und Dycus, 2016]. Die Art der Darstellung basiert auf dem Buch "The Exterior of the Horse" [Goubaux und Barrier, 1892], welches sich unter anderem mit dem Laufverhalten von Pferden beschäftigt.

- **Schritt**: Zwei oder Drei Beine haben gleichzeitig Kontakt zum Boden.
- **Trott**: Je zwei diagonal zueinander liegende Beine haben Bodenkontakt, beim Wechsel der Beinpaare gibt es eine Schwebephase. Trott ist die effizienteste der Gangarten von Hunden.
- **Passgang**: Jeweils die beiden Beine einer Seite haben Bodenkontakt, beim Wechsel gibt es eine Schwebephase. Passgang kommt nur selten bei Hunden vor.
- **Galopp**: Je zwei Beine hinten und danach vorne berühren leicht versetzt nacheinander den Boden, ein Teil der Zeit sind alle Beine in der Luft.

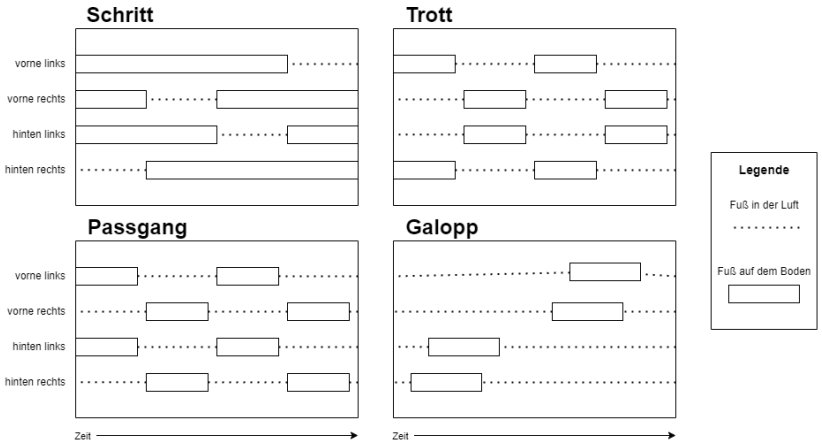


Abbildung 2.1: Beispielgangarten für Hunde (Schritt, Trott, Passgang, Galopp)

In der Abbildung 2.1 kann der Kontakt der Füße mit dem Boden über die Zeit für unterschiedliche Gangarten abgelesen werden. Die gezeigten Abläufe wiederholen sich dabei zyklisch.

2.2 ROS

Das Robot Operating System (ROS) ist ein Framework für die Entwicklung von Robotern. Es wurde ab 2007 im Rahmen des Stanford Artificial Intelligence Robot (STAIR) Projektes der Universität Stanford entwickelt. Seit 2012 befindet es sich unter der Aufsicht der Open Source Robotics Foundation, welche das Projekt als Open-Source-Projekt weiterführt. Das Ziel von ROS ist es, eine einheitliche Umgebung für die Entwicklung von Software in der Robotik zu schaffen.

Unter anderem stellt ROS eine Kommunikationsschicht auf Publish-Subscribe-Basis zwischen Prozessen bereit. Zudem existieren bereits große Mengen Open-Source-Pakete, die unterschiedlichste Aufgaben wie unter anderem Navigation, Kartierung und Pfadplanung beinhalten. Eine Übersicht der Pakete liefert die ROS Paketbibliothek [Open Robotics, 2025b]. Die zugänglichen Ressourcen für quadrupede Roboter sind jedoch begrenzt und sehr spezialisiert. Aus diesem Grund wird für diese Arbeit von der Hardwareabstraktion bis hin zum Laufen des Roboters nur bei allgemeineren Problemen auf öffentliche Ressourcen zurückgegriffen.

Um die grundlegenden Konzepte in ROS zu verstehen, werden an dieser Stelle einige Grundbegriffe aus der ROS-Dokumentation [Open Robotics, 2025a] vorgestellt.

- **Package**: Software in ROS wird in Packages organisiert. Ein Package kann unter anderem Nodes, Messages und Services enthalten.
- **Node**: Eine Node ist ein Prozess, der eine bestimmte Aufgabe erfüllt. Sie kommuniziert mittels Messages über Topics und kann Services und Parameter anbieten und nutzen.
- **Message**: Eine Message ist ein definierbarer Datentyp, der unter einer Topic versendet wird.
- **Topic**: Eine Topic ist ein Name, über den Messages gesendet und empfangen werden.
- **Service**: Ein Service ist ein Remote Procedure Call über Messages, bei dem nur der Servicename, nicht aber der genaue Empfänger bekannt ist.
- **Parameter**: Der ROS Parameter Server erlaubt es, für Programme benötigte Parameter mit Namen in ROS zu hinterlegen und abzurufen.
- **Namespace**: In ROS werden Namespaces genutzt, um eine hierarchische Struktur aus allen Ressourcen zu bilden.

Abbildung 2.2: Beispiel einer Publish-Subscribe-Beziehung zweier Nodes aus der ROS Dokumentation [Open Robotics, 2025a].

Abbildung 2.2 stammt aus dem sogenannten Node Graph eines Kapitels der ROS Tutorials. In der Darstellung sind die Nodes `/teleop_turtle` und `/turtlesim` zu sehen. `/teleop_turtle` published eine Message auf der Topic `command_velocity` im Namespace `/turtle1`. `/turtlesim` subscribed auf diese Topic und empfängt somit die Nachrichten des Publishers.

Die Inhalte der Messages können vielfältig sein, so können primitive Datentypen, aber auch Arrays flexibler Größe versendet werden. Inhalte können wie im Beispiel Zielgeschwindigkeiten, aber auch große Datenmengen wie zum Beispiel Bilder sein. Es sollte für jedes Topic jedoch in den meisten Anwendungsfällen nur einen Publisher geben und immer nur eine Art Message pro Topic übertragen werden.

2.3 Gazebo

Gazebo ist eine 3D-Simulationsumgebung, die bereits sehr gut in ROS integriert ist. Hierdurch wird es ermöglicht, Roboter in einer definierbaren Umgebung zu simulieren. Von ROS aus ist es möglich, Aktuatoren anzusteuern. Zudem können Sensordaten in der Simulation erzeugt und an ROS weitergegeben werden. Es existiert bereits eine große Menge an Plugins, die diese Sensoren und Aktuatoren simulieren. Eines dieser Plugins wird in Kapitel 4.1 genutzt, um die Motoren des Roboters zu simulieren.

2.4 Xacro

Für die Simulation in Gazebo muss eine Beschreibung der Robotergeometrie als Unified Robot Description Format (URDF) vorliegen. URDF ist eine Beschreibungssprache, die auf der Extensible Markup Language (XML) aufbaut. Sie unterstützt jedoch keine Variablen, Berechnungen oder Wiederverwendbarkeit. Da die Beschreibung von großen Robotern somit in URDF sehr umständlich ist, kann mittels XML Macros (Xacro) die Beschreibung vereinfacht werden. Die Xacro-Datei wird im Anschluss in eine URDF-Datei umgewandelt.

In einer Xacro oder URDF Datei können unter anderem die Gelenke und Verbindungen zwischen Gelenken definiert werden. Diese nennen sich Links und Joints des Roboters. Für eine Simulation werden zusätzlich Kollisionen und Visualisierungen definiert. Zudem werden die Gazebo-Plugins für Sensoren oder Aktuatoren direkt in die Beschreibung des Roboters mit aufgenommen.

2.5 Entwurfsmuster

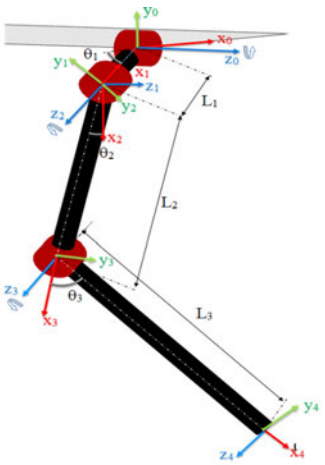
Bei der Implementierung der einzelnen Komponenten werden einige Entwurfsmuster oder auch Pattern verwendet. Diese Muster stammen aus dem Standardwerk "Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software" [Gamma u. a., 2002]. Die für diese Arbeit wichtigsten Entwurfsmuster sind:

- Factory zum Erzeugen von Objektstrukturen
- Composition zum Zusammenfassen von Objekten in hierarchischen Strukturen
- Strategie zum Austauschen unterschiedlicher Implementationen zur Laufzeit
- Adapter zum Adaptieren von Interfaces, die nicht zueinander passen

2.6 Denavit-Hartenberg-Konvention

Ein wichtiges zu lösendes Problem für Roboter ist die Berechnung der Position des Endeffektors. Hierbei werden die Gelenkwinkel des Roboters genutzt, um nacheinander die Positionen der Gelenke bis hin zum Endeffektor zu berechnen. Klassischerweise wird eine Matrixtransformation genutzt, um die Positionen zu berechnen [Siciliano und Khatib, 2008, p.13]. Diese Berechnung nennt sich auch Forward Kinematik. Die Menge an Parametern, die benötigt werden, um die Transformationen für eine kinematische Kette zu definieren, ist jedoch sehr groß. Aus diesem Grund wurde von Jacques Denavit und Richard S. Hartenberg eine Methode entwickelt, für Lower-Pair-Gelenke [Siciliano und Khatib, 2008, p.19]. die Transformationen zu vereinfachen. Zu den Lower-Pair-Gelenken gehören auch Revolute (Rotation) und Prismatic (Translation). Die hierfür verwendeten Parameter nennen sich Denavit-Hartenberg (DH)-Parameter [Siciliano und Khatib, 2008, p.23].

In der Arbeit "Inverse Kinematic Analysis of a Quadruped Robot" [Sen u. a., 2017] wurde eine Kinematik auf Basis der DH Konvention entwickelt. Der in der Veröffentlichung verwendete Roboter ist vom Aufbau der Kinematik bis auf die Maße identisch mit dem in dieser Arbeit entwickelten Roboter. Die DH-Parameter sind in Abbildung 2.3 dargestellt und können mit der im "Handbook of Robotics" [Siciliano und Khatib, 2008, p.23] beschriebenen Methode für die Berechnung der Transformationen genutzt werden.



Link	a_i	α_i	d_i	θ_i
0-1	0	L_1	0	θ_1
1-2	$-\pi/2$	0	0	$-\pi/2$
2-3	0	L_2	0	θ_2
3-4	0	L_3	0	θ_3

Abbildung 2.3: Darstellung und DH-Parameter für die Kinematik des Quadruped-Beines aus "Inverse Kinematic Analysis of a Quadruped Robot" [Sen u. a., 2017]

2.7 Stabilität und stabile Zyklen

Eines der Grundprinzipien, auf denen das Stehen und Gehen eines Roboters basiert, ist das Stützpolygon. Dieses ist die konvexe Hülle der Kontaktpunkte mit dem Boden. Bei laufenden Robotern sind diese Kontaktpunkte die Füße. Eine Position des Roboters ist statisch stabil, wenn die Projektion des Schwerpunktes (CoM) entlang der Erdanziehung innerhalb des Stützpolygons liegt. Eine Erweiterung dieses Konzepts ist es, dass das Stützpolygon den Zero Moment Point (ZMP) enthält. Der ZMP berücksichtigt neben der Gravitation auch andere Kräfte, die auf den Roboter wirken. Er beschreibt den Punkt, bei dem das resultierende Moment auf den Roboter null ist [Chai u. a., 2022, p.4].

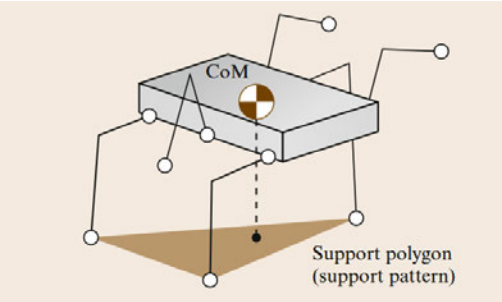


Abbildung 2.4: Darstellung des Stützpolygons und Schwerpunktes eines sechsbeinigen Roboters aus "Handbook of Robotics" [Siciliano und Khatib, 2008, p.378]

Bei einer Beschleunigung des Roboters muss darauf geachtet werden, dass der ZMP innerhalb des Stützpolygons bleibt. Ist dies nicht der Fall, kann der Roboter umkippen.

Die Bewegung eines Roboters ohne statische Stabilität ist bedeutend komplexer. Es lassen sich aber auch statisch nicht stabile Bewegungen finden, die ohne eine Feedbackschleife in einem gewissen Rahmen selbst stabilisierend sind. Dieses Verhalten nennt sich stabiler oder semistabiler Zyklus. Hierbei handelt es sich um Grenzzyklen. Ein Grenzzyklus ist eine geschlossene Kurve, in einem Phasenraum, die periodisch durchlaufen wird [Adamy, 2009, p.15].

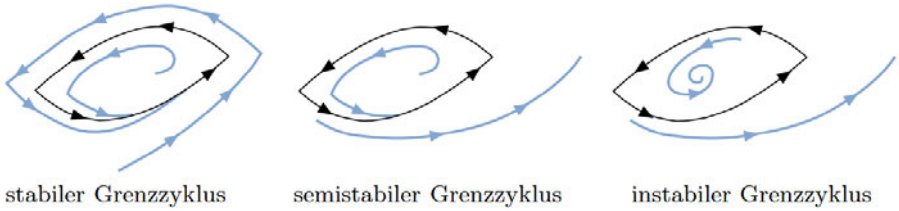


Abbildung 2.5: Darstellung eines Grenzzyklus in einem Phasenraum aus "Nichtlineare Regelung" [Adamy, 2009, p.16]

Die Abbildung 2.5 zeigt einen stabilen, semistabilen und instabilen Grenzzyklus in einem zweidimensionalen Phasenraum. Hierbei bewegt sich das System entlang der Pfeilrichtungen. Zu sehen ist, dass das System bei einem stabilen Grenzzyklus immer zum Zyklus zurückkehrt. Bei einem semistabilen Zyklus kehrt das System nur unter bestimmten Bedingungen zurück. Bei einem instabilen Grenzzyklus entfernt sich das System schon bei kleinsten Abweichungen vom Zyklus.

Für die in dieser Arbeit durchgeführten Untersuchungen von Gangarten besteht der Phasenraum aus den Abweichungen der Position des Roboters von einer Referenzposition [Siciliano und Khatib, 2016, p.1210].

3 Hardware und Hardwareansteuerung

3.1 Übersicht über den Roboter

Der für diese Arbeit weiterentwickelte vierbeinige Roboter besteht aus 12 Gelenken, 3 pro Bein. Die Beine tragen ein starres Chassis, auf dem unterschiedlichste Komponenten angebracht werden können. Die gesamte Konstruktion ist teils aus PETG 3D-Gedruckt und teils mittels einer Portalfräse aus Aluminium gefräst. Für die Versuche wird der Roboter in einen Stahlrahmen eingehängt. Somit können Tests durchgeführt werden, ohne dass der Boden berührt werden kann. Für Tests mit Bodenkontakt wird eine zusätzliche Plattform unter den Roboter gestellt. Somit kann der Roboter in der Halterung eingehängt bleiben und gleichzeitig frei stehen.

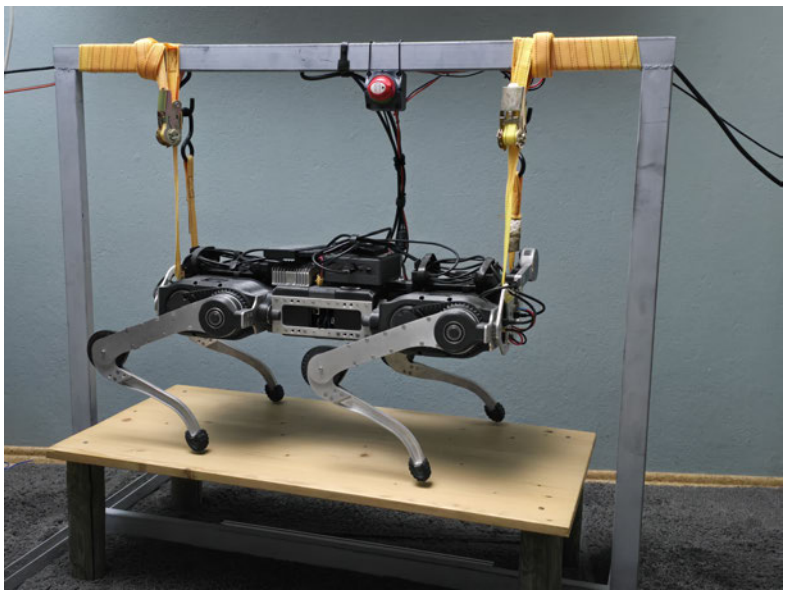


Abbildung 3.1: Foto des Roboters in einer Halterung mit der Plattform für Tests mit Bodenkontakt

3.2 Sicherheitsanmerkungen

Die gesamte Boardelektronik befindet sich mit 44,4V, 24V und 19V DC nach DIN EN 61140 noch in der Schutzklasse 3 (Schutzkleinspannung, <50V AC und <120V DC). Dennoch geht von den Batterien bei fehlerhaftem Laden und Entladen, sowie durch mechanische Einwirkungen ein Brandrisiko aus. Die Mechanik der verwendeten Plattform ist nicht fingersicher gebaut und nicht als kollaborativer Roboter gedacht. Demnach können die Bewegungen der Motoren mit entsprechenden Übersetzungsverhältnissen zu Verletzungen führen. Daher werden alle Experimente an der Hardware mit genügend Sicherheitsabstand zu Menschen und unter Brandschutzmaßnahmen durchgeführt. Zusätzlich befindet sich ein Not-Aus-Schalter direkt über dem Roboter, der die Versorgungsspannung der Motoren unterbricht.

3.3 Bauteile und Komponenten

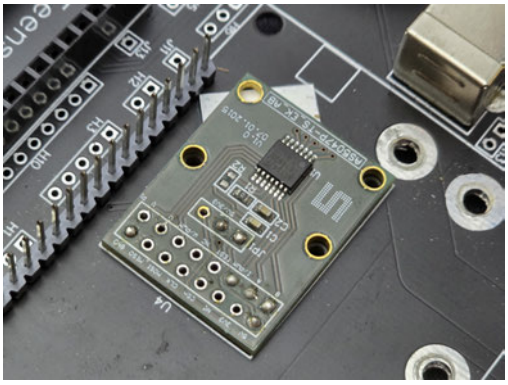
Bei mobilen Robotern gibt es eine Vielzahl von Bauteilen mit unterschiedlichen Anforderungen. Um eine Übersicht über die Komponenten und Entwurfsentscheidungen zu geben, werden in diesem Abschnitt einige wichtige Komponenten sowie die Entscheidungen für deren Verwendung beschrieben. Diese Liste ist nicht erschöpfend, sondern soll nur einen Überblick über einige der Entscheidungen vermitteln.

3.3.1 Boardcomputer

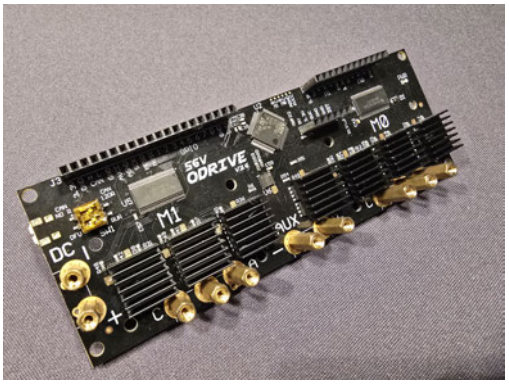
Als Boardcomputer wird ein Intel NUC 10i5 verwendet (Abbildung 3.2a). Das auf dem NUC laufende Betriebssystem ist Ubuntu 20.04 Focal Fossa, auf dem ROS in der Noetic distribution installiert ist. Der verbaute Intel Core i5-10210U Prozessor ist mit 8 Threads und maximal 4,2 GHz leistungsstark genug, um auch die Simulation des Roboters und die eigentliche Steuerung laufen zu lassen. Zudem ist der NUC mit einem TDP von 25 Watt auch für akkubetriebene Anwendungen geeignet.

(a) Intel NUC 10i5 Computer

(b) Intel Realsense T265 Tracking Kamera



(c) AMS-AS5047P Encoder



(d) ODrive 3.6 Umrichter

(e) Turnigy Multistar 9235-100kV Motor

(f) Batterien mit Ladegerät

Abbildung 3.2: Fotos unterschiedlicher im Roboter verbauter Komponenten

3.3.2 Tracking Kamera

Um die Position des Roboters im Raum festzustellen, ist eine Intel RealSense T265 Tracking-Kamera in der Front des Roboters verbaut. Diese nutzt zwei Weitwinkelkameras sowie eine Inertial Measurement Unit (IMU) zur Feststellung der Position (Abbildung 3.2b).

3.3.3 Encoder

Die verwendeten Encoder sind Halleffekt ABI Encoder mit dem Namen AMS-AS5047P (Abbildung 3.2c). Diese geben bei Drehungen Impulse auf zwei Kanälen (A und B) aus, welche um 90 Grad phasenverschoben sind. Dies ermöglicht die Bestimmung der Drehrichtung anhand der Reihenfolge der Pulse. Zudem kann der Winkel der Drehung durch die Anzahl der Pulse bestimmt werden. In einer Umdrehung des Motors werden 4096 Pulse ausgegeben. Zudem wird einmal pro Umdrehung ein Index (I) Signal ausgegeben, das die Feststellung der absoluten Position an dieser Stelle ermöglicht. Die Entscheidung für diese Encoder ist ein Kompromiss zwischen Kosten und Anforderungen.

Optimal für diese Anwendungen wären Absolute-Multiturn-Encoder, welche die Position über mehrere Umdrehungen auch im ausgeschalteten Zustand erfassen können. Aufgrund der hohen Kosten und großen Bauformen wurden diese jedoch nicht verwendet. Die daraus resultierende Notwendigkeit von Kalibrierungen werden in Abschnitt 5.2 genauer beschrieben.

Alternativ könnten kostengünstige optische ABI Encoder verwendet werden, die in ihrer Bauform jedoch ebenfalls sehr groß und damit nicht geeignet sind.

3.3.4 Umrichter

ODrive ist ein Open-Source-Projekt von ODrive Robotics. Es handelt sich um eine kostengünstige und kompakte Lösung, um BLDC Motoren anzusteuern. Im Roboter sind 6 ODrive 3.6 Umrichter verbaut (Abbildung 3.2d). Im Gegensatz zu klassischen BLDC-Controllern ist ODrive auf einen Closed-Loop-Betrieb ausgelegt. Damit können die Motoren auch im Stillstand ein hohes Drehmoment erzeugen. Hierfür wird ein Encoder benötigt, welcher die Position des Motors erfasst.

Ein alternativer Umrichter für BLDC-Motoren ist beispielsweise ein VESC, der in der Arbeit "Development of Open-Source Motor Controller Framework for Robotic Applications" [Choi, 2020] mit weiteren Umrichtern verglichen wird. Da diese jedoch nicht auf Closed-Loop-Betrieb ausgelegt sind, sind sie für diese Anwendung nicht ohne weiteres geeignet. Zudem gibt es industrielle Umrichter von Herstellern wie Siemens, ABB oder SEW, welche in der Regel sehr teuer und groß sind. Damit sind sie meistens für die mobile Robotik ungeeignet.

3.3.5 Motoren

Die verwendeten Motoren sind Turnigy Multistar 9235-100kV BLDC-Motoren (Abbildung 3.2e). Diese zeichnen sich durch einen niedrigen kV-Wert (rpm/V) aus, sodass sich bei $44,4\,V$ Versorgungsspannung maximale Drehzahl von $44400\,rpm$ ergibt. Zudem kann laut Tests von ODrive Robotics aus dem Stand ein Drehmoment von $4.71\,Nm$ erzeugt werden [Odrive Robotics, 2021]. Aufgrund der beschränkten Auswahl von Motoren, die ein hohes Drehmoment bei niedrigen Drehzahlen erzeugen können, gab es zur Zeit der Konstruktion der Hardware keine Alternativen zu den verwendeten Motoren, die nicht bedeutend teurer sind. Mittlerweile gibt es Nachbauten der im Mini Cheetah verwendeten Motoren [Katz, 2018]. Diese könnten eine Alternative zu den Turnigy Motoren sein und würden ebenfalls die Umrichter und Encoder ersetzten.

3.3.6 Motor Hubs

Ein Grundkonzept des Roboters sind die sogenannten "Motor Hubs" (Abbildung 3.3 und 3.4). In diesen werden die für den Betrieb zweier Motoren notwendigen Bauteile in einer Einheit zusammengefasst. Das Hauptziel der Hubs ist es, den Entwicklungsaufwand der Hardware zu reduzieren. Es werden 6 Hubs verbaut, die mechanisch, elektronisch und auch in der Software gleich aufgebaut sind. Sie unterscheiden sie sich nur durch die Seriennummern der Umrichter und die Einbauweise im Roboter.

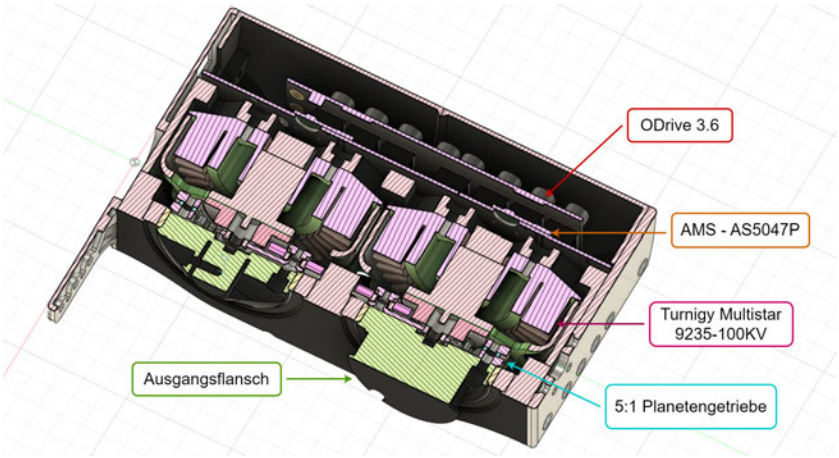


Abbildung 3.3: Querschnitt eines Motor Hubs mit Beschriftung der Komponenten

Jeder Motor Hub besteht aus einem ODrive 3.6, welcher als Umrichter je zwei Turnigy Multistar 9235-100kV BLDC-Motoren versorgt. Mittels der AMS-AS5047P Encoder bilden diese drei Komponenten ein Closed-Loop-System. Dieses ermöglicht eine genaue Positionierung des Motors auch unter Last. Zudem besitzen die Hubs die Möglichkeit, die Motoren aktiv zu kühlen. Zur Vereinfachung der Verkabelung ist ein PCB verbaut, das sich im hinteren Bereich des Hubs befindet. Die Motoren haben eine maximale Drehzahl von $44.400\,rpm$. Da für diese Anwendung nur eine geringere Drehzahl benötigt wird, ist in den Hubs ein Planetengetriebe mit einer Untersetzung von $5/1$ integriert. Dies reduziert die maximale Drehzahl auf $8880\,rpm$ oder auch 148 Umdrehungen pro Sekunde, welche wiederum in der Software auf 40 Umdrehungen pro Sekunde limitiert sind. Eine weitere Reduktion der Drehzahl findet außerhalb der Hubs statt.

Nach außen haben die Hubs einen USB-B Anschluss für die Kommunikation mit dem Boardcomputer und einen XT60 Anschluss für die Versorgungsspannung.

Für die Hubs werden zwei unterschiedliche Konfigurationen in Erwägung gezogen. Bei der ersten wird der ODrive über UART mit einem Teensy 4.0 Mikrocontroller verbunden, der im Motor Hub eingebaut ist. Dieser Teensy übernimmt die Kommunikation mit ROS über eine rosserial [ROS device drivers, 2025] Node, die es erlaubt, die normalen ROS-Messages und Services über Serial einem Mikrocontroller zur Verfügung zu stellen. Außerdem ermöglicht der Mikrocontroller eine Statusmitteilung der Motoren über LEDs, die an der Ober- und Unterseite des Hubs angebracht sind.

In der zweiten Konfiguration werden die Teensys aus der Kette eliminiert. ODrive-Umrichter bieten ein sogenanntes Native Protocol, für das es eine durch den Python Package Manager erhältliche Python-Library gibt, die die ODrive-Steuerung über ein Python-Dictionary ermöglicht. Diese Konfiguration ist jedoch nicht so flexibel wie die erste, da der Teensy aus der Kommunikation mit dem Boardcomputer ausgeschlossen wird. Zudem funktionieren die LEDs und die Lüfter nicht mehr.

Der ausschlaggebende Punkt für die Wahl des Ansatzes ist, dass mit dem Teensy die Boardspannung nur bei maximal 24V liegen darf. In der zweiten Konfiguration kann die Spannung auf 44.4V angehoben werden, da die Komponenten die diese Spannung nicht vertragen, nicht mehr verbaut sind. Somit wird das maximale Drehmoment der Motoren erhöht. Eine weitere Revision des PCBs im Hub kann auch die anderen Komponenten wieder nutzbar machen.

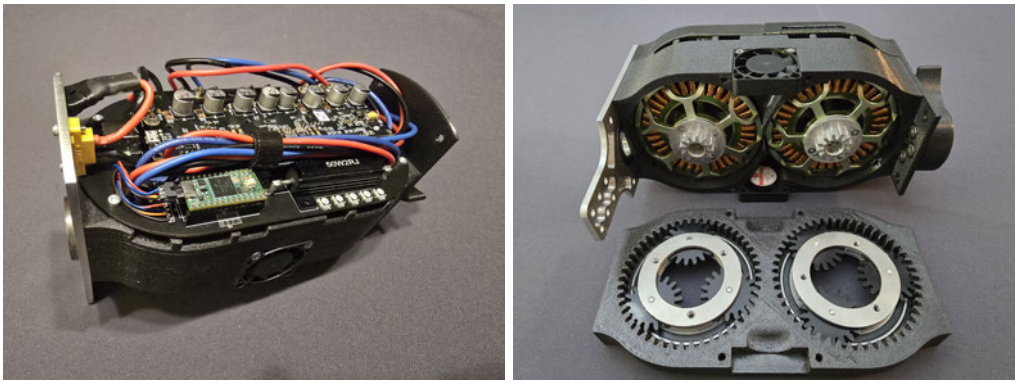


Abbildung 3.4: Fotos eines Motor Hub mit verbautem Teensy Mikrocontroller

3.3.7 Energieversorgung

Die Versorgung der Boardelektronik wird durch zwei 6S Lithium-Polymer-Akkus bereitgestellt (Abbildung 3.2f). Durch die Akkus entsteht eine Nennspannung von $44,4\,V$, diese wird direkt zur Versorgung der Umrichter verwendet. Der verbaute Boardcomputer wird mittels eines Step-Down-Converters auf 24V und 19V versorgt. Zum Schutz der Akkus sind diese mit einer $120\,A$ und die Umrichter je mit einer $20\,A$ Sicherung abgesichert. Eine Übersicht über die Versorgung gibt Abbildung 3.5.

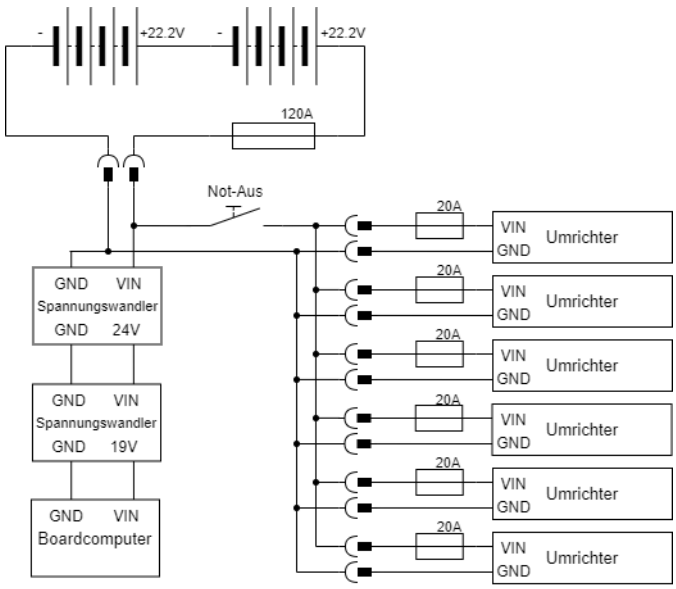


Abbildung 3.5: Übersicht über die Versorgung des Roboters

3.3.8 Beine

Die Beine sind nach ihrer Position benannt: Front (F) oder Rear (R) gefolgt von Left (L) oder Right (R). Die Achsen sind nach ihrem Bein und der Position in ihrer kinematischen Kette benannt. Zum Beispiel ist FL3 somit am Bein vorne links das unterste Gelenk.

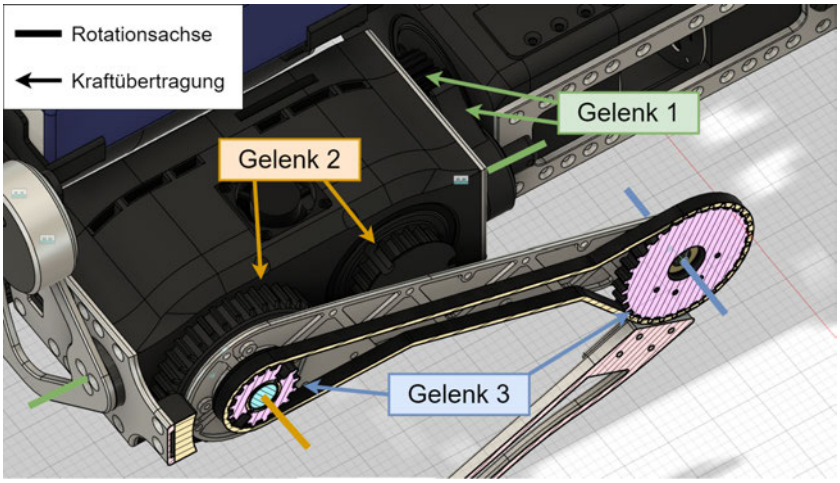


Abbildung 3.6: Übersicht über den Aufbau des Beines FL mit besonderem Augenmerk auf die Kraftübertragung von den Motoren zu den Gelenken

Das Design der Beine muss sich an die Motor Hubs anpassen. Da jeder Hub zwei direkt nebeneinander liegende Motoren antreibt, ist es nicht möglich, dass die Motoren ihre Gelenke, wie beispielsweise bei ANYmal von der ETH Zürich, direkt antreiben. Aus diesem Grund wird das Gelenk 3 über eine Achse angetrieben, welche es abhängig von der Position des Gelenk 2 macht. In Abbildung 3.6 sind die Gelenke mit den benötigten Komponenten zur Kraftübertragung von den Motoren dargestellt. Auf die Folgen dieser Abhängigkeit zwischen den Gelenken wird im Rahmen der Berechnung der Achspositionen eingegangen (Abschnitt 5.1). Der Vorteil dieses Ansatzes ist, dass die schweren Motoren nah am Chassis angebaut werden können. Somit müssen sie bei einer Bewegung der Beine nicht mit beschleunigt werden.

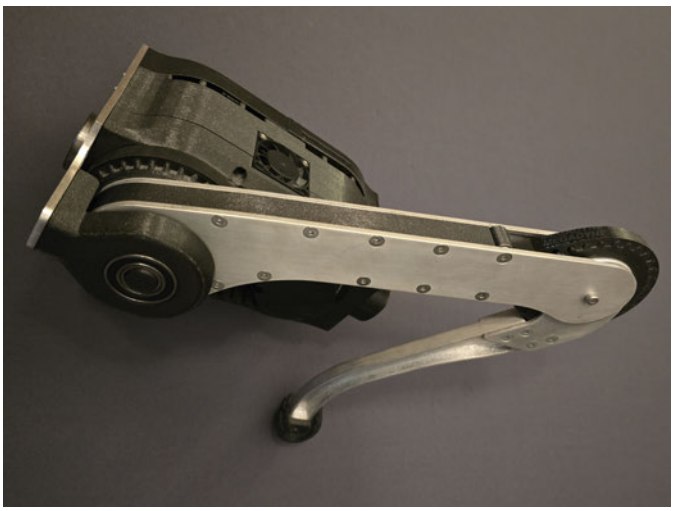


Abbildung 3.7: Foto eines Hub mit angebautem Bein

3.3.9 Nutzereingaben

Zur Steuerung des Roboters wird je nach Anwendung eine 3Dconnexion SpaceMouse oder ein Xbox One Controller verwendet (Abbildung 3.8). Die SpaceMouse ermöglicht eine erleichterte Steuerung des Roboters in 6 Freiheitsgraden (DoF). Der Xbox One Controller ist intuitiver für Richtungsvorgaben beim Laufen. Die unterstützten Controller lassen sich jedoch auch im Nachhinein durch die in Kapitel 7 vorgestellte Architektur leicht erweitern.

Abbildung 3.8: Xbox One Controller und 3Dconnexion SpaceMouse

3.4 Motoransteuerung

Eine der grundlegendsten Vorgehensweisen bei der Entwicklung der Software für technische Systeme ist das Erschaffen von Abstraktionen. Diese erlauben es, teilweise komplexe Probleme auf eine einfachere Schnittstelle zu reduzieren. In diesem Kapitel wird die Abstraktion der Motoransteuerung beschrieben.

Die Ansteuerung der Motoren erfolgt gemäß der gewählten Konfiguration für die Motor Hubs über die Kommunikation mit dem ODrive native Protocol. Als Abstraktionsebene wird eine ROS Node mit dem Namen `odrive_ros_bridge` geschrieben, die die Kommunikation mit den ODrives übernimmt.

An die Node gibt es grundlegende Anforderungen, die für die Anwendbarkeit und die Wiederverwendbarkeit wichtig sind.

1. Die Node muss mit mindestens 6 ODrives kompatibel sein.
2. Die ODrives sollen in ROS transparent sein, die Ansteuerung soll ausschließlich an die Motoren gerichtet sein.
3. Einzelne Motoren müssen sich über die Konfiguration ein- und ausschalten lassen.
4. Parameter müssen über Dynamic Reconfigure [ROS core stacks, 2025] einstellbar sein.
5. Alle Funktionalitäten, die für die Motorpositionierung relevant sind, sollen in ROS zugänglich sein.

Anzumerken ist, dass die Bridge speziell für Positionierungsanwendungen entworfen ist. Daher ist es nicht Teil der Anforderungen, alle Funktionalitäten des ODrive in ROS wiederzugeben. Es werden bereits einige Vereinfachungen und Komfortfunktionen von der Bridge übernommen.

```
# MotorTarget.msg

uint8 MOTOR_STATE_IDLE=0
uint8 MOTOR_STATE_ACTIVE=1
uint8 MOTOR_STATE_MOTOR_CAL=2
uint8 MOTOR_STATE_ENCODER_CAL=3

std_msgs/Header header
uint8 motor_state
float32 position
float32 velocity_limit
float32 torque_limit

```

```
# MotorFeedback.msg

uint8 MOTOR_STATE_UNDEFINED=0
uint8 MOTOR_STATE_IDLE=1
uint8 MOTOR_STATE_ACTIVE=2
uint8 MOTOR_STATE_MOTOR_CAL=3
uint8 MOTOR_STATE_ENCODER_CAL=4
uint8 MOTOR_STATE_ERROR=5

std_msgs/Header header
uint8 motor_state
string status_message
float32 position
float32 velocity
float32 torque
bool motor_calibrated
bool encoder_calibrated

```

Abbildung 3.9: `odrive_ros_bridge` Target und Feedback Messages je Motor

Um die Kommunikation zwischen der Bridge und anderen Nodes zu ermöglichen, werden zwei Messages benötigt (Abbildung 3.9). Die Bridge published je Motor eine `MotorFeedback`-Message und subscribed zu der dazugehörigen `MotorTarget`-Message. Namen der zu verwendenden Topics können über den Parameterserver an die Bridge mitgegeben werden. Die Identifikation der ODrives wird durch die Seriennummer vorgenommen, die ebenfalls auf dem Parameterserver abgelegt wird. Durch eine Namenskonvention der Parameter können unterschiedlich viele Seriennummern und Achsnamen angegeben werden.

Dynamic Reconfigure [ROS core stacks, 2025] erlaubt es, Parameter die zur Laufzeit geändert werden sollen, über ROS verwalten zu lassen. Mittels Kommandozeilenbefehlen oder über ein Plugin der ROS-Qt (RQT) Nutzeroberfläche lassen sich diese Parameter ändern. Um Dynamic Reconfigure zu nutzen, müssen alle Parameter in eine `.cfg` Datei eingetragen werden, die mit übersetzt wird. Dies erzeugt jedoch einen Widerspruch zu der Anforderung, dass die Bridge unterschiedliche Anzahlen an Motoren unterstützen

soll. Das DDynamic Reconfigure (Dynamic Dynamic Reconfigure) Paket [PAL Robotics S.L., 2020] erlaubt es, zur Laufzeit eine beliebige Menge an konfigurierbaren Parametern zu erzeugen. Hierbei wird keine `.cfg`-Datei benötigt.

Eine weitere Aufgabe der Bridge ist es, die Achswerte in ROS zu publishen. Auf diese Weise kann die aktuelle Position der Achsen auch aus dem ROS TF-Tree ausgelesen werden. Der TF-Tree ist eine in ROS standardisierte Methode, um Transformationen zwischen Koordinatensystemen zu verwalten. Nodes können benannte Koordinatensysteme relativ zu anderen Koordinatensystemen veröffentlichen. Eine jede Node kann dann die Transformationen zwischen den Koordinatensystemen abfragen. Genauere Informationen zum TF-Tree sind in der ROS Dokumentation zu finden [Open Robotics, 2025a].

3.5 Motorparametrierung

Die ODrives müssen parametriert werden. Hierzu gehören Einstellungen wie die Spezifikation der Umrichter und Motoren, aber auch die Einstellung der integrierten Regler. Eine Übersicht der Parameter findet sich in Tabelle 3.1. Das Ziel der Parametrierung ist der Schutz der Hardware und das möglichst schnelle Erreichen eines neuen Sollwerts, ohne dabei zu überschwingen. Außerdem soll bei Krafteinwirkungen auf den Motor die eingestellte Position schnell wieder erreicht werden.

Parameter	**Beschreibung**	**Wert**
`current_lim`	Strombegrenzung in A	30
`vel_limit`	Geschwindigkeitsbegrenzung in $turns/s$	40
`vel_gain`	Geschwindigkeitsregler Proportionalanteil in $Nm/(turn/s)$	TBD
`vel_int_gain`	Geschwindigkeitsregler Integralanteil in $Nm/((turn/s) \cdot s)$	TBD
`pos_gain`	Positionsreglerverstärkung in $(turn/s)/turn$	TBD

Tabelle 3.1: Übersicht über wichtige ODrive Parameter

Die Strom- und Geschwindigkeitslimits werden durch die Hardware vorgegeben. Für die Einstellung der Regler bietet ODrive ein Konsolentool an, mit dem man die Werte setzen und die Motoren bewegen kann, um die Resultate zu sehen. Da die `odrive_ros_bridge` dieselbe Funktion bietet, wird das Konsolentool allerdings nicht verwendet.

ODrive besitzt für die Positionsregelung drei kaskadierende Regler. Der erste ist der Motorstromregler, dieser wird vom Geschwindigkeitsregler angesteuert, welcher wiederum vom Positionsregler angesteuert wird (Abbildung 3.10). Ein großer Vorteil von kaskadierenden Reglern in der Antriebstechnik ist, dass die Ausgänge der Reglerstufen auf vorgegebene Limits begrenzt werden können. Auf diese Weise kann die Mechanik vor Schäden geschützt werden. Für genauere Informationen zu kaskadierenden Reglern und deren Parametrierung sei auf das "Taschenbuch der Regelungstechnik" [Lutz und Wendt, 2019] verwiesen.

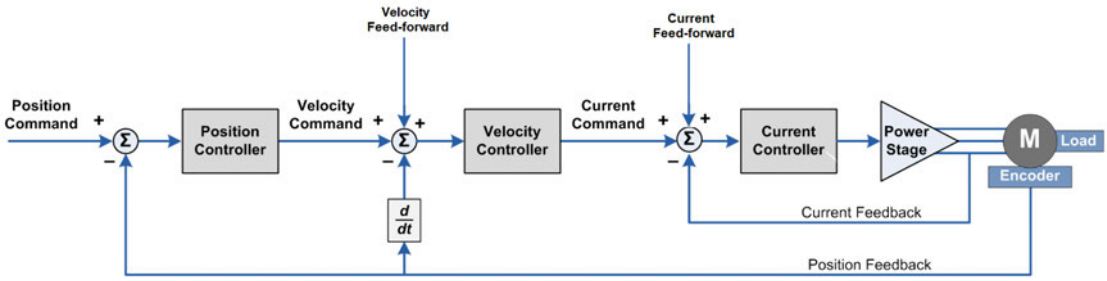


Abbildung 3.10: Kaskadierende Reglerstruktur des ODrives aus der ODrive Dokumentation [Odrive Robotics, 2021]

Der Positionsregler ist ein P-Regler. Der Geschwindigkeitsregler und der Stromregler sind PI-Regler. Parameter des Stromreglers werden für den normalen Betrieb vom ODrive selbst bestimmt. Die verbleibenden Parameter sind in Tabelle 3.1 aufgelistet.

Die Werte der Regler werden anhand eines Verfahrens bestimmt, welches in der ODrive-Dokumentation zur Einstellung von Reglern [Odrive Robotics, 2021] beschrieben ist. Dieses Verfahren ist an die Einstellung von P und PI Reglern nach Ziegler und Nichols [Lutz und Wendt, 2019, p. 494] angelehnt.

3.5.1 Retrospektive auf einen Fehler in der Parametrierung

An dieser Stelle der Arbeit ist ein Fehler aufgetreten, der zu einem späteren Zeitpunkt einen der Umrichter zerstört zu haben scheint.

Neben dem Maximalstrom der Motoren über den Parameter `current_lim`, gibt es den Parameter `current_lim_margin`. Ist `current_lim` erreicht, baut der Motor kein

weiteres Drehmoment mehr auf. Werden beispielsweise durch Rekuperation `current_lim` + `current_lim_margin` überschritten, wird der Motor aus Sicherheitsgründen abgeschaltet. Die Summe der beiden Werte nennt sich `requested_current_range`. Um den Strom durch die Motoren messen zu können, werden die in die Umrichter eingebauten Verstärker der Shunts so eingestellt, das sie den gesamten `requested_current_range` Bereich abdecken.

Aus einem unbekannten Grund wurde `requested_current_range` bei einem Experiment auf einem der Umrichter überschritten. Dies hat wie erwartet den Fehler `ERROR_CURRENT_SENSE_SATURATION` ausgegeben, der besagt, dass der maximale messbare Bereich der Strommessung überschritten wurde. Nach einem Neustart des Umrichters erwies sich dieser als defekt und es ließ sich keine Kommunikation mehr herstellen.

Um einen solchen Fehler zu vermeiden, sollte der Wert von `current_lim_margin` ebenfalls bei der Parametrierung mit beachtet werden. Hierbei muss die Marge so gewählt werden, dass sie nicht überschritten werden sollte. Ist sie zu hoch gewählt, hat dies jedoch negative Auswirkungen auf die Stromregelung, da die Strommessung durch die Shunts ungenauer wird.

3.6 Untersuchung des Motorverhaltens

3.6.1 Versuchsaufbau

Zur Untersuchung der nun parametrierten Motoren werden zwei Tests durchgeführt. Als Erstes wird die Sprungantwort des Reglers aufgezeichnet. Hierfür werden Sprünge von 30°, 60°, 90° und 120° durchgeführt. Es soll untersucht werden, ob der Regler den Anforderungen gemäß nicht überschwingt und wie lang die Einschwingzeit ist. Die Winkelangaben beziehen sich hierbei auf die Drehung des Hub, der Motor bewegt sich demnach um einen Faktor von 5 weiter.

Danach wird der Regler unter Last getestet. Da die Übersetzung der Antriebe mit 1:5 sehr niedrig ist, ist es möglich, durch das Drehen des Ausgangsflansches den Motor zu drehen. Dies kann beispielsweise durch Lastwechsel beim Laufen geschehen. Aus diesem Grund werden die Regler auf das Abfangen von Lasten getestet. Dazu werden am Motor über ein Rad mit 20 cm Durchmesser, nacheinander Gewichte von 2 kg und 5 kg angehängt, um die Ergebnisse vergleichen zu können (Abbildung 3.11). Diese Gewichte werden mit 250 deg/s

abgelassen, was einer Geschwindigkeit von $0.22\,m/s$ entspricht. Daraufhin werden die Gewichte auf $0\,m/s$ abgebremst.

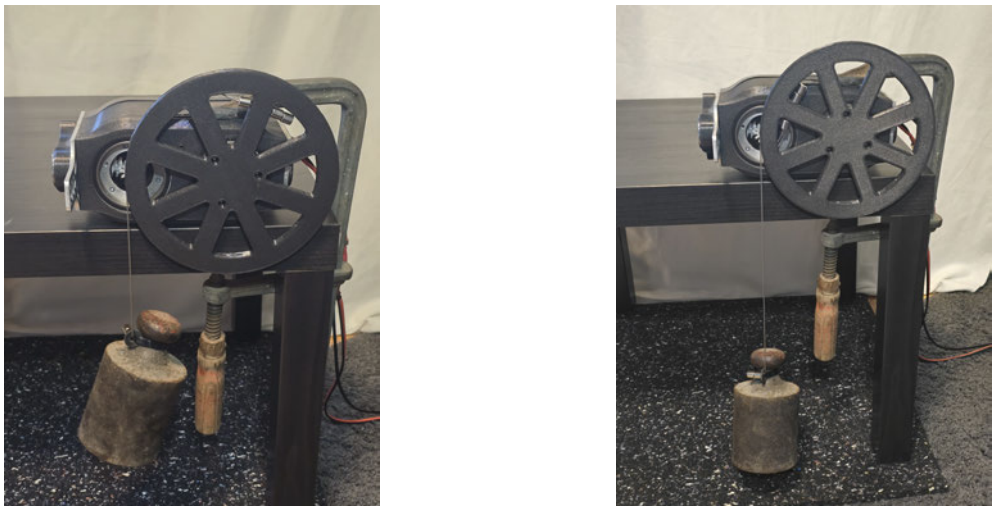


Abbildung 3.11: Testaufbau für Motortests unter Last mit einem Gewicht von $5\,kg$

3.6.2 Ergebnisse

In Abbildung 3.12 sind direkte Sprünge bis zu 120° ohne Last auf den Motor dargestellt. Nach dem Einstellen der Reglerparameter reagiert der Motor in allen getesteten Fällen mit einer Einschwingzeit von $\sim 120\,ms$ ohne Überschwingen.

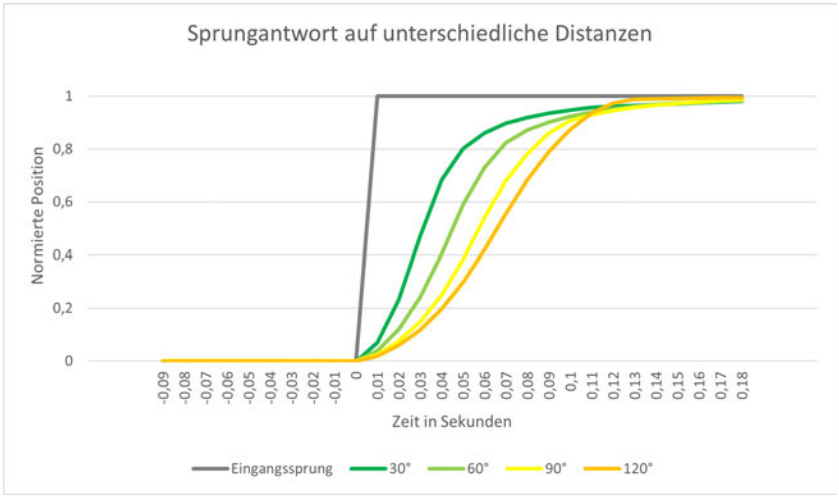


Abbildung 3.12: Sprungantwort des Reglers bei Sprungweiten von 30°, 60°, 90° und 120°.

Bei den Versuchen unter Last fällt auf, dass das Überschwingen und auch die Einschwingzeit bei größeren Lasten zunimmt (Abbildung 3.13). Da der Motor eine maximale Kraft hat, mit der er das Gewicht abbremsen kann, ist dies zu erwarten. Unter einer Last von $5\,kg$ beträgt die Einschwingzeit nach dem Abbremsen der Last $\sim 200\,ms$. Die maximale Kraft wird durch die Strombegrenzung des Motors in den Parametern des Umrichters festgelegt (Tabelle 3.1 und Abbildung 3.10).

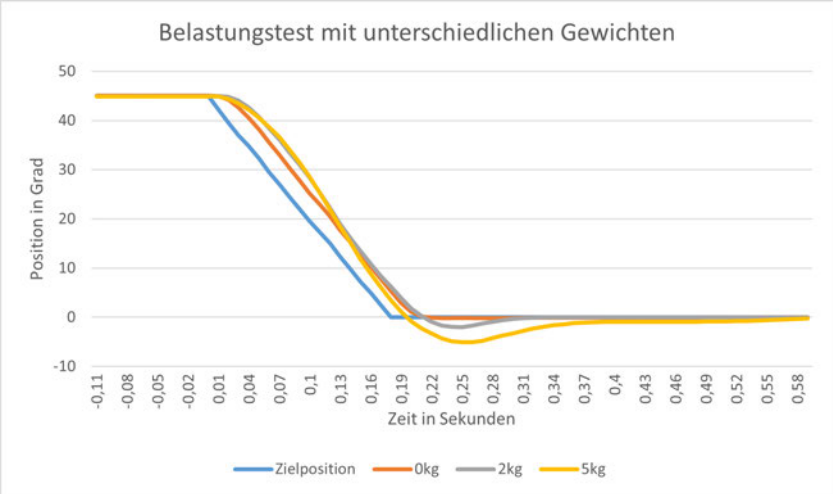


Abbildung 3.13: Genauigkeit des Reglers beim Abfangen unter unterschiedlichen Lasten

Die Motor Hubs sind somit in der Lage, die Motoren schnell und genau zu positionieren, ohne die Belastbarkeit der Motoren oder Umrichter zu überschreiten.

4 Simulation von Motoren

Um den Roboter später simulieren zu können, ist es wichtig, dass als Grundlage die tatsächlichen Bewegungen der Motoren korrekt abgebildet werden können. Die Simulation findet in Gazebo (Abschnitt 2.3) statt. Eine genauere Beschreibung der Simulation und der dafür benötigten Informationen über den Roboter folgt in Kapitel 6. An dieser Stelle soll als Grundlage für die weiteren Versuche ausschließlich sichergestellt werden, dass sich die Motoren der Simulation korrekt verhalten.

4.1 Simulierte Motoren in Gazebo

Die Motoren in Gazebo werden durch das Plugin `libgazebo_ros_control` gesteuert. Dieses Plugin verwendet `ros_control`, um die Steuerung in ROS zu ermöglichen. Hierfür wird ein "Joint State Interface" von Gazebo gepublished, das den aktuellen Zustand des Motors beinhaltet. Zudem wird auf ein "Joint Command Interface" subscribed, das die Zielwerte für das Drehmoment des Motors enthält. Diese beiden Interfaces für `ros_control` der Hardware Interface Layer, ab dem entschieden werden kann, ob eine Simulation oder echte Hardware angesteuert werden soll. Da der Hardware Interface Layer für diesen Roboter bereits gemäß Abbildung 3.9 definiert ist, existiert ein Adapter, der zwischen den beiden übersetzt. Somit können die erweiterten Funktionen wie zum Beispiel die Kalibrierung der Motoren und Encoder, die für die Ansteuerung der ODrives benötigt werden, auf gleiche Weise in der Simulation genutzt werden. Die zusätzlichen Funktionalitäten, die in der Simulation nicht existieren, werden vom Adapter ignoriert oder durch Dummywerte ersetzt.

4.2 Parametrierung simulierter Motoren

Das `libgazebo_ros_control` Plugin enthält einen PID-Regler, um die Position der Motoren zu steuern. Mittels Dynamic Reconfigure lässt sich dieser Regler einstellen. Das Ziel des Reglers ist nicht, ein möglichst schnelles Einschwingen ohne Überschwingen zu erreichen, sondern durch seine Positionsregelung möglichst nah das reale Verhalten des Motors abzubilden. Hierfür wird die im Rahmen der Zielsetzung (Abschnitt 1.2) definierte Toleranz von $\pm 2,5°$ oder $\pm 20\,ms$ zum Verhalten des realen Motors angenommen. Es ist sowohl eine Zeit- als auch eine Winkeltoleranz notwendig. Existierte nur eine zeitliche Toleranz, so wäre das Toleranzfeld im Stillstand des Motors nahezu null. Dies ist unrealistisch, da die Position der Hardware selbst immer um einen gewissen Winkel rauscht. Existierte nur eine Winkeltoleranz, so wäre das Toleranzfeld bei schnellen Bewegungen sehr klein. Entsprechend sind die in Abbildung 4.1 dargestellten Toleranzfelder eine Kombination der beiden Toleranzen.

Da der Regler nicht auf Einschwingzeit oder Stabilität eingestellt werden soll, kann das Verfahren von Ziegler und Nichols nicht angewendet werden. Stattdessen wird der Regler über einfache Annäherung bei unterschiedlichen Bewegungen auf das Verhalten des realen Motors eingestellt.

4.3 Untersuchung simulierter Motoren

4.3.1 Versuchsaufbau

Als Versuchsaufbau wird ein einzelner im Raum fixierter Motor in die Simulation eingefügt. An der Achse des simulierten Motors befindet sich ein kleines Gewicht, das die Trägheit des Motors selbst nachstellen soll.

Um zu testen, ob der Regler die Anforderungen aus Abschnitt 4.2 erfüllt, werden analog zu Abschnitt 3.6 Sprünge von 30°, 60°, 90° und 120° durchgeführt.

Die Untersuchung der simulierten Motoren unter größeren Lasten findet zusammen mit der Untersuchung des Körpers des Roboters in Kapitel 6.3 statt.

4.3.2 Ergebnisse

Wie in Abbildung 4.1 dargestellt, erreicht der simulierte Regler leicht schneller die neue Sollposition als der reale Motor, bleibt jedoch bei allen Tests innerhalb der vorgegebenen Toleranzen.

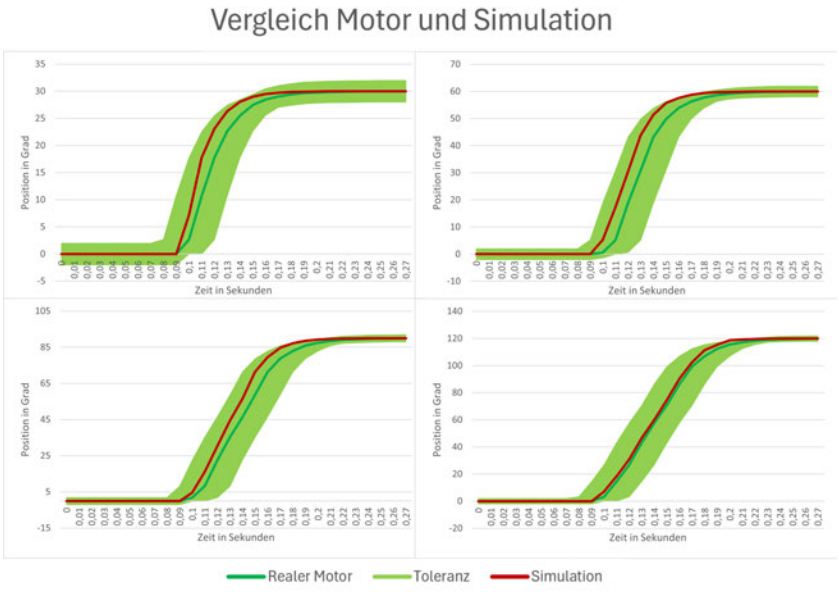


Abbildung 4.1: Sprungantwort mit Toleranz des Reglers bei Sprungweiten von 30°, 60°, 90° und 120°

Durch weitere Annäherungsschritte könnte dieses Verhalten noch verbessert werden. Da sich das System bereits innerhalb der Toleranzgrenzen befindet, wird in dieser Arbeit auf weitere Schritte zur Verbesserung verzichtet.

5 Bewegungskoordination

Dieses Kapitel behandelt die Konzepte, die für die Ansteuerung der Beine in kartesischen Koordinaten benötigt werden. Das Ziel ist es, die Motoren der Beine so anzusteuern, dass diese sich in einer linearen Bewegung von ihrer Startposition zu einer Zielposition bewegen. Hierbei sollen möglichst hohe Geschwindigkeiten erreicht werden, ohne dass die definierten Toleranzen überschritten werden.

Die Betrachtung beschränkt sich auf das Bein vorne rechts, das Verfahren der anderen Beine ist analog.

Für die Verständlichkeit der folgenden Ausführungen werden drei Begriffe voneinander abgegrenzt:

- **Achse**: Die Achsen des Roboters sind die Ausgangsflansche der Motor Hubs. Sie werden als Winkel zur Startposition des Hub angegeben. Die Achsen werden A_1, A_2 und A_3 genannt. Die Nummerierung erfolgt nach der Position in der kinematischen Kette.
- **Gelenk**: Die Gelenke des Roboters werden als Winkel zu ihrem jeweiligen Nullpunkt angegeben. Die Gelenke werden θ_1, θ_2 und θ_3 genannt. Die Nummerierung erfolgt nach der Position in der kinematischen Kette.
- **Position**: Die Position des Fußes ist in kartesische Koordinaten relativ zum Nullpunkt des Beines angegeben. Die Positionen werden x, y und z genannt.

5.1 Berechnung der Achspositionen

Als Eingangswert stellt die `odrive_ros_bridge` die Achswerte A_1, A_2 und A_3. Aus diesen Werten werden die Gelenke θ_1, θ_2 und θ_3 berechnet.

Durch die Mechanik des Roboters ergeben sich die folgenden Übersetzungsverhältnisse:

$$i_1 = 21/40 \tag{5.1}$$
$$i_2 = 20/36 \tag{5.2}$$
$$i_3 = 17/30 \tag{5.3}$$

Da θ_3 mechanisch sowohl von A_3 als auch von A_2 abhängt (Abschnitt 3.3.8), ist hier nicht nur die einfache Übersetzung anzuwenden. Stattdessen muss θ_2 in die Berechnung von θ_3 mit einbezogen werden.

$$\theta_1 = A_1 \cdot i_1 \tag{5.4}$$
$$\theta_2 = A_2 \cdot i_2 \tag{5.5}$$
$$\theta_3 = (A_3 + \theta_2) \cdot i_3 \tag{5.6}$$

Somit ergeben sich für die Berechnung der Achsen die folgenden Gleichungen:

$$A_1 = \frac{\theta_1}{i_1} \tag{5.7}$$
$$A_2 = \frac{\theta_2}{i_2} \tag{5.8}$$
$$A_3 = \frac{\theta_3}{i_3} - \theta_2 \tag{5.9}$$

Die Gelenke sind mechanisch auf bestimmte Winkel beschränkt. Diese sind der Tabelle 5.1 zu entnehmen. Um die Motoren und Umrichter nicht zu beschädigen, sind die Gelenke in der Software auf diese Werte begrenzt.

Achse	Min	Max
θ_1	$-30.0°$	$30.0°$
θ_2	$-1.5°$	$180.0°$
θ_3	$13.5°$	$180.0°$

Tabelle 5.1: Mechanische Limits der Gelenke am Beispiel des Beines vorne rechts

5.2 Kalibrierung der Achspositionen

Einer der Hauptnachteile des verwendeten ABI Encoders ist, dass sie keine Möglichkeit bieten, die absolute Position der Achsen zu bestimmen. Dies führt dazu, dass der Roboter nach jedem Neustart neu kalibriert werden muss. Zudem ist es nicht möglich, mit der aktuellen Hardware zu kontrollieren, ob die Kalibrierung erfolgreich war. Die Lösung für dieses Problem ist, dass nach dem Start des Roboters die Achsen einmal in ihre Endlagen bewegt werden. Zu den Achswerten wird ein Wert addiert, der den Versatz der Startposition des Gelenks zum Roboter-Nullpunkt darstellt. Der Offset wird auf dem ROS-Parameterserver abgelegt, sodass die Kalibrierung nur einmal gemacht werden muss, bis die ODrives oder ROS neu gestartet werden.

5.3 Berechnung der Kinematik

Um die Position des Fußes zu berechnen, wird eine Forward Kinematik und Inverse Kinematik benötigt. Die Forward Kinematik berechnet aus den Gelenkwinkeln und der Geometrie des Roboters die Position eines Endeffektors, in diesem Fall des Fußes. Die Inverse Kinematik berechnet aus der Position des Endeffektors und der Geometrie des Roboters die Gelenkwinkel.

Für die Implementation der Kinematik wird nicht der klassische Ansatz mittels DH-Parameter (Abschnitt 2.6) gewählt. Stattdessen wird die Position durch eine Menge trigonometrischer Gleichungen berechnet, die aus der Robotergeometrie hervorgehen. Dies ist zwar keine universelle Lösung, sie hat aber den Vorteil, dass sich Forward und Inverse Kinematik jeweils sehr einfach voneinander herleiten lassen. Hierbei ist der Winkel, in dem sich der Fuß befindet, nicht relevant, denn es wird ausschließlich die Position des Fußes für das Laufen benötigt.

Für diese Lösung wird mit der Inversen Kinematik begonnen. Die Berechnung wird in zwei Teile aufgeteilt. Hierbei ist die erste Ansicht von hinten auf das Bein und die zweite Ansicht auf die Seite des Beines.

Für die Inverse Kinematik ist die Zielposition x, y, z sowie die Längenkonstanten des Beines L_1, L_2, L_3 gegeben. Gesucht sind die Winkel $\theta_1, \theta_2, \theta_3$.

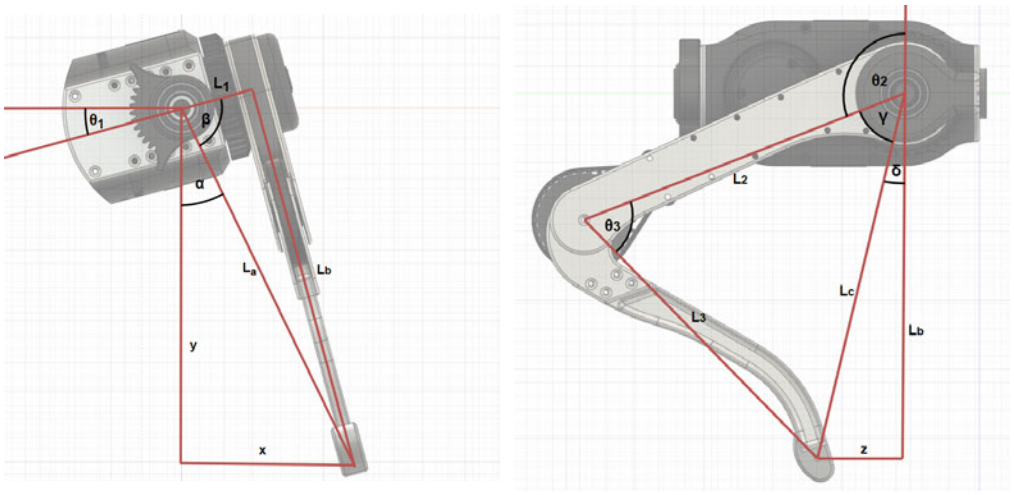


Abbildung 5.1: Aufbau der Kinematik-Berechnung aus der Ansicht von hinten (links im Bild) und seitlich (rechts im Bild)

$$L_a = \sqrt{x^2 + y^2} \tag{5.10}$$

$$\alpha = atan\left(\frac{x}{y}\right) \tag{5.11}$$

$$\beta = acos\left(\frac{L_1}{L_a}\right) \tag{5.12}$$

$$\theta_1 = \frac{\pi}{2} - \alpha - \beta \tag{5.13}$$

$$L_b = \sqrt{L_a^2 - L_1^2} \tag{5.14}$$

$$L_c = \sqrt{L_b^2 + z^2} \tag{5.15}$$

$$\gamma = acos\left(\frac{L_c^2 + L_2^2 - L_3^2}{2 \cdot L_c \cdot L_2}\right) \tag{5.16}$$

$$\delta = atan\left(\frac{z}{L_b}\right) \tag{5.17}$$

$$\theta_2 = \pi + \delta - \gamma \tag{5.18}$$

$$\theta_3 = acos\left(\frac{L_2^2 + L_3^2 - L_c^2}{2 \cdot L_2 \cdot L_3}\right) \tag{5.19}$$

Für die Forward Kinematik werden die Gleichungen umgestellt. In diesem Fall sind die Winkel $\theta_1, \theta_2, \theta_3$ sowie die Längen-Konstanten L_1, L_2, L_3 gegeben. Gesucht ist die Position des Fußes x, y, z.

$$L_c = \sqrt{-(2 \cdot L_2 \cdot L_3 \cdot cos(\theta_3)) + L_2^2 + L_3^2} \tag{5.20}$$

$$\gamma = acos\left(\frac{L_c^2 + L_2^2 - L_3^2}{2 \cdot L_c \cdot L_2}\right) \tag{5.21}$$

$$\delta = \theta_2 + \gamma - \pi \tag{5.22}$$

$$L_b = cos(\delta) \cdot L_c \tag{5.23}$$

$$z = tan(\delta) \cdot L_b \tag{5.24}$$

$$L_a = \sqrt{L_b^2 + L_1^2} \tag{5.25}$$

$$\beta = acos(L_1/L_a) \tag{5.26}$$

$$\alpha = \pi/2 - \theta_1 - \beta \tag{5.27}$$

$$x = sin(\alpha) \cdot L_a \tag{5.28}$$

$$y = cos(\alpha) \cdot L_a \tag{5.29}$$

5.4 Begrenzung der möglichen Konfigurationen

Die Beschreibung und Notation des Konfigurationsraums basiert auf "Handbook of Robotics" [Siciliano und Khatib, 2008, p. 110].

Durch die drei Gelenke pro Bein entsteht ein Konfigurationsraum $\mathcal{C}$ mit drei Dimensionen für jedes Bein. Dieser beinhaltet jede mögliche Konfiguration q, die sich innerhalb der mechanischen Grenzen der Beine (Tabelle 5.1) befindet. Die Menge aller möglichen Positionen, die ein Bein der Geometrie $\mathcal{A}$ einnehmen kann, nennt sich Workspace $\mathcal{W}$ und besteht aus drei räumlichen Dimensionen. $\mathcal{A}(q)$ ist die Position des Beins in $\mathcal{W}$, die durch die Konfiguration q mit der Geometrie $\mathcal{A}$ beschrieben wird.

Um Kollisionen der Füße mit dem Körper des Roboters, sowie mit anderen Beinen zu vermeiden, wird eine Hindernis-Geometrie $\mathcal{O}$ definiert, die $\mathcal{W}$ einschränkt. Der Raum aller erlaubten Konfigurationen $\mathcal{C}_{free}$ ist definiert durch $\mathcal{C}_{free} = \{q \in \mathcal{C} | \mathcal{A}(q) \in \mathcal{W} \wedge \mathcal{A}(q) \notin \mathcal{O}\}$. Alle nicht erlaubten Konfigurationen nennen sich $\mathcal{C}_{obs}$ und sind definiert durch $\mathcal{C}_{obs} = \{q \in \mathcal{C} | q \notin \mathcal{C}_{free}\}$

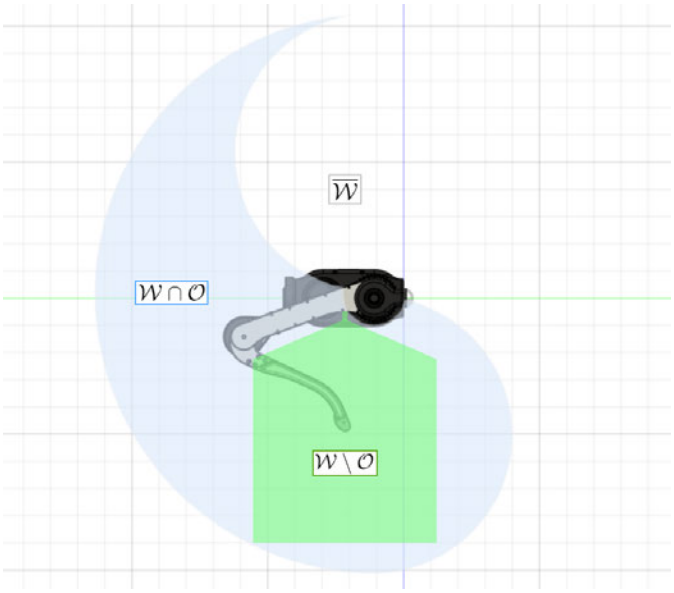


Abbildung 5.2: Seitenansicht der Hindernis-Geometrie (blau) und des durch die Hindernis-Geometrie eingeschränkten Konfigurationsraums (grün)

Kollisionen mit der Umgebung werden im Rahmen dieser Arbeit nicht als Hindernisse angesehen, da Kollisionen mit dem Boden Teil des Laufens sind. Hindernisse, mit denen Kollisionen auftreten können, sind nicht Teil dieser Arbeit. Soll eine Bewegung von einer Konfiguration $q_1 \in \mathcal{C}_{free}$ zu einer Konfiguration $q_g \in \mathcal{C}_{free}$ durchgeführt werden, so muss die gesamte Bewegung in $\mathcal{C}_{free}$ liegen. Es wird also ein Pfad $\tau : [0,1] \rightarrow \mathcal{C}_{free}$ gesucht, sodass $\tau(0) = q_1$ und $\tau(1) = q_g$ gilt.

Soll eine neue Konfiguration angefahren werden, wird eine lineare Bewegung von der Startposition $\mathcal{A}(q_1) \in \mathcal{W} \setminus \mathcal{O}$ zu der Zielposition $\mathcal{A}(q_g) \in \mathcal{W} \setminus \mathcal{O}$ durchgeführt. Ist der Raum $\mathcal{C}_{free}$ im kartesischen Raum konvex, so ist für den linearen Pfad τ sichergestellt, dass $\forall t \in [0,1] : \tau(t) \in \mathcal{C}_{free}$. Dies liegt in der Definition von konvexen Räumen begründet, die besagt, dass die Verbindung zweier Punkte in einem konvexen Raum ebenfalls in diesem Raum liegt.

Ist der Raum nicht konvex, so besteht die Möglichkeit, dass $\exists t \in [0,1] : \tau(t) \in \mathcal{C}_{obs}$. In diesem Fall gibt es grundsätzlich zwei Optionen:

- Der lineare Pfad wird verlassen und ein neuer Pfad wird gesucht, der $\mathcal{C}_{obs}$ umgeht.
- Die Bewegung wird nicht durchgeführt.

Da die Bewegungen der Beine beim Laufen abhängig voneinander sind, kann nicht von dem erwarteten linearen Pfad abgewichen werden. Dieser Fehler kann also nicht über die Pfadplanung im Konfigurationsraum, sondern nur über die Bewegungsplanung behoben werden und die Bewegung kann somit nicht durchgeführt werden.

5.5 Untersuchung der Linearität von Beinbewegungen

Um den berechneten Pfad umzusetzen, gibt es eine wichtige Einschränkung der Hardware, die beachtet werden muss. Die maximale Anzahl an Zielpositionen, die an das Bein übergeben werden können, ist durch die Kommunikationsgeschwindigkeit begrenzt. Eine lineare Bewegung im kartesischen Raum resultiert jedoch nicht in einer konstanten Bewegung der Motoren. Stattdessen resultiert eine Bewegung eines Motors in einer Kreisbahn im kartesischen Raum.

Um die Bewegung der Beine zu linearisieren werden Zwischenziele eingefügt. Zusätzlich besitzen die Umrichter einen Eingangsfilter, der einen weicheren Übergang zwischen den Positionen ermöglicht. Da die Kommunikation zum Umrichter auf 100 Zielpositionen pro Sekunde begrenzt ist, wird die Bandbreite dieses Filters auf $100\,Hz$ gesetzt.

Die theoretisch maximale Geschwindigkeit der Beine kann auf Basis der Robotergeometrie und der Maximalgeschwindigkeiten der Motoren berechnet werden. Da hierbei die mechanischen Eigenschaften des Roboters, sowie das Verhalten der Regler bei Änderungen der Zielposition nicht mit einbezogen werden, ist eine solche Berechnung nicht zielführend. Stattdessen wird die Maximalgeschwindigkeit experimentell bestimmt. Hierfür wird die Geschwindigkeit iterativ verringert, bis der Fehler die Toleranzgrenze unterschreitet.

5.5.1 Versuchsaufbau

Zur Überprüfung der Linearität und Feststellung der Maximalgeschwindigkeit werden Bewegungen in unterschiedlichen Geschwindigkeiten durchgeführt. Hierbei muss es eine Toleranz vom linearen zum tatsächlichen Pfad geben. Je kleiner der erlaubte Fehler ist, desto genauer ist die Umsetzung der Bewegung. Die Maximalgeschwindigkeit sinkt jedoch, da für eine gegebene Bewegung mehr Zwischenziele benötigt werden. Die tolerierte Abweichung zum Pfad beträgt gemäß Abschnitt 1.2 $20\,mm$.

Für diesen Versuch werden Bewegungen verwendet, die sowohl die Grenzen des Workspaces, als auch die erwarteten Bewegungen des Roboters abdecken. Es wird eine vertikale Bewegung durchgeführt, die die obere und untere Grenze des zu erwartenden Arbeitsbereiches erreicht. Zudem wird eine diagonale Bewegung in einer konstanten Höhe durchgeführt. Während der Bewegung wird die Position des Fußes aufgezeichnet, die durch die vom Umrichter gemessenen Motorpositionen berechnet wird.

Zur Evaluierung der Bewegung wird die kürzeste Distanz zwischen der Position des Fußes und dem Zielpfad berechnet. Die maximal erlaubte Abweichung ergibt somit einen Zylinder entlang des Zielpfades. In Abbildung 5.3 sind zur Veranschaulichung des Versuchs zwei diagonale Bewegungen im kartesischen Raum dargestellt, wobei eine innerhalb der Toleranzgrenze liegt und die andere nicht.

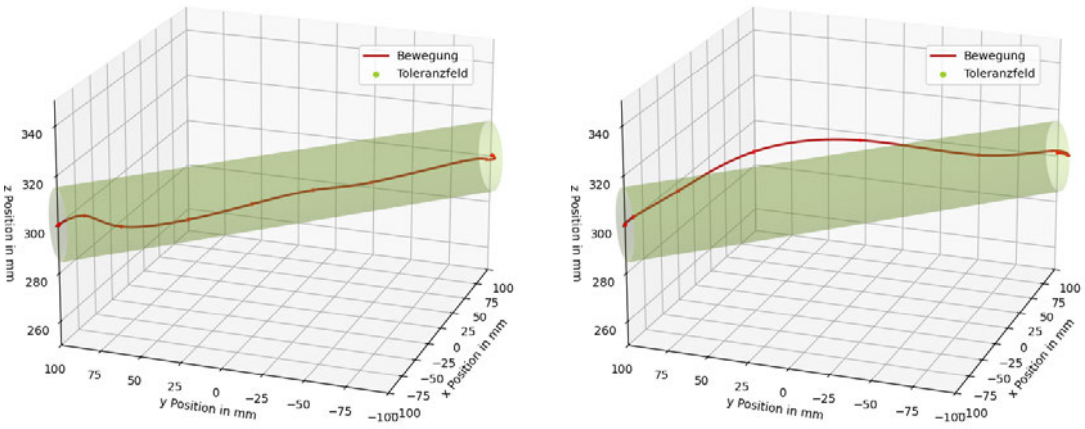


Abbildung 5.3: Darstellung einer diagonalen Bewegung bei maximal $2000\,mm/s$ (links) und $3000\,mm/s$ (rechts) mit einer Toleranz von $20\,mm$

Der für die Feststellung der Maximalgeschwindigkeit durchgeführte Versuch besteht aus zwei Bewegungen, die mit je fünf Geschwindigkeiten durchgeführt werden. Jeder Durchlauf wird fünfmal wiederholt und die maximale, minimale und durchschnittliche Abweichung berechnet.

5.5.2 Ergebnisse

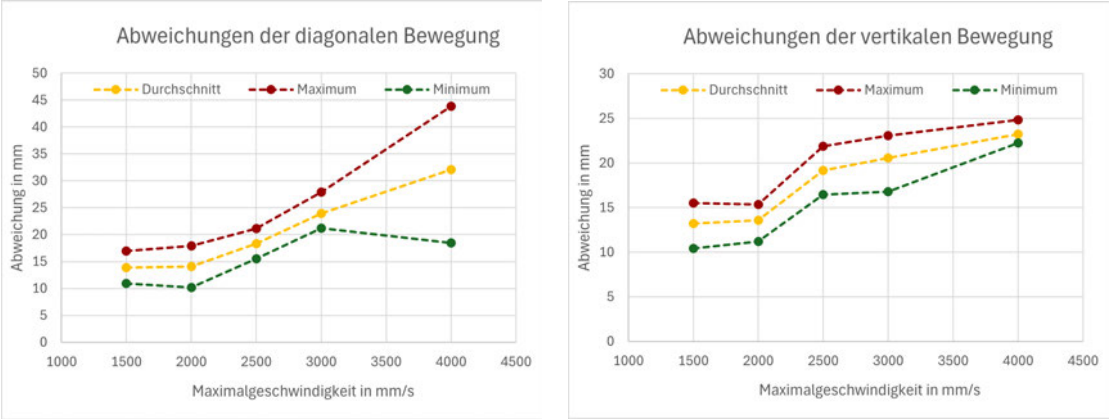


Abbildung 5.4: Ergebnisse der Linearitätstests für diagonale (links) und vertikale (rechts) Bewegungen

Das Ergebnis zeigt, dass bei einer Geschwindigkeit bis 2000 mm/s die Abweichung sowohl bei der vertikalen als auch bei der diagonalen Bewegung unter 20 mm liegt.

Die Geschwindigkeit von 2000 mm/s wird somit als maximale Geschwindigkeit für die Bewegung des Roboters festgelegt.

5.6 Untersuchung der Bewegungslinearität unter Last

Neben unbelasteten Bewegungen ist ebenso relevant, wie sich die durchgeführten Bewegungen unter Last verhalten, das heißt, wenn die Beine das Körpergewicht des Roboters tragen müssen.

5.6.1 Versuchsaufbau

Für diesen Versuch wird der Roboter das erste Mal aus eigener Kraft auf die Füße gestellt und eine vertikale und eine horizontale Bewegung durchgeführt. Das Ziel dieses Tests ist es, zu ermitteln, ob die Bewegung trotz der zusätzlichen Last innerhalb der Toleranzen ausgeführt wird.

Da die Position des Roboters abhängig von allen Beinen ist, wird die Bewegung nicht über die Position der Füße, sondern mittels der am Körper verbauten Intel RealSense Tracking Kamera gemessen. Laut dem Datenblatt der Kamera [Intel Corporation, 2019] kann eine Abweichung von bis zu 1 % der zurückgelegten Strecke auftreten. Um potenziellen Fehlern entgegenzuwirken, wird der Roboter nach dem Test in die Ausgangsposition zurückgefahren und die Abweichung der Position geprüft. Dabei werden keine Abweichungen festgestellt, die die Ergebnisse beeinflussen könnten. Genauere Angaben zu den Toleranzen der Kamera lassen sich der Dokumentation nicht entnehmen. Gemessen wird eine Messfrequenz der Kamera von 200 Hz. Im Stillstand rauschen die Messwerte mit einer Standardabweichung von 0.15 mm ohne nennenswerten Drift über 10 Minuten. Zudem können ebenfalls keine nennenswerten Abweichungen bei einer wiederholten Vorwärts- und Rückwärtsbewegung von 1 m festgestellt werden. Die Position der Kamera wird auf den Mittelpunkt des Roboters zurückgerechnet ("trunk" in Abbildung 6.1)

5.6.2 Ergebnisse

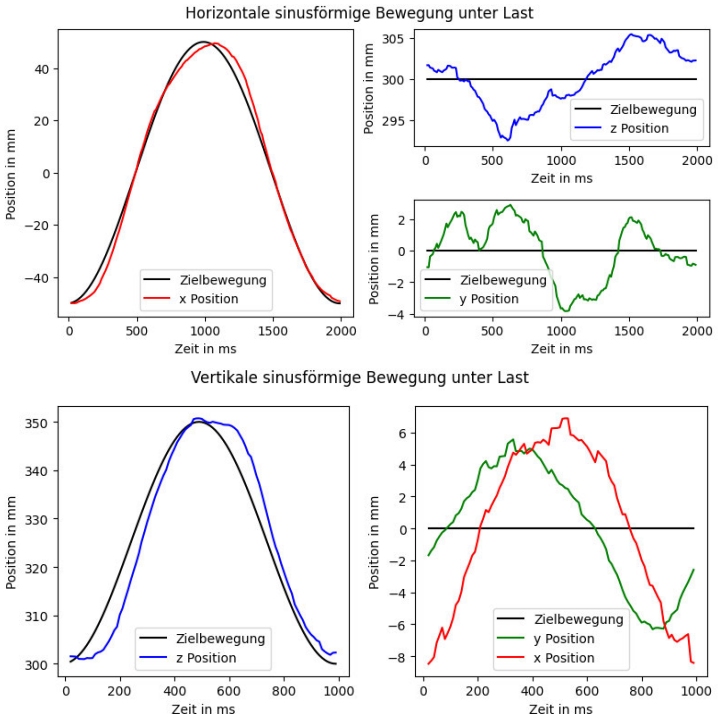


Abbildung 5.5: Darstellung einer vertikalen und horizontalen Sinusbewegung der Beine unter der Last des Roboterchassis

Die Messergebnisse gemäß Abbildung 5.5 zeigen, dass bei Bewegung unter Last jeweils Abweichung der stillstehenden Richtungen auftreten (rechts in der Abbildung). Bei der horizontalen Bewegung beträgt diese Abweichung maximal $7.9\,mm$ und durchschnittlich $3.7\,mm$. Bei der vertikalen Bewegung beträgt die maximale Abweichung $8.9\,mm$ und die durchschnittliche Abweichung $6.1\,mm$. Damit liegen die Abweichungen innerhalb der Toleranzen. Anzumerken ist jedoch, dass die Bewegungsgeschwindigkeit dieses Tests bedeutend langsamer ist, als die der Tests ohne Last (Abbildung 5.4).

5.7 Untersuchung der Bewegungslinearität bei ruckartigen Bewegungen

Für quadrupede Roboter ist es relevant, dass die Beine auch beim ruckartigen Anheben oder Absenken des Roboters innerhalb der Toleranzen bleiben. Entsprechend wird, zusätzlich zu den Sinusbewegungen, die Abweichung beim ruckartigen Abfangen und Anheben des Roboters getestet.

5.7.1 Versuchsaufbau

Für diesen Versuch wird als Zielposition ein Rechtecksignal für die z-Bewegung verwendet. Da es sich um eine Bewegung des ganzen Roboters handelt, wird die getestete Bewegungsgeschwindigkeit aus Sicherheitsgründen auf $500\,mm/s$ begrenzt. Die Aufnahme der Positionen erfolgt wie in Abschnitt 5.6 beschrieben.

5.7.2 Ergebnisse

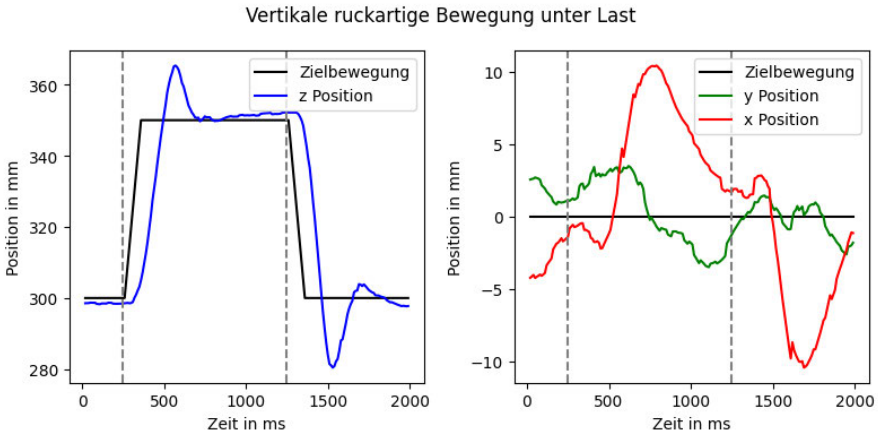


Abbildung 5.6: Darstellung einer ruckartigen vertikalen Bewegung des Roboters unter Last

Die Ergebnisse in Abbildung 5.6 zeigen, das die Bewegung des Roboters in x- und y-Richtung innerhalb der Toleranzen bleibt. Die maximale Abweichung liegt bei 10.4 mm und die durchschnittliche Abweichung bei 5.0 mm. Es tritt jedoch wie bei den belasteten Motorbewegungen aus Abbildung 3.13 ein Überschwingen in z sowohl beim Anheben als auch beim Absenken auf. Dieses Überschwingen ist zwar noch innerhalb der Toleranzen, jedoch sollte es bei zukünftigen Weiterentwicklungen genauer untersucht werden.

6 Simulation des Gesamtsystems

6.1 Beschreibung des Roboters

Die Beschreibung für die Simulation, des in dieser Arbeit verwendeten Roboters basiert auf dem Git Repository von Unitree Robotics [2025], welches unter der BSD 3-Clause License verwendbar ist. Die grundlegende Geometrie des Roboters Unitree A1 entspricht der für diese Arbeit verwendeten Plattform. Maße, 3D-Modelle, Kollisionen und Achsen sind auf die in dieser Arbeit verwendeten Plattform angepasst. Die Ansteuerung der Motoren erfolgt über das in Abschnitt 4.1 beschriebene Gazebo-Plugin.

6.1.1 Geometrie

Die Definition der Geometrie des Roboters beinhaltet die Beschreibung der Links und Joints. Die Beschreibung der Links beinhaltet die Position, an der der Link endet, sowie die Ausrichtung des neuen Koordinatensystems, das am Ende des Links entsteht. Die Joint-Beschreibung beinhaltet die Art des Joints, in diesem Fall ausschließlich Revolute, sowie die dafür benötigten Eigenschaften wie Drehrichtung und die Limits der Drehung. Die Geometrie des somit beschriebenen Roboters ist in Abbildung 6.1 dargestellt. Aus dieser Beschreibung kann ein TF-Tree generiert werden (Abbildung 6.2).

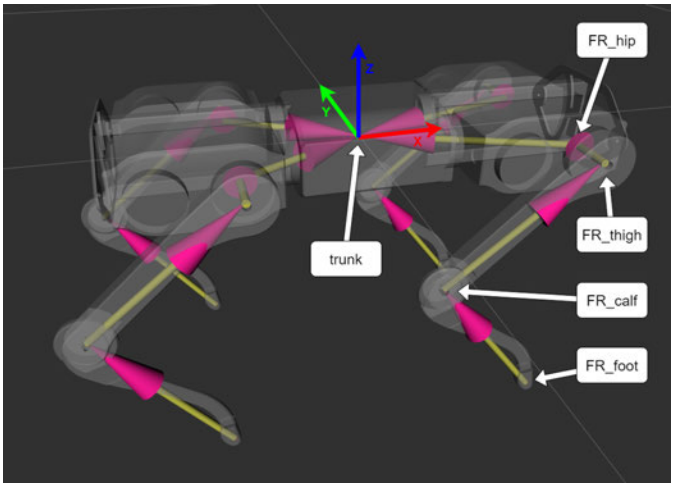


Abbildung 6.1: Darstellung der Links des Roboters mit dem Koordinatenursprung und der Orientierung der Achsen

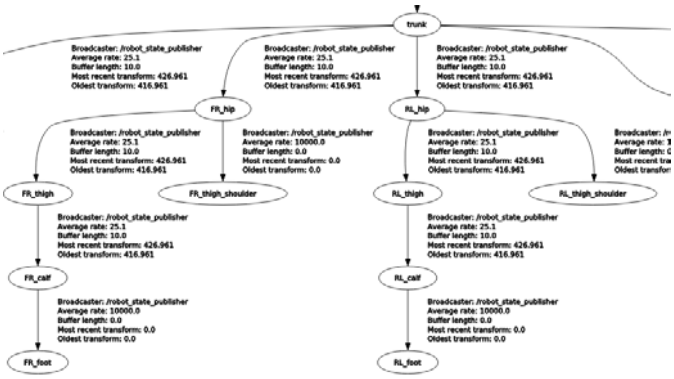


Abbildung 6.2: Ausschnitt aus dem TF-Tree des Roboters

6.1.2 3D-Modelle

Für die Darstellung des Roboters in Gazebo werden 3D-Modelle benötigt. Diese sind eine vereinfachte Version der 3D-CAD Modelle des Roboters. Die Modelle werden in Gazebo nur für die Visualisierung verwendet und haben keinen Einfluss auf die Simulation. Um Rechenleistung zu sparen, sind die Modelle nicht so hoch aufgelöst wie die 3D-CAD Modelle.

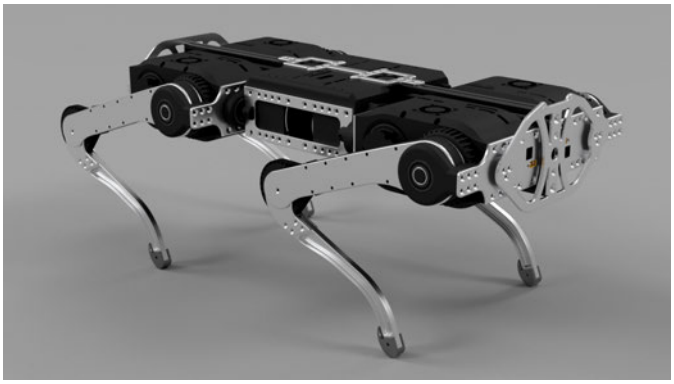


Abbildung 6.3: 3D-Modell des Roboters ohne Anbauteile in Autodesk Fusion 360

Für die Kollisionsberechnung werden zusätzlich zu den Visualisierungsmodellen Kollisionsmodelle benötigt. Diese sind so einfach wie möglich gehalten, um die Berechnung zu beschleunigen. So sind die gebogenen Beine des Roboters durch eine Box angenähert, da die genaue Form für die Simulation im Anwendungsfall dieser Arbeit nicht relevant ist.

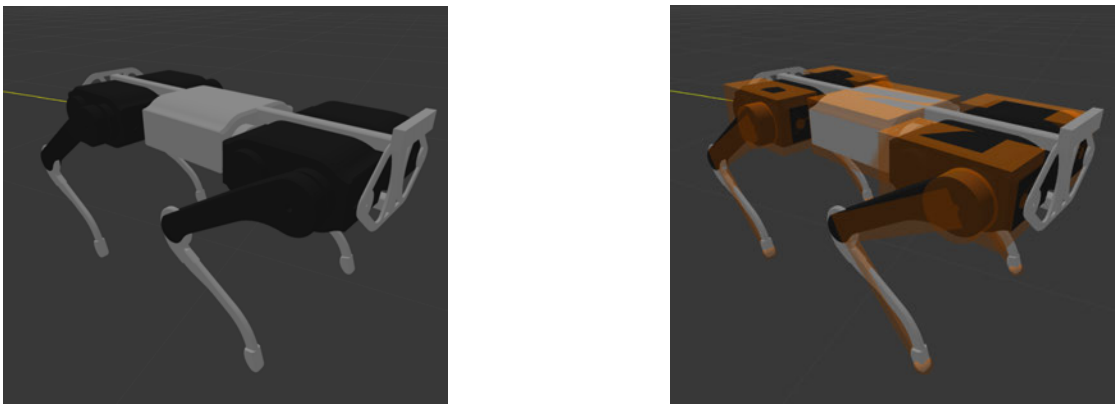


Abbildung 6.4: Visualisierungs- und Kollisionsmodell des Roboters in Gazebo

6.1.3 Trägheit und Gewicht

Damit Gazebo das Verhalten des Roboters korrekt simulieren kann, wird für jeden Körper der Trägheitstensor [Siciliano und Khatib, 2016, p.36] benötigt. Dieser ist eine 3x3-Matrix, die die Trägheit bei Änderungen des Drehimpulses beschreibt. Die Berechnung der Trägheitstensoren wird in diesem Fall durch Autodesk Fusion 360 durchgeführt. Zudem wird das Gewicht der Komponenten benötigt. Die Werte werden in der Xacro-Datei des Roboters hinterlegt.

6.2 Untersuchung der Abweichung von Beinbewegungen in der Simulation

Um die Abweichung zwischen den realen Beinen und der Simulation zu messen, werden wie bei der Motorsimulation aus Kapitel 4.3 Toleranzfelder definiert, die die maximale Abweichung zwischen den realen Beinen und der Simulation darstellen. Hierfür wird die zeitliche Toleranz von $\pm 20\,ms$ der Motorsimulation aus Kapitel 4.2 beibehalten. Die Toleranz wird auf Basis der Abweichung der Motorsimulation und die dadurch resultierende maximale Abweichung im kartesischen Raum abgeschätzt.

Die Benennung der verwendeten Variablen ist in Abbildung 5.1 zu sehen. Der Konfigurationsraum der Beine in kartesischen Koordinaten beschränkt L_b auf $L_{bMAX} = 400\,mm$. Durch eine Abweichung von θ_2 um die in Abschnitt 1.2 festgelegte Toleranz von $\Delta\theta_2 = 2,5^\circ$ lässt sich durch den Kosinussatz die maximale Abweichung am Fuß berechnen.

$$\sqrt{L_{bMAX}^2 + L_{bMAX}^2 - 2 \cdot L_{bMAX} \cdot L_{bMAX} \cdot \cos(\Delta\theta_2)} \approx 16.614\,mm \tag{6.1}$$

Auf Basis dieser Abschätzung wird die Toleranz im kartesischen Raum auf $15\,mm$ in x, y und z abgerundet. Wie auch schon in Abschnitt 4.3 bezieht sich diese Toleranz nicht auf die Abweichung der Simulation zur Zielvorgabe, sondern auf die Abweichung der Simulation zur Realität.

6.2.1 Versuchsaufbau

Für den Versuch werden dieselben Bewegungen wie in Abschnitt 5.5 verwendet, die am Roboter mit der ermittelten Maximalgeschwindigkeit von $2\,m/s$ simuliert werden.

6.2.2 Ergebnisse

Abbildung 6.5: Darstellung unterschiedlicher Bewegungen in der Simulation und der Realität zur Veranschaulichung der Abweichung der Simulation

Die Ergebnisse des Tests sind in Abbildung 6.5 dargestellt. Die Simulation befindet sich noch knapp innerhalb der gegebenen Toleranzen. Das gezeigte Verhalten ähnelt in allen Achsen dem des realen Roboters.

6.3 Untersuchung der Abweichung von langsamen Körperbewegungen in der Simulation

Um die Simulationsgenauigkeit der Beine unter der Last des Körpers und die Simulation des gesamten Körpers allgemein zu testen, werden wie bei den Versuchen der Beine, Testbewegungen durchgeführt. Als Toleranzen für die Bewegungen werden wie bei den Beinen (Abschnitt 6.2) $\pm 15\,mm$ in x, y und z und $\pm 20\,ms$ in der Zeit angenommen. In diesem Versuch soll die Simulation zeigen, dass sie einfache Bewegungen des Körpers unter Last abbilden kann.

6.3.1 Versuchsaufbau

Die Messung der Position erfolgt in der Mitte des Körpers ("trunk" in Abbildung 6.1). Bei dem realen Roboter wird die Position des Körpers erneut durch die verbaute Tracking-Kamera aufgenommen. Diese ist in Abschnitt 3.3.2 beschrieben. Die Messunsicherheiten der Kamera sind dieselben wie in Abschnitt 5.6.

In der Simulation wird die Bewegung ermittelt, indem die Position des Körpers aus der `model_states` Message ausgelesen wird, welche von Gazebo bereitgestellt wird. Die auf diese Weise ausgelesenen Positionsdaten haben keine Messabweichung, da sie den exakten Wert aus der Simulation darstellen.

Als Zielwert für die Bewegungen wird ein Sinus in z- und x-Richtung genutzt. Die Versuchsbewegungen gleichen den belasteten Tests der Beine aus Abschnitt 5.6.

6.3.2 Ergebnisse

Die für die Beine festgelegten Toleranzen werden, wie in Abbildung 6.6 zu sehen, eingehalten. Der reale Roboter oszilliert insbesondere bei der vertikalen Bewegung in z-Richtung bedeutend mehr als in der Simulation. Da sich die Oszillation jedoch innerhalb der Toleranzen befindet, wird sie als akzeptabel angesehen.

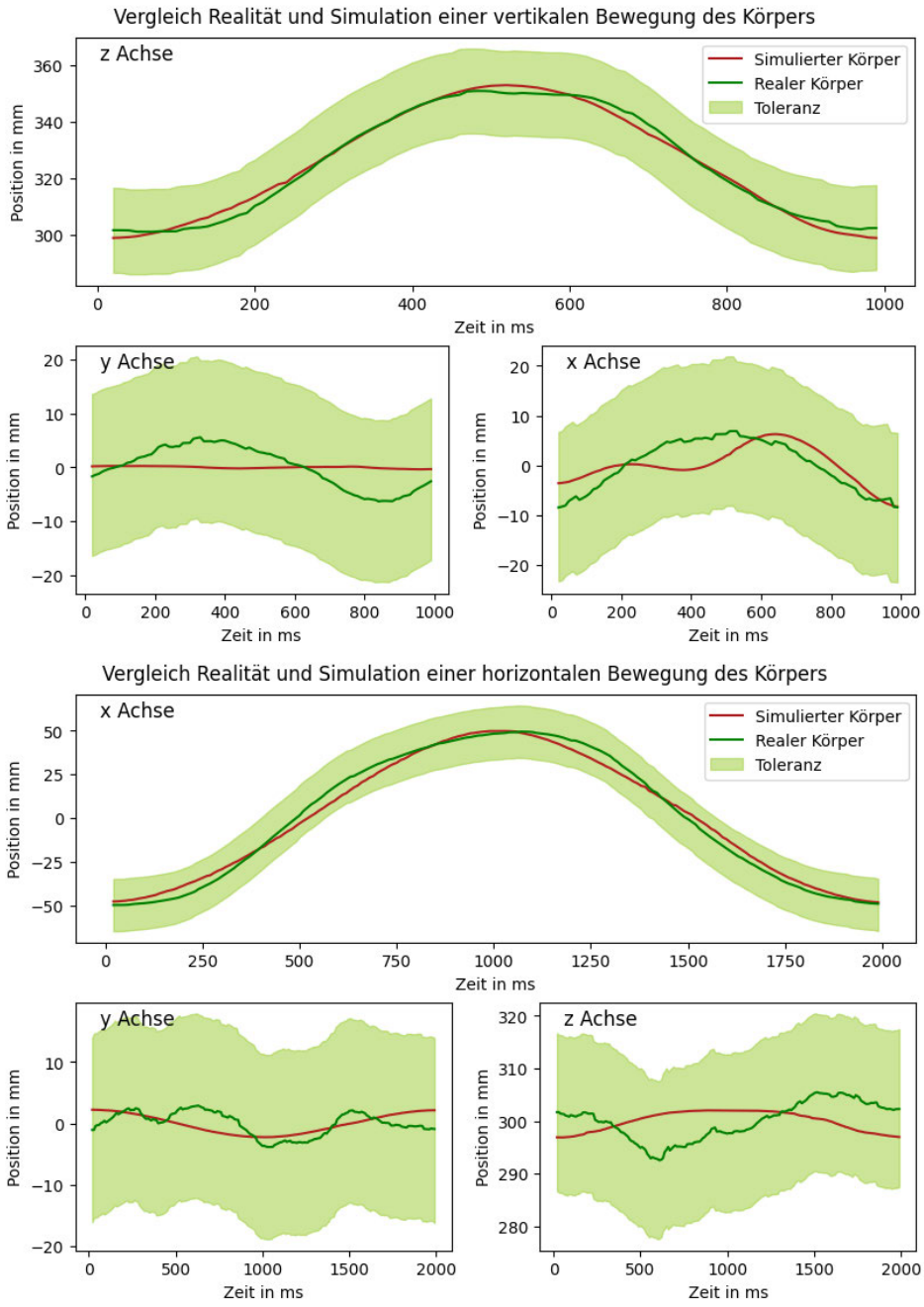


Abbildung 6.6: Darstellung zweier Sinus-Bewegungen des Roboters in der Simulation und der Realität zur Veranschaulichung der Abweichung der Simulation

6.4 Untersuchung der Abweichung von ruckartigen Körperbewegungen in der Simulation

Um Sicherzustellen, dass die Beine und die Simulation sich auch bei ruckartigen Bewegungen ähnlich verhalten, wird ebenfalls der Versuch aus Abschnitt 5.7 in der Simulation wiederholt.

6.4.1 Versuchsaufbau

Der Versuchsaufbau ist identisch zu dem in Abschnitt 6.3. Hierbei werden die Beine mit einer Geschwindigkeit von $500\,mm/s$ über eine Distanz von $50\,mm$ eingezogen und ausgefahren. Daraufhin wird der Körper bis zum Stillstand abgebremst. Das Ziel ist es festzustellen, ob das resultierende Überschwingen des Körpers ebenfalls in der Simulation abgebildet wird.

6.4.2 Ergebnisse

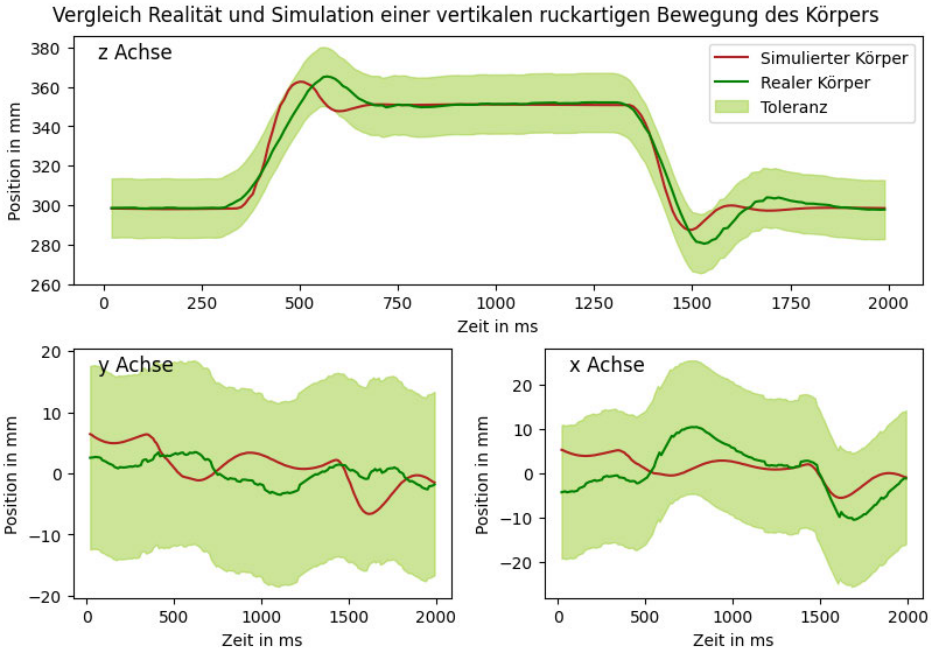


Abbildung 6.7: Darstellung einer ruckartigen Bewegung des Roboters in der Simulation und der Realität zur Veranschaulichung der Abweichung der Simulation

Wie in Abbildung 6.7 zu sehen, existiert das Überschwingen auch in der Simulation, auch wenn die Reaktion auf das Rechtecksignal in der Simulation schneller ist. Die getesteten Bewegungen liegen innerhalb der gesetzten Toleranzen.

6.5 Festgestellte Probleme in der Gazebo Simulation

Bei der Simulation des Roboters in Gazebo sind zwei Probleme aufgefallen, die an dieser Stelle kurz beschrieben werden sollen. Sie treten auf, wenn es zu viele Kontaktpunkte des Roboters mit dem Boden gibt oder wenn die Beine komplett gestreckt sind. In beiden Fällen fängt der Roboter an, in eine zufällige Richtung zu driften. Dieses Verhalten ist in Abbildung 6.8 anhand der gelben Linie dargestellt, die den zurückgelegten Pfad anzeigt.

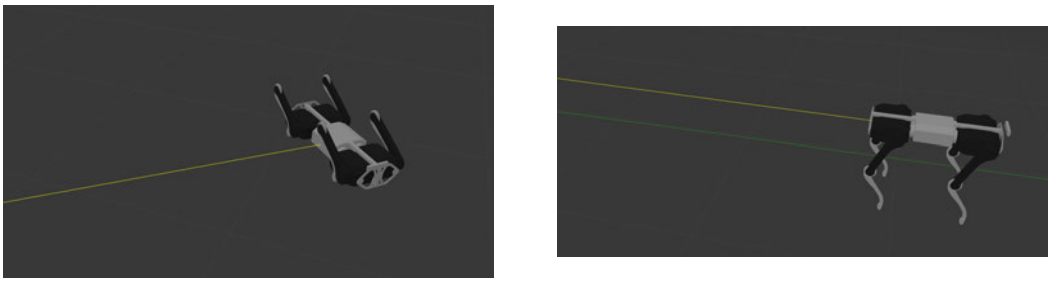


Abbildung 6.8: Visualisierung des Drifts bei problematischen Positionen in Gazebo anhand der gelben Pfadlinie

Liegend wurde der Drift mit $44\,mm/s$ gemessen. Stehend liegt der Drift bei $152\,mm/s$. Da diese beiden Positionen jedoch bei Versuchen zum Laufen des Roboters nicht auftreten, ist dieses Problem im Rahmen dieser Arbeit nicht relevant. Der Drift außerhalb dieser beiden Fälle liegt bei $0.00092\,mm/s$. Dieser Wert ist so gering, dass er vernachlässigt werden kann.

Durch die durchgeführten Tests wird ab jetzt davon ausgegangen, dass die Simulation in Gazebo die Bewegungen des Roboters innerhalb der gesetzten Toleranzen erfolgreich abbilden kann.

7 Robotersteuerung

Zum Testen von unterschiedlichen Gangarten wird eine Robotersteuerung entworfen, die die Ansteuerung der Beine übernimmt. Hierbei liegt das Hauptaugenmerk auf der Reduktion von Komplexität für die Gangarten. Auf diese Weise ist die Softwareentwicklung der Gangarten einfacher, auch wenn der Aufwand für den Rest der Steuerung steigt.

7.1 Klassen für Gangarten (Gait)

Um den Klassen für die Gangarten möglichst viel Komplexität abzunehmen, werden nur zwei Anforderungen an die Klassen gestellt, die diese erfüllen müssen.

- Gangarten müssen eine nach Abschnitt 2.7 im Stillstand statisch stabile Ausgangsposition haben, aus der sie die Bewegungen beginnen können.
- Gangarten müssen bei Aufforderung in eine im Stillstand statisch stabile Position zurückkehren.

Diese Anforderungen werden über ein Interface dargestellt, das die Gangarten implementieren müssen. Zusätzlich gibt es noch ein Interface für die Ansteuerung durch den Nutzer, dessen Standardfunktionen überschrieben werden können, wenn die Ansteuerung genutzt werden soll.

7.2 Klassen für die Repositionierung (Reposition)

Um die Gangarten zu wechseln, muss eine Möglichkeit geschaffen werden, dass der Roboter von einer im Stillstand statisch stabilen Position zu einer anderen wechselt. Diese Aufgabe übernimmt die Repositionierung.

Das Ziel der Repositionierung ist es, dass der Roboter von einer statisch stabilen Konfiguration der vier Beine in eine andere statisch stabile Konfiguration wechseln kann.

Die folgende Notation basiert erneut auf der im "Handbook of Robotics" [Siciliano und Khatib, 2008, p. 110] verwendeten Notation. Im Gegensatz zu Abschnitt 5.4, werden hier als Konfigurationsraum alle Achsen des Roboters betrachtet und nicht nur die eines Beines.

q ist eine mögliche Konfiguration des Roboters, die alle vier Beine beinhaltet und $\mathcal{C}$ ist die Menge aller möglichen Konfigurationen des Roboters. $\mathcal{C}_{\text{free}}$ ist die Menge der Roboterkonfigurationen bei denen die Beine eine gültige Position haben. q_1 und q_g sind die Start- und Zielkonfigurationen der Repositionierung. Diese Definitionen sind demnach analog zu Abschnitt 5.4.

C_{stable} sei definiert durch die Menge aller erlaubten Konfigurationen, die statisch stabil sind. Demnach ist $C_{\text{stable}} = \{q \in \mathcal{C}_{\text{free}} | q \text{ ist statisch stabil}\}$.

Analog zu den Beinbewegungen sucht die Repositionierung einen Pfad $\tau : [0, 1] \rightarrow \mathcal{C}_{\text{free}}$, sodass $\tau(0) = q_1$ und $\tau(1) = q_g$ gilt. Zusätzlich muss $\forall t \in [0, 1] : \tau(t) \in C_{\text{stable}}$ gelten. Auf diese Weise ist sichergestellt, dass der Roboter während der Bewegung statisch stabil bleibt.

Die Umsetzung der vollständigen Logik der Repositionierung ist sehr umfangreich und wird in dieser Arbeit aus Zeitgründen nicht durchgeführt. Stattdessen wird eine einfache Version implementiert, die den Roboter in eine neue Position bringt, ohne dabei die Stabilität zu gewährleisten. Diese ist nur für Testzwecke gedacht und sollte aufgrund von Risiken für die Hardware nicht auf einem echten, stehenden Roboter verwendet werden.

7.3 Klassen für Nutzereingaben (UserInput)

Um Eingaben vom Nutzer zu ermöglichen, gibt es zwei Optionen.

- Die Eingabe erfolgt über das an die Gangarten bereitgestellte Interface zur Steuerung durch den Nutzer.
- Die Gangarten definieren eine eigene Ansteuerung beispielsweise über ROS Topics.

Um das bereitgestellte Interface zu nutzen, können Adapterklassen implementiert werden, die es erlauben, unterschiedliche Eingabegeräte zu verwenden. Für die Kommunikation mit ROS Joy und 3DConnexion SpaceMouse ist ein solcher Adapter implementiert. Der große Vorteil beim Nutzen des Interfaces ist, dass die Gangarten nicht mehr abhängig von der spezifischen Art der Steuerung sind. Zudem erlaubt dies dem GaitSelector, die Nutzereingaben nur an die verwendete Gangart weiterzuleiten.

7.4 Klasse für die Beinansteuerung (LegControl)

Die Beinansteuerung ist in unterschiedliche Komponenten aufgeteilt, die bestimmte Aufgaben übernehmen.

- Ansteuerung der Motoren über die in Abbildung 3.9 dargestellten Messages
- Berechnung von Achswinkeln und Kalibrierung (Abschnitt 5.1)
- Berechnung der Forward Kinematik und Inversen Kinematik (Abschnitt 5.3)
- Begrenzung des Konfigurationsraumes (Abschnitt 5.4)

Die einzelnen Komponenten sind austauschbar und werden nach außen von einer Klasse repräsentiert.

7.5 Konzept für das Wechseln von Gangarten (GaitSelector)

Um das Wechseln von Gangarten zu ermöglichen, wird eine Stellvertreter-Gangart mit dem Namen Gangsteuerung geschaffen. Die Gangsteuerung verhält sich wie eine normale Gangart, beinhaltet im Hintergrund jedoch beliebige weitere Gangarten, zwischen denen sie wechseln kann. Hierfür kontrolliert sie den Informationsfluss zwischen der Nutzersteuerung und den Gangarten und schaltet die Gangarten je nach Anforderung ein oder aus. Auf diese Weise erlaubt sie immer nur einer Gangart zur Zeit den Zugriff auf die Beinansteuerung. Die Aufgabe der Stellvertreter-Gangart ist es, die Übergänge von und zu den Gangarten zu steuern. Hierfür wird die Repositionierung genutzt, um den Roboter in die von der neuen Gangart benötigte Position zu bringen. Zur Steuerung des

Wechsels zwischen den Gangarten wird ein einfacher Zustandsautomat verwendet. Dieser ist in Kombination mit einer kurzen Beschreibung der Benennungen in Abbildung 7.1 dargestellt.

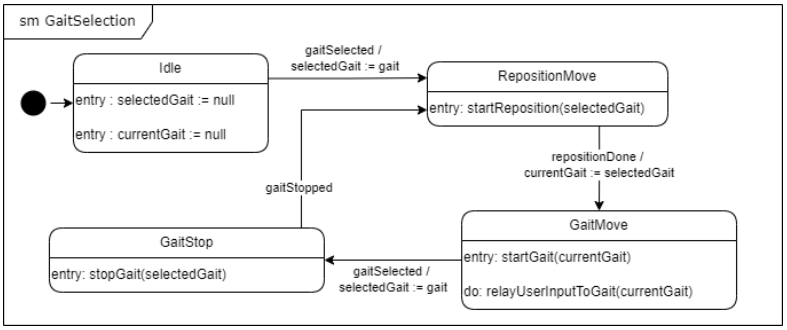


Abbildung 7.1: Zustandsautomat für das Wechseln von Gangarten durch die Gangsteuerung

- currentGait ist eine Variable, die die aktuelle Gangart speichert.
- selectedGait ist eine Variable, die die ausgewählte Gangart speichert.
- gaitSelected ist ein Event, das ausgelöst wird, wenn eine Gangart über die Nutzersteuerung ausgewählt wird. Der Name der Gangart ist in ”gait” gespeichert.
- startReposition(gait) ist eine Action, die die Repositionierung zur Startposition von ”gait” startet.
- repositionDone ist ein Event, das ausgelöst wird, wenn die Repositionierung abgeschlossen ist.
- startGait(gait) ist eine Action, die die Gangart ”gait” zum Starten auffordert.
- relayUserInputToGait(gait) ist eine Action, die die Nutzereingaben an die Gangart ”gait” weiterleitet.
- stopGait(gait) ist eine Action, die die Gangart ”gait” auffordert, in einen stabilen Zustand zu wechseln und sich zu beenden.
- gaitStopped ist ein Event, das ausgelöst wird, wenn die Gangart den stabilen Zustand erreicht hat und beendet ist.

7.6 Übersicht über die Klassen

Um das Austauschen der aktuellen Gangart zu ermöglichen, bestimmt die Gangsteuerung, welche Gangart die Beinansteuerung nutzen darf. Um dies zu verdeutlichen, zeigt Abbildung 7.2 ein vereinfachtes Klassendiagramm für die Gangsteuerung. Hierbei ist zu sehen, dass die Gangsteuerung dieselbe Schnittstelle implementiert wie die Gangarten. Der Unterschied besteht darin, dass die Gangsteuerung die Instanzen von Gangarten und Repositionierung aggregiert und als deren Stellvertreter fungiert.

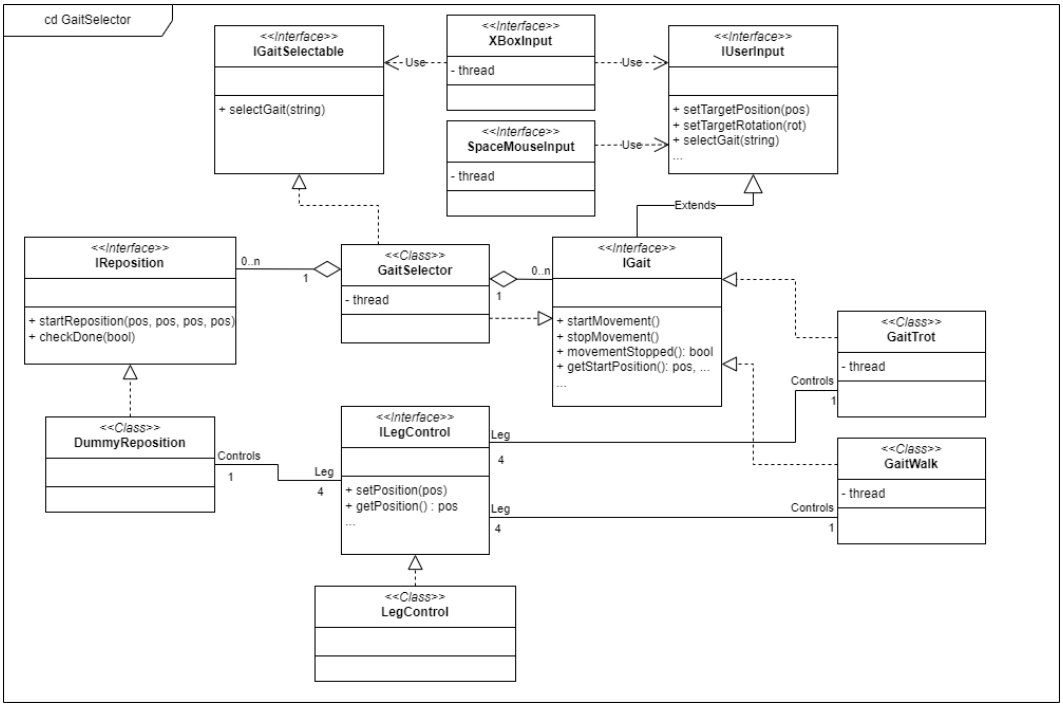


Abbildung 7.2: Vereinfachtes Klassendiagramm für das Wechseln von Gangarten durch die Gangsteuerung

Die Stellvertreter-Beziehung ist in Abbildung 7.3 noch einmal dargestellt, indem eine Konstruktion mit der Möglichkeit mehrerer Gangarten und eine mit einer festen Gangart gezeigt wird.

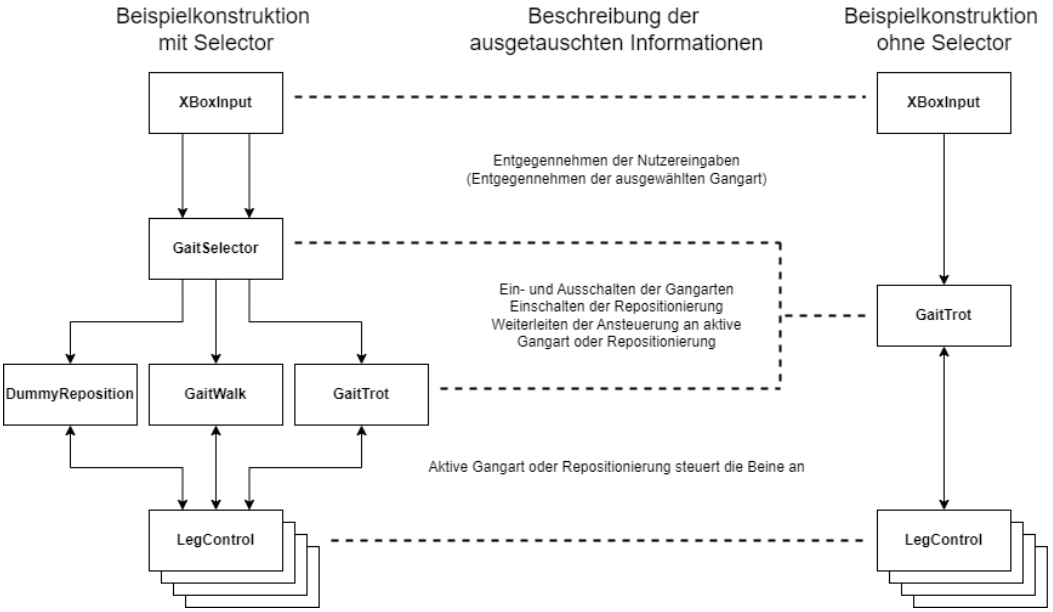


Abbildung 7.3: Informelle Darstellung einer Beispielkonstruktion der in Abbildung 7.2 gezeigten Klassen, sowie die Richtung der ausgetauschten Informationen

8 Gangarten und erste Schritte

Im folgenden Kapitel werden Gangarten vorgestellt und für die Softwarearchitektur des Roboters implementiert. Diese Gangarten werden daraufhin in der Simulation anhand von drei beispielhaften Metriken bewertet. Diese Metriken sind die Maximalgeschwindigkeit, die Toleranz gegen äußere Kräfte und die Bewegungsruhe beim Laufen.

Da wie in Abschnitt 3.5.1 beschrieben, einer der Umrichter durch einen Versuch nicht mehr nutzbar ist, ist es im Rahmen dieser Arbeit nicht mehr möglich, die simulierten Resultate anhand des realen Roboters zu verifizieren.

8.1 Gangarten

Um Gangarten miteinander vergleichen zu können, werden die Gangarten Schritt und Trott implementiert und in je zwei Variationen getestet.

Alle Implementierungen für diese Arbeit sind Open-Loop. Sie basieren somit nicht auf Sensoren, durch die der Roboter in Balance gehalten wird.

8.1.1 Gangart Schritt

Die für diese Arbeit geschriebene Version des Schritts hat wie in Abschnitt 2.1 beschrieben, maximal ein Bein in der Luft. Damit die Gangart statisch stabil ist, wird beim Anheben eines Beines der Schwerpunkt des Roboters in Richtung der anderen Beine verlagert. Somit bleibt der Schwerpunkt über dem Stützpolygon. Dies wird durch eine Kreisbewegung des Chassis erreicht. Neben der Laufgeschwindigkeit, die durch Nutzereingaben gesteuert wird, besitzt diese Gangart außerdem eine Menge an Parametern, die das Verhalten beeinflussen. Diese Parameter sind in Tabelle 8.1 beschrieben.

Parameter	Beschreibung
`height`	Die Höhe des Körpers
`step_height`	Die Höhe des angehobenen Beines
`step_speed`	Die maximale Geschwindigkeit
`step_overlap`	Der Anteil der Schrittlänge, bei der alle Beine auf dem Boden sind
`lean_distance`	Der Radius der Kreisbewegung des Schwerpunktes
`lean_phase`	Die Phasenverschiebung der Schwerpunktverlagerung zum angehobenen Bein

Tabelle 8.1: Parameter für die Gangart Schritt

Die Gangart Schritt ist bei niedrigen Geschwindigkeiten statisch stabil. Da bei höheren Geschwindigkeiten die Schritte schneller werden, wird auch das Verlagern des Schwerpunktes schneller. Dies führt letztlich zum Umkippen des Roboters. Wird bei schnelleren Geschwindigkeiten die Verlagerung des Schwerpunktes entfernt, so lassen sich stabile Zyklen finden. Die Geschwindigkeit, ab der die Verlagerung des Schwerpunktes nicht mehr sinnvoll ist, wird in Kapitel 8.2 genauer beschrieben.

Aus diesem Grund wird Schritt in zwei Versionen aufgeteilt, die sich nur durch ihre Parameter unterscheiden. Der langsame Schritt ist statisch stabil, da sich der Schwerpunkt immer über dem Stützpolygon befindet. Das schnelle Schritt basiert auf stabilen Zyklen (Abschnitt 2.7).

8.1.2 Gangart Trott

Die beiden für diese Arbeit implementierte Trott Gangarten haben wie in Abschnitt 2.1 beschrieben zwei diagonale Beine in der Luft jedoch keine Schwebephase beim Wechseln der Beinpaare. Die erste für diese Arbeit geschriebene Version des Trott ist rein reaktiv. Dies bedeutet, dass sich bei einer durch den Nutzer vorgegebenen Zielbewegung, die beiden auf dem Boden befindlichen Beine in die entgegengesetzte Richtung bewegen, um einen Vorschub zu erzeugen. In regelmäßigen Abständen werden diese Beine angehoben, um sie neu zu positionieren. Bei der Neupositionierung der Beine werden sie in ihre Ausgangsstellung zurückbewegt.

Die Ausgangsstellung der Beine ist so gewählt, dass in dieser Position der Schwerpunkt des Roboters genau in der Mitte des Stützpolygons liegt. Zudem wird die Last gleichmäßig auf alle Beine verteilt. Je weiter sich die Füße im Rahmen des Bewegungsablaufs von dieser Position entfernen, desto ungleichmäßiger wird das Gewicht verteilt.

Die zweite Version der Gangart Trott ist prädiktiv. Bei einer Eingabe reagieren die Beine auf die Zielrichtung, bei der Neupositionierung des Beines wird jedoch das Bein über die Ausgangsposition hinaus bewegt. Hierbei wird die Abschätzung in die Zukunft gemacht, dass sich die Bewegungsrichtung des Roboters nicht ändert. Dies führt dazu, das die Beine nach Beendigung der Vorschubphase noch nicht so weit von der Ausgangsposition entfernt sind. Zudem besteht der Vorteil, dass die maximale Distanz, die ein Bein vor der Neupositionierung zurücklegen kann, im optimalen Fall doppelt so groß ist wie bei der reaktiven Version. Dies erhöht die Maximalgeschwindigkeit bei gleicher Schrittfrequenz. Der Unterschied der beiden Versionen ist in Abbildung 8.1 dargestellt. Bei starken Richtungsänderungen kann es jedoch dazu kommen, dass das Bein sich weiter von der Ausgangsposition entfernt, als beim reaktiven Trott. In diesem Fall wird die Stabilität der Gangart stark beeinträchtigt.

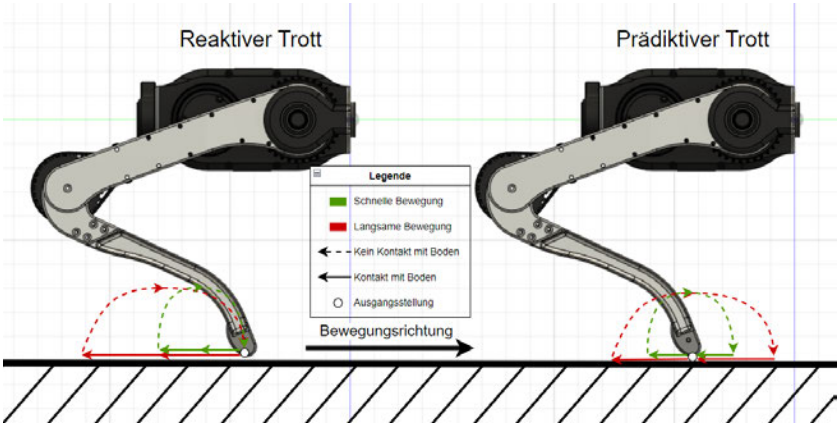


Abbildung 8.1: Gegenüberstellung der Vorwärtsbewegung eines Beines in den beiden Trott-Versionen

Wie auch Schritt, besitzen die beiden Trott Implementierungen mehrere Parameter, mit denen das Verhalten so angepasst werden kann, dass ein stabiler Zyklus entsteht.

Parameter	Beschreibung
`height`	Die Höhe des Körpers
`step_height`	Die Höhe des angehobenen Beines
`step_speed`	Die maximale Geschwindigkeit
`step_overlap`	Der Anteil der Schrittlänge, bei der alle Beine auf dem Boden sind

Tabelle 8.2: Parameter der Gangart Trott

8.2 Untersuchung der Geschwindigkeit und Stabilität

Eine der wichtigsten Eigenschaften einer Gangart sind die Geschwindigkeiten, bei denen sie funktioniert. Zudem muss eine Gangart resistent gegen äußere Einwirkung sein. Im nachfolgenden Versuch werden die vier in Abschnitt 8.1 vorgestellten Versionen von Gangarten in der Simulation auf Geschwindigkeit und Stabilität getestet.

8.2.1 Testmethodik

Für den Test wird der Roboter bei unterschiedlichen Geschwindigkeiten unterschiedlich starken Stößen ausgesetzt. Gazebo bietet hier die Möglichkeit, Kräfte auf den Roboter auszuüben. Während der Roboter läuft, werden in zufälligen Abständen bis zu $20\,kN$ über einen Zeitraum von $1\,ms$ seitlich auf den Roboter ausgeübt. Dies wird mindestens 20-mal für jede getestete Stoßkraft wiederholt, um ein aussagekräftiges Ergebnis zu bekommen. Die Vorgehensweise ist in Abbildung 8.2 dargestellt.

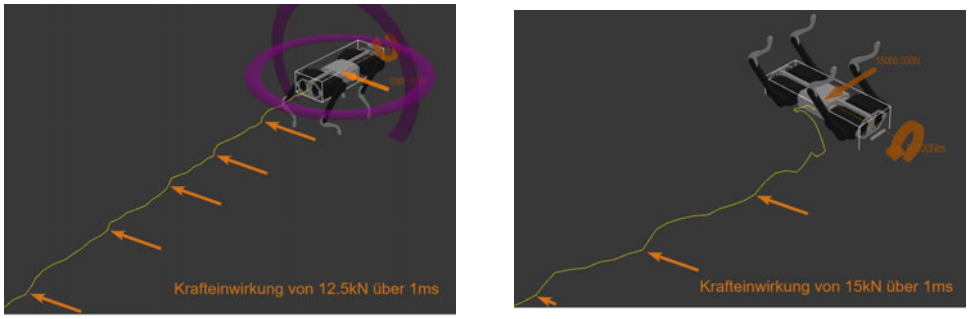


Abbildung 8.2: Der Roboter beim prädiktiven Trott unter der Einwirkung von $12.5\,kN$ (links) und $15\,kN$ (rechts)

Die Kraft der Stöße wird so lange erhöht, bis der Roboter zu kippen beginnt. Ist die Geschwindigkeit bei den Gangarten zu hoch, so beginnt der Roboter zu kippen, ohne dass eine Kraft auf den Roboter ausgeübt wird. Aufgrund dieser Tests werden die Parameter der Gangarten so angepasst, dass sie bei einer möglichst hohen Geschwindigkeit stabil bleiben.

8.2.2 Ergebnisse

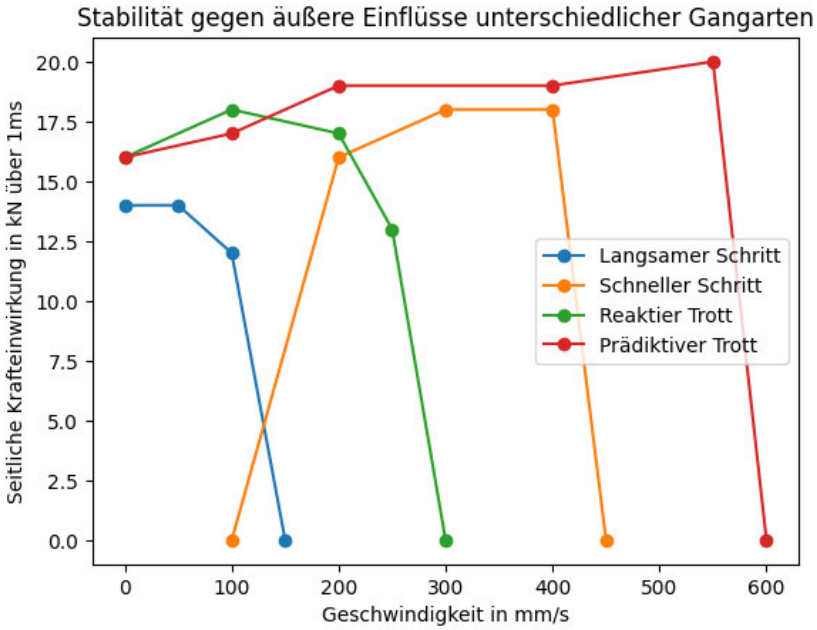


Abbildung 8.3: Krafteinwirkung auf den Roboter bei verschiedenen Geschwindigkeiten für die unterschiedlichen Gangarten

Die Ergebnisse der optimierten Gangarten sind in Abbildung 8.3 dargestellt. Aus diesen Ergebnissen lassen sich einige Eigenschaften entnehmen. Der langsame Schritt ist mit $100\,mm/s$ die langsamste der getesteten Gangarten, gefolgt vom reaktiven Trott mit $350\,mm/s$. Der schnelle Schritt ist mit $400\,mm/s$ die zweitschnellste Gangart und der prädiktive Trott ist mit $550\,mm/s$ die schnellste. Zudem fällt auf, dass der schnelle Schritt unter $200\,mm/s$ nicht mehr stabil ist.

Was die Stabilität gegen äußere Kräfte angeht, so existiert ein signifikanter Unterschied zwischen dem statisch stabilen langsamen Schritt und den Gangarten die auf stabilen

Zyklen basieren. Die, die auf stabilen Zyklen basieren, vertragen den Tests nach eine seitliche Krafteinwirkung von ungefähr $17.5\,kN$ über $1\,ms$. Der langsame Schritt wurde bereits über $14\,kN$ instabil.

8.3 Untersuchung der Bewegungsruhe

Für jede der nicht stabilen Gangarten lassen sich eine Vielzahl an Konfigurationen finden, die beim Laufen in stabile Zyklen resultieren. Die Formen dieser Zyklen sind je nach Gangart unterschiedlich, sollten sich aber nach einer vollständigen Schrittfolge wiederholen. Anhand der Abweichung lassen sich Aussagen über die Bewegungsruhe des Körpers treffen, je kleiner die Abweichung, desto weniger schwankt der Körper beim Laufen. Dies ist zwar nicht direkt für den Roboter relevant, kann jedoch die Messung einiger Sensoren wie potenziell später verbaute LiDAR-Sensoren beeinflussen.

8.3.1 Testmethodik

Für die Darstellung dieser Zyklen wird in der Simulation die Position des Roboters über drei Schrittzyklen der jeweiligen Gangart aufgenommen und die berechnete Zielbewegung des Roboters von der Position des Körpers abgezogen. Somit ergibt sich die Abweichung des Körpers zur Zielbewegung. Je geringer die Abweichung der Zyklen vom Ursprung, desto ruhiger bewegt sich der Roboter.

8.3.2 Ergebnisse

Wie in Abbildung 8.4 und Tabelle 8.3 zu sehen, ist der prädiktive Trott, die ruhigste der hier getesteten Gangarten. Langsames Schritt ist mit Abstand die unruhigste Gangart, da sie durch das absichtliche fortlaufende Verlagern des Schwerpunktes einen großen Versatz hat.

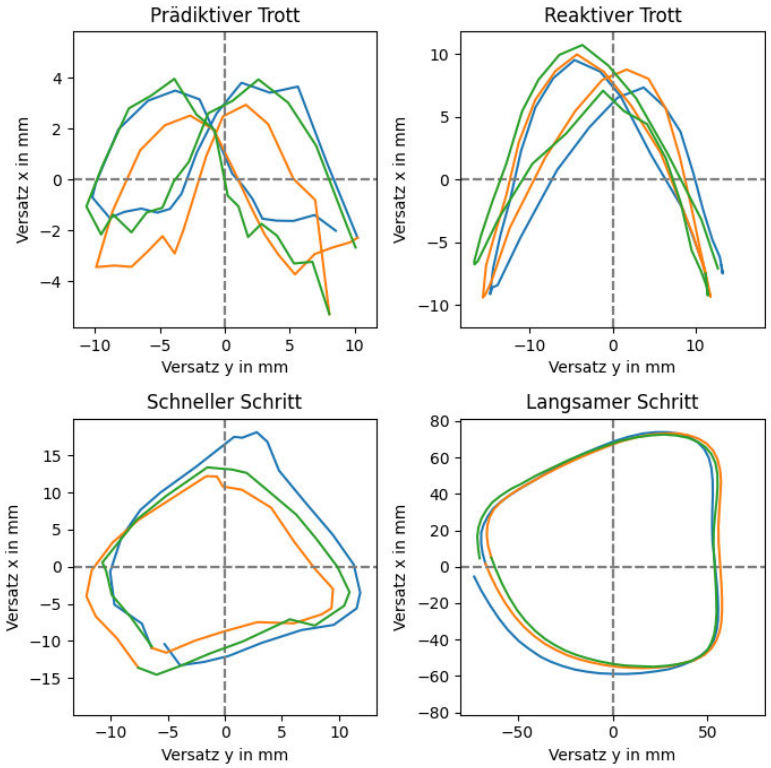


Abbildung 8.4: Darstellung der Zyklen verschiedener Gangarten als Versatz zur Zielbewegung (je Farbe wird ein gemessener Zyklus dargestellt)

Gangart	**Max. Versatz x**	**Max. Versatz y**	**Abs. Max. Versatz**
Prädiktiver Trott	5.3 mm	10.7 mm	10.7 mm
Reaktiver Trott	10.7 mm	16.7 mm	18.3 mm
Schneller Schritt	18.1 mm	12.1 mm	18.4 mm
Langsamer Schritt	73.9 mm	72.8 mm	83.8 mm

Tabelle 8.3: Maximaler Versatz zum Ursprung der verschiedenen Gangarten

9 Fazit

9.1 Zusammenfassung der Evaluationen

Da diese Arbeit gestaffelt aufgebaut ist, folgt der Übersicht halber eine Zusammenfassung der Ergebnisse der unterschiedlichen Versuche.

9.1.1 Motoren

Bei den Motoren wird unbelastet eine Einschwingzeit von $120\,ms$ ohne Überschwingen erreicht. Belastet mit einem Gewicht von $5\,kg$ wird wie erwartet, ein deutliches Überschwingen mit einer Einschwingzeit von $200\,ms$ gemessen (Abschnitt 3.6.2). Die Simulation hat ohne zusätzliche Last die Toleranz von $\pm 2°$ oder $\pm 20\,ms$ eingehalten (Abschnitt 4.3.2).

9.1.2 Beine

Die maximale Abweichung bei Bewegungen liegen unter der Maximalgeschwindigkeit von $2000\,mm/s$ innerhalb der Toleranzen von $20\,mm$ (Abschnitt 5.5.2). Bei gleichmäßigen Bewegungen unter der Last des Körpers wird eine durchschnittliche Abweichung von $3.7\,mm$ zum Zielpfad gemessen (Abschnitt 5.6.2). Bei ruckartigen Bewegungen wird wie auch bei den Motoren ein deutliches Überschwingen gemessen, das sich ebenfalls innerhalb der Toleranzen bewegt (Abschnitt 5.7.2). Die Abweichung des simulierten Körpers bei unterschiedlichen Bewegungen sowohl belastet, als auch unbelastet, liegt innerhalb der Toleranz von $15\,mm$ (Abschnitt 6.2.2, 6.3.2 und 6.4.2).

9.1.3 Bewegung

Die Analyse der implementierten Gangarten durch drei unterschiedliche Metriken hat gezeigt, dass die Gangarten anhand der Simulation verglichen werden können. Hierbei hat sich unter anderem herausgestellt, dass der prädiktive Trott die schnellste und die ruhigste Gangart ist. Zudem wird festgestellt, dass der statisch stabile langsame Schritt weniger seitliche Kräfte aushält, als die anderen Gangarten (Abschnitt 8.2.2 und 8.3.2).

9.2 Diskussion

9.2.1 Realer Roboter

Die Hardware des Roboters hat sich im Laufe der Arbeit in dem verwendeten Testumfeld bis auf den fehlerhaft parametrierten Umrichter (Abschnitt 3.5.1) als zuverlässig herausgestellt, könnte aber dennoch verbessert werden. Der verbaute Rechner ist leistungsfähig genug, um die Simulation und die Steuerung des Roboters zu bewältigen. Die verbauten Motoren und Umrichter sind ausreichend stark und genau für diesen Anwendungsfall. Die Übersetzungsverhältnisse der Motor Hubs könnten noch erhöht werden, sodass der Roboter mehr Kraft hat. Sollte die Plattform außerhalb des Testumfeldes genutzt werden, müssten einige Teile der Mechanik noch überarbeitet werden. Die Beine sind nicht stabil genug, um beim Umfallen des Roboters einen Schaden zu vermeiden. Zudem würde ein Überrollkäfig für den Schutz der Elektronik und der Batterien benötigt werden.

Die Kalibrierungen der Beine des Roboters sind zeitaufwändig. Das Sichern der Kalibrierungsinformationen auf dem ROS Parameterserver ermöglicht es jedoch zuverlässig, dass die Kalibrierung nur ein mal beim Starten des Roboters durchgeführt werden muss.

Problematisch ist, dass die Motoren unter Last überschwingen. Dieses Verhalten ist bereits bei den Motortests aufgetreten (Abbildung 3.13), wurde jedoch in Kauf genommen, um die schnelle Einschwingzeit der Motoren beizubehalten. In der Retrospektive wäre es besser gewesen die Einschwingzeit zu verlangsamen, um das Überschwingen zu verringern oder zu vermeiden. Dadurch könnte vermutlich auch das Überschwingen bei ruckartigen Bewegungen des gesamten Roboters verringert werden (Abschnitt 5.7.2).

Bei der gemessenen Maximalgeschwindigkeit der Beine von $2000\,mm/s$ wird die Toleranz von $20\,mm$ zur Zielbewegung eingehalten. Wie erwartet, ist die Abweichung bei

langsameren Bewegungen, auch unter Last bedeutend geringer. Die Abweichung des Roboterkörpers unter Last, von durchschnittlich 3.7 mm (Abschnitt 5.6.2), war erstaunlich genau. Vor dem Fortschritt der in dieser Arbeit erreicht wurde, wurde zur Verifikation der mechanischen Komponenten in den Beinen, eine ähnliche Bewegung durchgeführt. Diese Bewegung erzeugte Schwingungen des Roboterkörpers von mehreren Zentimetern. Demnach hat die Überarbeitung der Regler sowie der Steuerung der Beine die Genauigkeit Bewegungen des Roboterkörpers deutlich verbessert.

Aufgrund des Fehlers in der Parametrierung der Umrichter, konnten die implementierten Gangarten nicht auf der Hardware getestet werden (Abschnitt 3.5.1). Da die Simulation jedoch in anderen Tests immer recht konsistent die Hardware abgebildet hat, liegt nahe, dass die implementierten Gangarten auch auf der Hardware funktionieren. Eine Bestätigung dieser Annahme steht jedoch noch aus.

9.2.2 Simulierter Roboter

Die Simulation des Roboters war größtenteils konsistent mit dem realen Roboter. Auch wenn die Regler der Motoren zu von Beginn an etwas schneller reagiert haben, ließen sich Phänomene wie das Überschwingen unter Last, welches beim realen Roboter aufgetreten ist, auch in der Simulation wiederfinden (Abbildung 6.4.2). Zudem war die Übertragbarkeit zwischen den unterschiedlichen Tests der Simulation stets gegeben. Die Probleme, die bei unterschiedlichen Positionen des Roboters in Gazebo aufgetreten sind, waren wie erwartet für die in dieser Arbeit durchgeführten Tests nicht relevant und sind beim Testen auch nie negativ aufgefallen (Abschnitt 6.5).

9.3 Schluss

Lassen sich die vorgegebenen Anforderungen an die Genauigkeit des Roboters mit den verbauten Hardware und der in dieser Arbeit entwickelten Software erfüllen?
Bei der Genauigkeit der Bewegungen ließ sich durch die angepassten Regler, sowie die Steuerung der Beine, die geforderte Genauigkeit von 5 % der Schulterhöhe bei Geschwindigkeiten bis zu 2000 mm/s erreichen. Bei langsameren Bewegungen war die Abweichung trotz des Körpergewichtes bedeutend geringer.

Ist Gazebo geeignet um einen Quadrupeden Roboter innerhalb der gesetzten Toleranzen abzubilden?
Die Ergebnisse der Simulation in Gazebo von einem zum nächsten Test haben keine unerwarteten Abweichungen gezeigt und waren sehr konsistent. Die Darstellung von Bewegungen neben denen der angesteuerten Gelenke, ist wechselhaft. So werden bei Bewegungen in eine Richtung immer auch Abweichungen in andere Richtungen gemessen (Abbildung 6.6). Diese Abweichungen sind teilweise sehr genau nachgestellt, teilweise werden sie jedoch im geringen Rahmen nicht nachvollziehbar in andere Richtungen abgebildet. Alles in allem ist Gazebo jedoch soweit es in dieser Arbeit festgestellt werden konnte, geeignet um Gangarten eines Quadrupeden Roboters zu simulieren.

Lassen sich Gangarten Schritt und Trott auf Basis der entwickelten Plattform vergleichen und bewerten?
Die entwickelte Softwarearchitektur hat sich als gute Grundlage für die Analyse unterschiedlicher Gangarten erwiesen. Auch wenn die Repositionierung der Beine nur provisorisch implementiert wurde, erlaubt die Steuerung einen Zugriff auf viele relevante Daten die für die Bewertung von Gangarten wichtig sind. Zudem ist die Software leicht mit weiteren Gangarten erweiterbar.

9.4 Ausblick

Für die zukünftige Weiterführung der Arbeit gibt es noch einige offene Punkte.

Die Hardware des Roboters bedarf noch einer Revision. Die Beine müssen stabiler konstruiert werden, die Übersetzungsverhältnisse der Motoren sollten noch ein mal angepasst werden. Damit die Lüfter der Motoren bei einer Versorgungsspannung von 48V funktionieren, müssen die verbauten Platinen überarbeitet werden.

Zudem kann die Parametrierung der Motoren noch verbessert werden, um das Überschwingen unter Last zu vermindern.

Softwareseitig fehlt noch die Implementierung der Repositionierung der Beine zum Wechseln zwischen Gangarten.

Als Ausblick für aufbauende Arbeiten steht das gesamte Feld der quadrupeden Robotik offen. Die Plattform erlaubt es durch die Simulation, mittels Reinforcement Learning

Gangarten zu erlernen und diese auf der Hardware zu testen. Hierfür könnte die Arbeit "Extreme Parkour with Legged Robots" [Cheng u. a., 2023] als Grundlage dienen.

Ein weiteres interessantes Feld könnte "DeepPhase: periodic autoencoders for learning motion phase manifolds" [Starke u. a., 2022] sein. Hierbei könnte die Nutzbarkeit der DeepPhase Methode für das Wechseln zwischen Gangarten beim Laufen auf realen Robotern getestet werden. Die Softwarearchitektur ist bereits so ausgelegt, dass mit einigen Erweiterungen auch das Wechseln zwischen Gangarten während des Laufens möglich sein könnte.

Literaturverzeichnis

[Adamy 2009] Adamy, Jürgen: *Nichtlineare Regelung*. Springer, 2009

[Arm u. a. 2019] Arm, Philip ; Zenkl, Radek ; Barton, Patrick ; Beglinger, Lars ; Dietsche, Alex ; Ferrazzini, Luca ; Hampp, Elias ; Hinder, Jan ; Huber, Camille ; Schaufelberger, David ; Schmitt, Felix ; Sun, Benjamin ; Stolz, Boris ; Kolvenbach, Hendrik ; Hutter, Marco: SpaceBok: A Dynamic Legged Robot for Space Exploration. In: *2019 International Conference on Robotics and Automation (ICRA)*, 2019, S. 6288–6294

[Carr und Dycus 2016] Carr, Brittany J. ; Dycus, David: *Canine Gait Analysis*. Today's Veterinary Practice, 2016. – URL https://todaysveterinarypractice.com/wp-content/uploads/sites/4/2016/05/2016-0304_Rehab-Gait-Analysis.pdf

[Chai u. a. 2022] Chai, Hui ; Li, Yibin ; Song, Rui ; Zhang, Guoteng ; Zhang, Qin ; Liu, Song ; Hou, Jinmian ; Xin, Yaxian ; Yuan, Ming ; Zhang, Guoxuan ; Yang, Zhiyuan: A survey of the development of quadruped robots: Joint configuration, dynamic locomotion control method and mobile manipulation approach. In: *Biomimetic Intelligence and Robotics* 2 (2022), Nr. 1, S. 100029. – URL https://www.sciencedirect.com/science/article/pii/S2667379721000292. – ISSN 2667-3797

[Cheng u. a. 2023] Cheng, Xuxin ; Shi, Kexin ; Agarwal, Ananye ; Pathak, Deepak: *Extreme Parkour with Legged Robots*. 2023. – URL https://arxiv.org/abs/2309.14341

[Choi 2020] Choi, Dongil: Development of Open-Source Motor Controller Framework for Robotic Applications. In: *IEEE Access* 8 (2020), S. 14134–14145

[Gamma u. a. 2002] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. Addison-Wesley, 2002

[Goubaux und Barrier 1892] Goubaux, Armand ; Barrier, Gustave: *The Exterior of the Horse*. J. B. LIPPINCOTT COMPANY., 1892

[Haynes und Rizzi 2006] Haynes, G.C. ; Rizzi, A.A.: Gaits and gait transitions for legged robots. In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, 2006, S. 1117–1122

[Hutter u. a. 2016] Hutter, Marco ; Gehring, Christian ; Jud, Dominic ; Lauber, Andreas ; Bellicoso, C. D. ; Tsounis, Vassilios ; Hwangbo, Jemin ; Bodie, Karen ; Fankhauser, Peter ; Bloesch, Michael ; Diethelm, Remo ; Bachmann, Samuel ; Melzer, Amir ; Hoepflinger, Mark: ANYmal - a highly mobile and dynamic quadrupedal robot. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, S. 38–44

[Intel Corporation 2019] Intel Corporation: *Intel Realsense T265 Dokumentation*. https://dev.intelrealsense.com/docs/tracking-camera-t265. 2019

[Katz 2018] Katz, Benjamin: *A low cost modular actuator for dynamic robots*, Dissertation, 01 2018

[Katz u. a. 2019] Katz, Benjamin ; Carlo, Jared D. ; Kim, Sangbae: Mini Cheetah: A Platform for Pushing the Limits of Dynamic Quadruped Control. In: *2019 International Conference on Robotics and Automation (ICRA)*, 2019, S. 6295–6301

[Kuehn u. a. 2016] Kuehn, Daniel ; Schilling, Moritz ; Stark, Tobias ; Zenzes, Martin ; Kirchner, Frank: System Design and Testing of the Hominid Robot Charlie. In: *Journal of Field Robotics* 34 (2016), 07

[Lutz und Wendt 2019] Lutz, Holger ; Wendt, Wolfgang: *Taschenbuch der Regelungstechnik*. Europa-Lehrmittel, 2019

[Muralidharan u. a. 2021] Muralidharan, Seshagopalan T. ; Zhu, Ruihao ; Ji, Qinglei ; Feng, Lei ; Wang, Xi V. ; Wang, Lihui: A soft quadruped robot enabled by continuum actuators. In: *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, 2021, S. 834–840

[Odrive Robotics 2021] Odrive Robotics: *ODrive Dokumentation*. https://docs.odriverobotics.com/. 2021

[Open Robotics 2025a] Open Robotics: *ROS Dokumentation*. http://wiki.ros.org/Documentation. 2025

[Open Robotics 2025b] Open Robotics: *ROS Packages*. https://index.ros.org/packages/. 2025

[PAL Robotics S.L. 2020] PAL Robotics S.L.: *DDynamic-Reconfigure*. https://github.com/pal-robotics/ddynamic_reconfigure. 2020

[ROS core stacks 2025] ROS core stacks: *Dynamic-Reconfigure*. https://github.com/ros/dynamic_reconfigure. 2025

[ROS device drivers 2025] ROS device drivers: *rosserial*. https://github.com/ros-drivers/rosserial. 2025

[Sen u. a. 2017] Sen, Muhammed ; Bakircioglu, Veli ; Kalyoncu, Mete: Inverse Kinematic Analysis of a Quadruped Robot. In: *International Journal of Scientific & Technology Research* 6 (2017), 01, S. 285–289

[Siciliano und Khatib 2008] Siciliano, Bruno ; Khatib, Oussama: *Springer Handbook of Robotics*. Springer, 2008

[Siciliano und Khatib 2016] Siciliano, Bruno ; Khatib, Oussama: *Springer Handbook of Robotics 2nd Edition*. Springer, 2016

[Starke u. a. 2022] Starke, Sebastian ; Mason, Ian ; Komura, Taku: DeepPhase: periodic autoencoders for learning motion phase manifolds. In: *ACM Trans. Graph.* 41 (2022), Nr. 4. – URL https://doi.org/10.1145/3528223.3530178. – ISSN 0730-0301

[Unitree Robotics 2025] Unitree Robotics: *unitree_ros*. https://github.com/unitreerobotics/unitree_ros. 2025

A Anhang

A.1 Verwendete Hilfsmittel

In der Tabelle A.1 sind die im Rahmen der Bearbeitung des Themas der Bachelorarbeit verwendeten Werkzeuge und Hilfsmittel aufgelistet.

Tool	Verwendung
LaTeX	Textsatz- und Layout-Werkzeug verwendet zur Erstellung dieses Dokuments
Fusion 360	CAD- und CAM-Software, zur Erstellung von 3D-Modellen und Fräspfaden
UCCNC	CNC-Steuerungssoftware, verwendet zur Steuerung der Fräsmaschine
PrusaSlicer	Slicer-Software, verwendet zur Erstellung von Druckpfaden für den 3D-Drucker
EasyEDA	Schaltplan- und Layout-Software, verwendet zur Erstellung von Platinen
OdriveTool	Software zur Konfiguration der Umrichter
ROS	Roboter-Betriebssystem, verwendet zur Steuerung des Roboters
Gazebo	Simulationssoftware, verwendet zur Simulation des Roboters
VSCode	IDE zur Programmierung und Erstellung dieses Dokuments
draw.io	Software zum Erstellen von Diagrammen und Zeichnungen

Tabelle A.1: Verwendete Hilfsmittel und Werkzeuge

Erklärung zur selbständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort Datum Unterschrift im Original