HAW HAMBURG

Bachelorarbeit

Wolf Dieter Nickel Voelger

Entwicklung einer Low-Level Fahrsteuerung zur Vermeidung von Schäden am Untergrund für mobile allradgelenkte Robotersysteme

Fakultät Technik und Informatik *Faculty of Engineering and Computer Science*
Department Fahrzeugtechnik und Flugzeugbau *Department of Automotive and Aeronautical Engineering*

Wolf Dieter Nickel Voelger

Entwicklung einer Low-Level Fahrsteuerung zur Vermeidung von Schäden am Untergrund für mobile allradgelenkte Robotersysteme

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

Betreuender Prüfer: Prof. Dr.-Ing. Benedikt Plaumann (HAW Hamburg)
Zweitgutachter: Dipl. Ing. Roland Ortega (IDE.Automation GmbH)

Eingereicht am: 03.05.2023

Wolf Dieter Nickel Voelger

Thema der Arbeit

Entwicklung einer Low-Level Fahrsteuerung zur Vermeidung von Schäden am Untergrund für mobile allradgelenkte Robotersysteme

Stichworte

Autonome Robotik, Landwirtschaftsroboter, ROS2, CANopen, Gazebo, Bodenverdichtung

Kurzzusammenfassung

In der vorliegenden Bachelor-Thesis wird dokumentiert, wie die Fahrsteuerung einer universellen Roboterbasis implementiert wird. Neben der Fahrsteuerung wird zudem ein überlagerter Algorithmus entwickelt, der Bodenbeschädigungen beim Lenken reduzieren soll. Die Umsetzung der Fahrsteuerung erfolgt auf einem Mikrocontroller, der von einem zweiten Mikrocontroller die Sollwerte für die Fahrtrichtung erhält. Die Fahrsteuerung wird mithilfe eines Simulationstools erstellt und getestet. Abschließend werden die Ergebnisse der simulierten Trajektorie bewertet und ein Ausblick auf weitere Entwicklungspotenziale gegeben.

Title of Thesis

Development of a low-level driving control system to avoid damage to the ground for mobile all-wheel steered robot systems

Keywords

Autonomous robotics, agricultural robot, ROS2, CANopen, Gazebo, soil compaction

Abstract

This bachelor thesis documents how the driving control of a universal robot base is implemented. In addition to the driving control, an overlaid algorithm is also determined and is intended to reduce ground damage during steering. The implementation of the driving control takes place on a microcontroller that receives setpoints for the driving direction from a second microcontroller. The driving control is created and tested with the help of a simulation tool. Finally, the results of the simulated trajectory are evaluated and an outlook on further development potentials is given.

Inhaltsverzeichnis

Abkürzungen

CAD Computer Aided Design

CAN Controller Area Network

CiA CAN in Automation

COB Communication Object Identifier

CRC Cyclic Redundancy Check

DAE Digital Asset Exchange File

DRZ Deutsches-Rettungsrobotik-Zentrum

IDE Integrated Development Environment

IMU Inertial Measurement Unit

KI Künstlicher Intelligenz

KRIBL Kleine Roboter für die Intelligente Biologische Landwirtschaft

NMT Network Management

OSI Open System Interconnection

PDO Process Data Object

ROS Robot Operating System

SDF Simulation Description Format

SDO Service Data Object

STL Standard Triangle Language

URDF Unified Robot Description Format

xacro XML Macros

XML Extensible Markup Language

1 Hinführung vor dem Hintergrund der Aufgabenstellung

Das Bundesministerium für Ernährung und Landwirtschaft möchte im Rahmen des Kleine Roboter für die Intelligente Biologische Landwirtschaft (KRIBL)-Projekts kleine Robotersysteme für ökologische, landwirtschaftliche Betriebe ermöglichen. Dazu zählen kleine Betriebe, die sich große, schwere und hoch automatisierte Maschinen nicht leisten können. Ziel des Vorhabens ist es, die Entwicklung eines mobilen Basis-Robotersystems voranzutreiben, an das verschiedene technische Lösungen im Baukastenprinzip installiert werden können. Dazu zählen beispielsweise ein Roboterarm, Kameras oder Sensoren.

Ein Konzept, bestehend aus einem Grundgestell, Akkumulatoren, einem Computer, einem Mikrocontroller und vier individuell ansteuerbaren Lenkachsen mit jeweils einem Radnabenmotor, wurde bereits für das Basis-Robotersystem angefertigt. Im Verlauf dieser Bachelorarbeit soll für das Basis-Robotersystem die Fortbewegung durch Ansteuerung der Lenkachsen und der Radnabenmotoren ermöglicht werden. Dabei soll die Steuerungslogik in einer Low-Level Hardware, dem Mikrocontroller, vom Computer abgekapselt werden. Die Fahrtrichtung des Robotersystems soll über einen Geschwindigkeitsvektor und einer Rotationsgeschwindigkeit festgelegt werden, die von einem übergeordneten System, dem Computer, bereitgestellt werden.

Um Schäden am Ackerboden zu vermeiden, soll möglichst auf eine Drehwinkeländerung im Stand verzichtet werden. Die Herausforderung dieser Arbeit liegt darin, einen Algorithmus zu entwickeln, der als übergeordnete Instanz in jedem Fahrmodus greift und die Standlenkung verhindert. Die gewünschte Trajektorie ist jedoch schnellstmöglich einzuhalten, wobei die Manövrierfähigkeit vor dem Schutz des Bodens Vorrang hat. Bei engen Platzverhältnissen sollen enge Kurvenradien dennoch möglich sein. Diese Arbeit gilt als erfolgreich, wenn das Basis-Robotersystem durch die Eingabe von oben genannten Sollwerten verfahren kann.

1.1 Das Projekt KRIBL

Das Projekt KRIBL wird von der Universität zu Lübeck am Institut für Robotik und Kognitive Systeme koordiniert. Ein Projektpartner ist unter anderem die Firma IDE.Automation GmbH aus Lübeck. Die IDE.Automation GmbH ist ein mittelständisches Unternehmen im Bereich des Sondermaschinenbaus und liefert im Rahmen des KRIBL-Projektes Know-How aus dem Bereich der Industrie. Dazu gehören Konzeption, Aufbau, Programmierung, Inbetriebnahme und Verdrahtung des Basisroboters. Im Rahmen dieser Bachelorarbeit findet die Inbetriebnahme sowie die Programmierung der Fahrsteuerung statt.

1.2 Hintergrund

Durch die steigende Anzahl der Bevölkerung auf unserem Planeten wird immer mehr Nahrung benötigt. Dabei sind Anbauflächen sowie die Anzahl an menschlichen Arbeitskräften begrenzt. Ein Trend, der sich seit dem letzten Jahrhundert abzeichnet, ist die Nutzung von automatisierten technischen Gerätschaften. So besitzen moderne Traktoren bereits selbstfahrende Systeme, die eine landwirtschaftliche Fläche bearbeiten können[10]. Bei der Automatisierung geht es um die optimierte Nutzung von Ressourcen. Ziel ist es, mehr Ertrag aus einer definierten Fläche zu generieren. Zusätzlich sollen laufende Betriebskosten reduziert werden. Dabei ist nicht nur der wirtschaftliche Aspekt zu bedenken, sondern auch der ökologische. Der Einsatz von Pestiziden soll für höheren Ertrag sorgen. Jedoch belasten Pestizide die Umwelt, da diese ins Grundwasser gelangen und die Artenvielfalt zerstören[11].
Der Einsatz von Robotik in der Landwirtschaft soll unter anderem den Arbeitskräftemangel, sowie Umweltschädigung durch den Einsatz von Pestiziden entgegen wirken. Dies geschieht durch den Verzicht auf Pestizide, indem Unkraut beispielsweise mechanisch entfernt wird.
Bereits auf dem Markt erhältliche Roboter sind für kleinere ökologische Betriebe finanziell nicht realisierbar; daher liegt der Fokus von KRIBL auf Kostenoptimierung. Das Projekt KRIBL wurde ins Leben gerufen, um den Einsatz von Künstlicher Intelligenz (KI) zu entwickeln und eine Roboterbasis zu konzeptionieren, die im Baukastenprinzip nach den Bedürfnissen des Landwirts erweitert werden kann. Dabei sollen Kamerasysteme, Sensorik und Aktoren, wie beispielsweise ein Roboterarm, an der Roboterbasis montiert werden können. Einige Anwendungsszenarien werden nachfolgend vorgestellt[3]:

Autonomes Monitoring von Unkraut und Wildwuchs

Ein autonomer Roboter fährt über die Pflanzenreihe und registriert sowohl den aktuellen Wachstumszustand, als auch vorhandene Unkräuter. Diese können ohne den Einsatz von Pestiziden mechanisch vom Roboter entfernt werden.

Erkennung und Abwehr von Raubvögeln und Raubtieren in der offenen Geflügelhaltung

Ein vollautonomer Roboter fährt auf offenem Feld und registriert mithilfe von Sensorik jagende Raubtiere und Raubvögel und wehrt Angriffe tiergerecht ab.

Fahren in unebenem Gelände

Der Roboter fährt auch in unebenem Gelände und kann durch Bewuchsmonitoring feststellen, ob Weidezäune freigeschnitten werden müssen.

Zusätzlich zur Roboterbasis soll ein Algorithmus eingeführt werden, um den Ackerboden bei Lenkung zu schonen. Eine plastische Verformung des Bodens führt zu einer höheren Bodenverdichtung. Diese beeinträchtigt das Wurzelwachstum und die Wasserinfiltration, welche den Ernteertrag schmälert[2].

2 Stand der Technik

In diesem Abschnitt wird der aktuelle Stand der Technik vorgestellt. Zunächst wird die in der Roboterbasis verwendete Hardware betrachtet. Anschließend wird auf das Controller Area Network (CAN)Open-Protokoll eingegangen, welches nötig ist, um die Antriebe mit dem Mikrocontroller anzusteuern.

Um einen Lösungsansatz für einen Algorithmus für das Lenken im Stand zu finden, werden die am Rad herrschenden Kräfte genauer untersucht. Die Roboterbasis wird auf dem Robot Operating System (ROS) Framework aufgebaut. ROS2 hat in der Robotik aufgrund seiner Softwarearchitektur eine große Beliebtheit erlangt und gilt bereits als Standard für Robotikanwendungen[21].

2.1 Verwendete Hardware & Anforderungen

Durch die Vorüberlegungen der Projektpartner gibt es Vorgaben bezüglich der zu verwendenden Hardware (Tabelle 2.1). Diese Vorgaben bilden die Grundlagen der Arbeit.

Bezeichnung	Aufgabe
MicroGiant-Board	Low-Level-Verarbeitung der Sollwerte
Nvidia Jetson Nano	Bereitstellen der Sollwerte
Nanotec PD4-E591	Lenkmotor
FLD48V300W2211	Nabenmotor mit Ackerschlepprad
Fritz!Box Router	WLAN-Router
Phoenix Contact Switch	LAN-Switch
Playstation 4 Controller	Erzeugen von Sollwerten
Intel Nuc	Computer zum Speichern und Auswerten von Bilddaten

Tabelle 2.1: Verwendete Hardware im Roboterbasisystem

Neben der Hardware wird vom Projektpartner IDE.Automation die Roboterbasis gestellt. In Abbildung 2.1 ist die Roboterbasis im frühen Entwicklungsstadium dargestellt.

Das MicroGiant-Board ist eine vom Deutsches-Rettungsrobotik-Zentrum (DRZ) gefertigte Platine mit einem Espressif ESP32 Mikroprozessor. Diese stellt die Low-Level-Implementierungsschicht dar, womit die Antriebe (Lenk- und Nabenmotoren) mit einer hardwarenahen Programmierung mit Fahrbefehlen versorgt werden.

Der Nvidia Jetson Nano, im folgenden Jetson genannt, ist ein zusätzlicher Mikrocontroller, der für Künstliche Intelligenz-Anwendungen geschaffen wurde. Dieser stellt den Sollgeschwindigkeits- und Rotationsgeschwindigkeitsvektor bereit.

Für einen kabellosen Betrieb auf dem Feld soll über einen WLAN-Accesspoint und einen LAN-Switch ein kabelloser Zugang zum Jetson bereitgestellt werden.

Für den Roboterbasis-Prototyp werden mithilfe eines Playstations 4 Controllers die Fahrtrichtung, beziehungsweise die Sollwerte generiert. Zusätzlich wird ein Intel Nuc Computer verbaut, auf welchem Bildverarbeitungsalgorithmen ausgeführt werden sollen. Dieser Aufgabenbereich ist Teil des Arbeitspaketes der Universität zu Lübeck und wird deshalb nicht näher betrachtet.

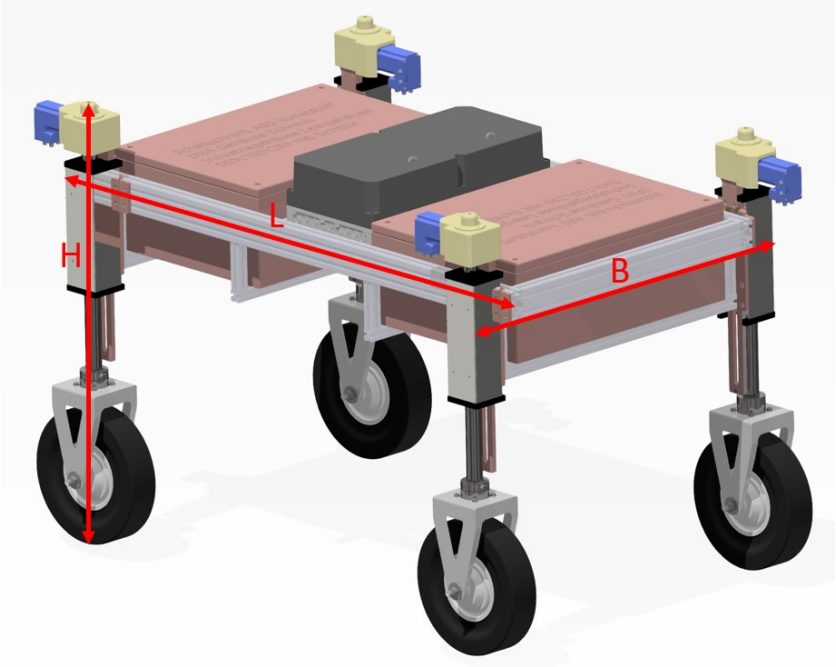

Abbildung 2.1: Design der Roboterbasis mit den Außenmaßen B: 1,12 m, L: 1,65 m, H: 1,10 m, m: 180 kg

Ein wichtiger Aspekt des in Abbildung 2.1 gezeigten Designs stellt die Anpassungsfähigkeit an unterschiedliche Anbauarten und -breiten dar. In diesem Kontext wurde ein Konzept

entwickelt, das einen Roboter mit einer Spurbreite von 1,5 m für Gemüse und einer Spurbreite von 0,5-0,75 m für Mais umfasst. Dabei ist geplant, dass der Roboter mit seiner kurzen Seite über die schmale Spur fährt und mit der langen Seite über die breite Spur. Dies ermöglicht einen flexiblen Einsatz in verschiedenen Feldern und Kulturen.

Für eine optimale Manövrierfähigkeit und Wendigkeit sieht das Konzept eine Vierradlenkung vor. Hierdurch kann der Roboter auch auf engen Feldern und bei der Navigation um Hindernisse flexibel eingesetzt werden. Darüber hinaus bietet die Höhe des Roboters weitere Anpassungsmöglichkeiten hinsichtlich weiterer Aufbauten, wie beispielsweise einem Roboterarm. In Abbildung 2.2 ist die Roboterbasis mit einem montierten Roboterarm dargestellt.

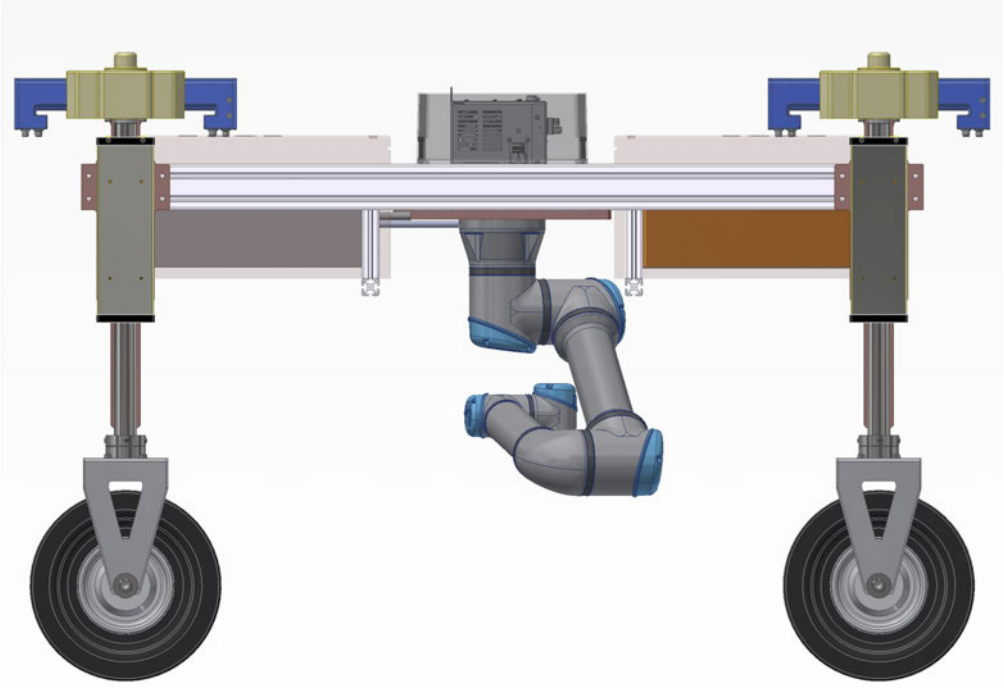

Abbildung 2.2: Design der Roboterbasis mit montiertem Roboterarm

2.2 CANopen

CANopen ist ein Kommunikationsprotokoll, welches auf CAN basiert. Das CAN-Protokoll ist ein serieller Feldbus und wurde ursprünglich für Kraftfahrzeuge entwickelt. Es beschreibt die unteren zwei Schichten des Open System Interconnection (OSI)-Refererenz -modells. CAN gilt als eines der zuverlässigsten Protokolle, welches zeitgleich sehr flexibel ist. Die hohe Zuverlässigkeit wird unter anderem durch eine differentielle Übertragung, die Nutzung eines Cyclic Redundancy Check (CRC)-Polynoms und weiteren fehlererkennenden Maßnahmen erreicht. Die Übertragungsrate ist bis zu 1 $\frac{Mbit}{s}$ möglich. Zusätzlich ist es echtzeitfähig. Ein Bussystem gilt dann als echtzeitfähig, wenn Nachrichten immer zu einem definierten Zeitpunkt auftreten[7].
Durch die Flexibilität hat das CAN-Protokoll Einzug in die verschiedensten Bereichen gehalten. Darunter fällt auch die Automatisierungstechnik. Die CAN in Automation (CiA) hat, bezogen auf das OSI-Refererenzmodell, die Applikations-Schicht für das CAN-Protokoll normiert. Dadurch sind Geräte- und Anwendungsprofile entstanden, mit denen die Implementierung und Inbetriebnahme von Gerätschaften erleichtert werden sollen.
CANopen spezifiziert beispielsweise die Start-Sequenz, aber auch das applikative Verhalten eines Gerätes. So legt die Spezifikationsreihe CiA 301 das Kommunikationsprofil dar. Für diese Arbeit ist die Spezifikationsreihe CiA 402 interessant, da es das Kommunikationsprofil für elektrische Antriebe spezifiziert[23]. In diesem ist beschrieben, in welchem Speicherregister, Parameter wie eine Drehzahl gesetzt werden, aber auch, aus welchem Register aktuelle Drehzahlen ausgelesen werden können.
Da die Low-Level-Fahrsteuerung der Antriebe über das MicroGiant-Board erfolgen soll, muss das CANopen-Protokoll auf dem Mikroprozessor ESP32 implementiert werden. Dazu wird auf die Open-Source-Bibliothek „CANopenNode“[1] zurückgegriffen. Diese bietet die Basis für den CANopen-Standard. Dafür ist es erforderlich, die Anwendungsschicht auf das spezifische Projekt angepasst zu implementieren. Die projektspezifische Implentierung ist in Unterabschnitt 5.2.2 dokumentiert.

[1] https://github.com/CANopenNode/, letzter Zugriff: 14.04.2023

Die wichtigsten Dienste dieser Arbeit sind der Network Management (NMT)-Dienst, der Service Data Object (SDO)-Dienst und der Process Data Object (PDO)-Dienst. Es existieren weitere, jedoch sind diese für die erste prototypische Entwicklung nicht notwendig, da sie über die Grundfunktionalität hinaus gehen. Die genannten Dienste werden im Folgenden genauer erläutert.

2.2.1 Network Management (NMT)

Der Network Management Dienst ist für das Setzen der Betriebszustände (Operational-State) aller Teilnehmer im Bus verantwortlich. Basierend auf einer Master/Slave Topologie sendet der Master den gewünschten Operational-State. Der Operational-State wird in der State-Machine des jeweiligen Gerätes verarbeitet. Die möglichen Zustände sind:

- Initialization
- Pre-Operational
- Stopped
- Operational

Jeder Bus-Teilnehmer durchläuft gemäß der CANopen-Spezifikation beim Einschalten die Initialisierungsphase. Diese startet bei *Initialization* und endet bei *Operational*. Im Zustand *Operational* ist der Teilnehmer bereit für Konfigurations- und Startbefehle. Besonders in der Phase der Inbetriebnahme ist das NMT-Protokoll interessant, da jeder Teilnehmer beim Übergang von *Initialization* zu *Pre-Operational* eine Boot-Up Nachricht sendet. Diese bildet die Anmeldung ins Netzwerk und bestätigt eine korrekte Hochlaufphase des Teilnehmers.

Für den Betrieb des Roboters ist ein NMT-Befehl zu senden, um die Antriebe in den Zustand *Operational*, beziehungsweise „betriebsbereit“ zu versetzen. Dafür ist unter anderem die Node-ID nötig. Die Node-ID ist eine einzigartige Identifikationsnummer und wird vom Anwender festgelegt. Sie dient zur Adressierung der angeschlossenen Bus-Teilnehmer[23].

Tabelle 2.2 zeigt den Aufbau einer Nachricht, um über einen NMT-Befehl alle Bus-Teilnehmer in den Status *Operational* zu versetzen.

Hinweis: Bei der Schreibweise mit 0x00 handelt es sich um die hexadezimale Schreibweise

ID	Byte 0	Byte 1
0x00 = Node-ID des NMT Commands	CMD 0x01 = switch to operational 0x02 = switch to stop 0x80 = switch to pre-operational 0x81 = reset node 0x82 = reset communication	Node-ID des zu erreichenden Teilnehmers 0x00 = Alle Teilnehmer
0x00	0x01	0x00

Tabelle 2.2: Aufbau des NMT-Befehls[13]

2.2.2 Servicedatenobjekte (SDO)

Mithilfe von Servicedatenobjekten lassen sich CANopen-Bus-Teilnehmer konfigurieren und Daten aus dem Objektverzeichnis lesen oder schreiben. Darüber hinaus wird es für Diagnoseaufgaben verwendet. Das Telegramm besteht aus acht Bytes, wobei vier Bytes für Befehle und Adressierung zur Verfügung stehen. Die weiteren vier Bytes beinhalten die CAN-Nachricht. Die Kommunikation über SDO findet weitesgehend über die Node-ID 0x600 + Node-ID (empfangen) und 0x580 + Node-ID (senden) statt[23]. Der Aufbau des Telegramms ist in Tabelle 2.3 dargestellt.

Zweck	Adressierung				Daten			
Bytereihenfolge	HB	LB	HB	HB	LB	HB	LB	HB
ID	B0	B1	B2	B3	B4	B5	B6	B7
0x580 + Node-ID	SDO Identifier	Zu lesendes/ schreibendes Register		Subindex	Zu lesende oder zu schreibende Daten			

Tabelle 2.3: Aufbau einer SDO-Nachricht. B=Byte, LB=LowByte, HB=HighByte

Die Node-ID ist der zu erreichende Bus-Teilnehmer. Der SDO-Identifier beschreibt einen Befehl, wie zum Beispiel eine Lese- oder Schreibaufforderung. Diese sind in Abbildung 2.3 abgebildet.

In Byte eins und zwei ist das zu lesende oder zu schreibende Register festgelegt. Der Subindex definiert, welche Stelle des Registers angesprochen werden soll. Die letzten vier Bytes beinhalten die CAN-Nachricht.

Command	Description	User data	Function
22h	SDO(rx), Download Request	undefined	Send parameters to sensor
23h		4 bytes	
2Bh		2 bytes	
2Fh		1 bytes	
60h	SDO(tx), Download Response	-	Confirm parameter transfer to client
40h	SDO(rx), Upload Request	-	Request parameters from sensor
42h	SDO(rx), Upload Response	undefined	Send parameters to client
43h		4 bytes	
4Bh		2 bytes	
4Fh		1 bytes	
80h	SDO(tx), Abort Domain Transfer	4 bytes	Sensor sends error code to client

Abbildung 2.3: SDO-Identifier[27]

2.2.3 Prozessdatenobjekte (PDO)

Um möglichst viele Daten beziehungsweise Informationen in einem Buszyklus bereitstellen zu können, bietet CANopen den Dienst PDO an. Im Vergleich zur Kommunikation mit SDO ist es beim PDO möglich, CAN-Nachrichten mit einer maximalen Größe von acht, statt vier Bytes zu senden. Möglich wird dies durch die Vermeidung von Overhead im Telegramm. Overhead ist ein Nachrichteninhalt, der durch Identifizierung und Addressierung entsteht aber nicht dem eigentlich Inhalt dienlich ist. Die ersten vier Bytes der Nachricht bilden den Overhead, der zu Konfigurations- und Addressierungszwecken verwendet wird. Reduziert wird dieser durch die Festlegung von bereits vorkonfigurierten Nachrichten. Zusätzlich wird dadurch die Echtzeitfähigkeit des Busses hergestellt. Für die Nutzung des PDO-Protokolls muss das Objectdictionary eines Gerätes über Transfer- und Empfangsobjekte verlinkt werden. Das Objectdictionary beschreibt die Hardwarekonfiguration des jeweiligen CAN-Teilnehmers. Hier ist hinterlegt, welche Information sich in welchem Speicherregister befindet. Dazu zählen zum Beispiel Gerätename, Gerätehersteller, aber auch die Adressbereiche zum Verarbeiten einer Empfangs- oder Sendenachricht[23].

In Tabelle 2.4 werden Registerbereiche gezeigt, die konfiguriert werden können, um das PDO-Protokoll zu nutzen. Um beispielsweise das Register 0x1600 für die Übertragung via RPDO (Receive-PDO) zu nutzen, wird die Konfiguration im Register 0x1400 festgelegt. Das nächste PDO-Register (0x1601) wird über 0x1401 festgesetzt. Es folgt also die Regel: Register 0x1600 + n wird von 0x1400 + n konfiguriert.

Analog für das RPDO wird das TPDO (Transmit-PDO) eingestellt: Register 0x1A00 + n wird von 0x1800 + n konfiguriert.

Registerbereich	Bezeichnung
0x1400-0x15FF	RPDO-Kommunikation: Konfiguration des RPDO-Mappings wie CAN-ID, Transmission-Type
0x1600-0x17FF	RPDO-Mapping
0x1800-0x19FF	TPDO-Kommunikation: Konfiguration des TPDO-Mappings wie CAN-ID, Transmission-Type, Inhibittime.
0x1A00-1BFF	TPDO-Mapping

Tabelle 2.4: Registerbereiche für die Nutzung von PDO

Abbildung 2.4 zeigt beispielhaft eine Kommunikation über PDO, um eine Nachricht von einem Gerät (Mikrocontroller) zum Anderen (Motor A) zu senden. Zuerst müssen die Geräte konfiguriert werden.

In diesem Beispiel sollen die in den Registern 0x6000 und 0x6010 gespeicherten Nachrichten über das Mappingregister 0x1A0 versendet werden. Dazu wird zunächst das Mappingregister über das TPDO Kommunikationsregister 0x1800 eingestellt. Hier wird ein Communication Object Identifier (COB) gewählt. Standardmäßig wird über das Register 0x1800 die COB-ID aus der ID 180 + Node-ID erstellt. In diesem Fall ergibt es die COB-ID 181.
Weiterhin wird im Register 0x1800 eingestellt, an welcher Stelle im Telegramm die Daten gesetzt werden. Die Daten aus dem Register 0x6000 und 0x6010 bestehen jeweils aus vier Bytes und werden durch das Register 0x1A00 versendet.

Auf der Empfangsseite des Motors wird das Register 0x1600 durch das Register 0x1400 eingestellt. Hier wird bekannt gegeben, an welcher Stelle im Telegramm welche Nachricht zu erwarten ist. Das Register 0x1600 weist den Registern 0x6042 und 0x6100 jeweils die empfangenen Nachrichten zu.

Der Transmissiontype gibt an, ob die Nachricht (a)synchron gesendet/empfangen werden soll. Synchron bedeutet, dass eine Nachricht zu einem definierten Takt gesendet wird. Ein asynchroner Versand der Nachricht erfolgt immer bei einer Wertänderung. Zusätzlich kann eine Inhibittime und Eventtime eingerichtet werden. Die Inhibittime ist eine Zeit, in der das Register keine Nachricht rausschickt, obwohl eine Wertänderung stattgefunden hat. Die Eventtime ist eine Zeit, in der das Register spätestens eine Nachricht rausgeschickt[23].

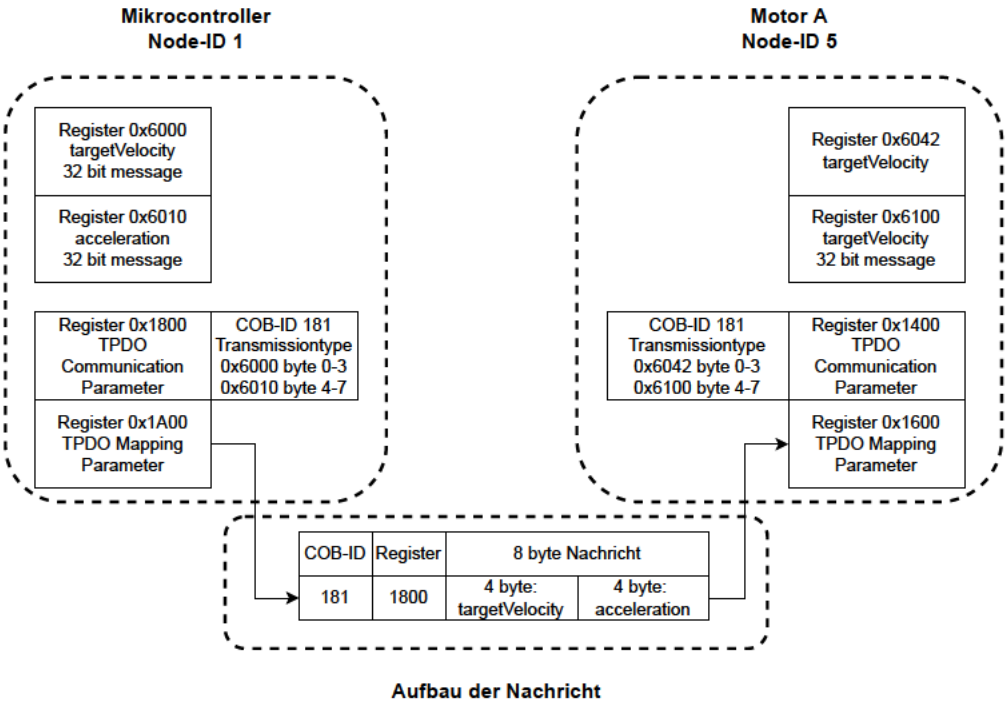

Abbildung 2.4: Beispielhaftes TPDO-Mapping

An dieser Stelle sei erwähnt, dass die gewählten Register ein Beispiel sind. Für die Anwendung müssen die Angaben des Herstellers beachtet werden, da jedes Objectdictionary ab der Anwendungsschicht unterschiedlich aufgebaut ist.

Für diese Arbeit wird es nötig sein, ein Objectdictionary für den Mikrocontroller zu erstellen, sowie die Konfiguration der Sende- und Empfangsregister vorzunehmen.

2.3 Kräfte am Rad

In diesem Kapitel werden die Kräfte, welche am Rad beim Fahren zwischen Reifen und Ackerboden entstehen, analysiert. Zusätzlich wird der Fokus auf die Zusammenhänge von Geschwindigkeit und Radstellung beim Lenken gesetzt. Die Arbeitsgeschwindigkeit des Basisroboters auf dem Feld ist auf maximal $1,5\,\frac{m}{s}$, die maximale Fahrzeuggeschwindigkeit ist auf $2,5\,\frac{m}{s}$ festgelegt.

Die entstehenden Reibungskräfte zwischen Fahrbahn und Reifen hängen vom Schlupf ab. Der Schlupf ist das Verhältnis zwischen Radumfangsgeschwindigkeit zur Geschwindigkeit des Fahrzeuges. Für ein angetriebenes Rad lässt sich der Schlupf in Reifenlängsrichtung κ_A (auch Längsschlupf genannt) wie folgt definieren[4]:

$$\kappa_A = \frac{\omega \cdot r - \nu_x}{\omega \cdot r} \tag{2.1}$$

$$\kappa_A = 1 - \frac{\nu_x}{\omega \cdot r} \tag{2.2}$$

Ein durchdrehendes Rad, welches keine Fahrzeugbewegung bewirkt, erzeugt einen Schlupf von 1. Ein frei rollendes Rad, dessen Radumfangsgeschwindigkeit $\omega \cdot r$ der Fahrzeuggeschwindigkeit v_x gleicht, erzeugt einen Schlupf von 0. Wenn Schlupf auftritt, ist die Radumfangsgeschwindigkeit größer als die Geschwindigkeit des Fahrzeuges. Ein hoher Schlupf bedeutet also eine geringere Reibungskraft und damit auch eine geringere Traktion des Rades.

Die Traktion Fx_{max} des Rades ist jedoch entscheidend für die Fortbewegung des Fahrzeugs, da sie verhindert, dass das Rad durchdreht und sich im Boden festfährt. Der Kraftschlussbeiwert μ_h proportional zur Radlast F_z ergibt Fx_{max}[4].

$$Fx_{max} = \mu_h \cdot F_z \tag{2.3}$$

Der Kraftschlussbeiwert wird maßgeblich vom Schlupf bestimmt. Dabei haben die Witterung, sowie die Reibpartner aus dem Reifenmaterial und dem Untergrundmaterial die größte Auswirkung auf den Schlupf. Für einen Offroad-Reifen und Sand liegt μ_h bei 0,6. Eine trockene Fahrbahn und Offroad-Reifen wird zu 0,75 angenommen[9].

Wie in Abbildung 2.5 zu sehen ist, liegt der höchste Kraftschluss bei μ_h an. An dieser Stelle entsteht der kritische Schlupf und wird je nach Literatur mit 0,1 bis 0,3[26][4] angegeben.

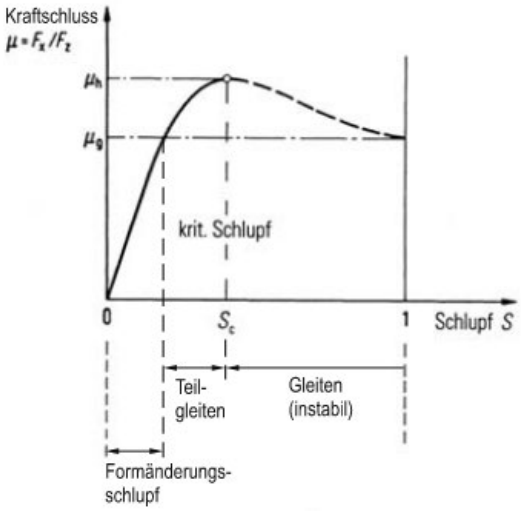

Abbildung 2.5: Kraftschluss-Schlupf-Kurve[12]

Neben dem Längsschlupf existiert der Querschlupf. Dieser entsteht bei Kurvenfahrt, sobald Seitenkräfte auf das Rad wirken. Definiert wird der Querschlupf S_α mit

$$S_\alpha = \frac{\nu_{w,quer}}{\nu_{w,längs}} = \frac{\nu_w \cdot \sin(\alpha)}{\nu_w \cdot \cos(\alpha)} \tag{2.4}$$

Dabei beschreibt ν_w, wie in Abbildung 2.6 verdeutlicht, den Geschwindigkeitsvektor in Fahrzeugrichtung. Die Indizes $\nu_{w,quer}$ und $\nu_{w,längs}$ stehen für den Quer- und Längsanteil der resultierenden Fahrzeuggeschwindigkeit. Der Winkel zwischen eingeschlagenem Rad und der Fahrtrichtung (im Bild als Vektor ν_w dargestellt) des Fahrzeuges wird auch Schräglaufwinkel α genannt.

Aus den trigonometrischen Beziehungen lässt sich weiterhin ableiten, dass Gleichung 2.4 zu

$$S_\alpha = \tan(\alpha) \tag{2.5}$$

umgeformt werden kann. Damit ist der Querschlupf vom Schräglaufwinkel abhängig. Für niedrige (Dreh-)Geschwindigkeiten und zur einfacheren Berechnung kann der Schräglaufwinkel als Lenkwinkel angenommen werden[4].

Abbildung 2.6: Schräglaufwinkel am Reifen[4]

Weiterhin lassen sich die Zusammenhänge zwischen Antriebs- und Querschlupf am Kammschen Kreis in Abbildung 2.7 verdeutlichen. In diesem ist zu erkennen, dass bei einem theoretischen Lenkwinkel von ±90° kein Vortrieb mehr möglich ist, da nur noch Seitenkräfte wirken. Bei einem Lenkwinkel von 0° existieren nur noch Umfangskräfte, die für den Vortrieb verantwortlich sind. Ein Lenkwinkel von $-90° < Lenkwinkel < 90°$ mit $Lenkwinkel \neq 0°$ treten stets Umfangs- und Seitenkräfte und somit auch Antriebs- und Querschlupf auf.

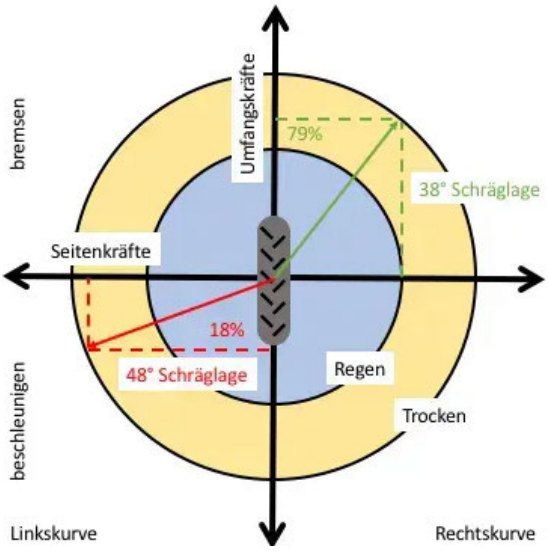

Abbildung 2.7: Kammscher Kreis[1]

Weiterhin zeigt die Abbildung 2.7 zwei Bereiche, die mit *Regen* und *Trocken* betitelt sind. Damit werden Bereiche angegeben, die auf den Schlupf und die übertragbaren Kräfte hinweisen. Bei Regen sinkt der Kraftschlussbeiwert. Gleichung 2.3 verdeutlicht diesen Zusammenhang. Bei kleinerem μ_h sinkt auch Fx_{max}.

2.4 Bodenverdichtung

In diesem Kapitel wird der Zusammenhang zwischen Bodenverhältnissen und Schlupf erläutert. Beim Fahren auf festem Untergrund, wie bspw. einer asphaltierten Straße, entsteht der Vortrieb durch Mikroverschweißungen zwischen Fahrbahn und Rad, sowie dem elastischen Verhalten des Rades. Die Befahrbarkeit von Ackerboden ist davon abhängig, wieviel Schubkraft der Boden aufnehmen kann. Die Bodendeformation ist aufgrund der vielen Parameter des Bodens ein hoch komplexes Thema. Faktoren, die Einfluss auf die aufnehmbare Schubkraft haben, sind die Kornverteilung und Kornform im Gefüge, die Kohäsionskräfte und die Permeabilität, um nur einige zu nennen[2]. Je tiefer ein Reifen einsinkt, desto mehr Kraft wird aufgrund des erhöhten Rollwiderstandes benötigt. Um die vielen Faktoren des Bodens nicht genauer beachten zu müssen, kann der Schlupf herangezogen werden. Der Schlupf ist abhängig von

- der Deformation des Reifens,
- den Deformation des Ackerbodens und
- den Gleiten des Reifens auf dem Boden durch abgescherte Bodenteile[29]

Die Deformation des Reifens wird hier nicht näher betrachtet, da sie im Vergleich zur Deformation des Ackerbodens viel kleiner ist. Abgescherte Bodenteile entstehen, wenn ein Reifen auf der Stelle oder mit niedriger Drehzahl gelenkt wird. Dabei wird die Kohäsionskraft des Untergrundes überschritten und die Bodenbestandteile lösen sich voneinander. Das Gleiten von Reifen entsteht, wenn das Profil zugesetzt wird[28]. Bei Offroad-Anwendungen füllen sich die Zwischenräume des Profils mit Bodenmaterial. Im Bereich der Kraftfahrzeugtechnik ist dieser Effekt mit Aquaplaning gleichzusetzen.

2.5 Lenkmethoden

Bei Kraftfahrzeugen und Landmaschinen hat sich vorwiegend die Ackermannlenkung durchgesetzt. Hier werden die zwei Vorderrädern gelenkt, wobei das kurveninnere Rad

einen größeren Lenkwinkel einschlagen muss als das kurvenäußere. In Abbildung 2.8 wird dieser Zusammenhang grafisch dargestellt. Die Normalen der Radmittelebenen führen auf einen gemeinsamen Punkt hin. Dieser wird Momentanpol M des Fahrzeuges genannt und liegt bei der Ackermannlenkung auf Höhe der Hinterachse. Der Momentanpol M stellt den Punkt dar, um den sich das Fahrzeug dreht.

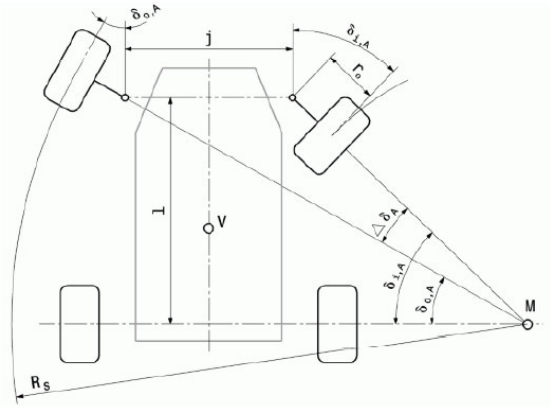

Abbildung 2.8: Schematische Darstellung der Ackermannlenkung[20]

Die Allradlenkung wurde entwickelt, um ein höheres Maß an Wendigkeit zu erzielen. Bei dieser Methode lenken neben den Vorderrädern zusätzlich noch die Hinterräder. Das hat zur Folge, dass der Momentanpol des Fahrzeuges verändert wird. Wie in Abbildung 2.9 veranschaulicht, erzeugt das zusätzliche Lenken der Hinterräder eine Verschiebung des Momentanpols *M1* zum Fahrzeugschwerpunkt (vergleiche *M1* zu *M2*) hin. Das hat zur Folge, dass ein geringerer Lenkradius benötigt wird und somit die Wendigkeit erhöht[20] wird.

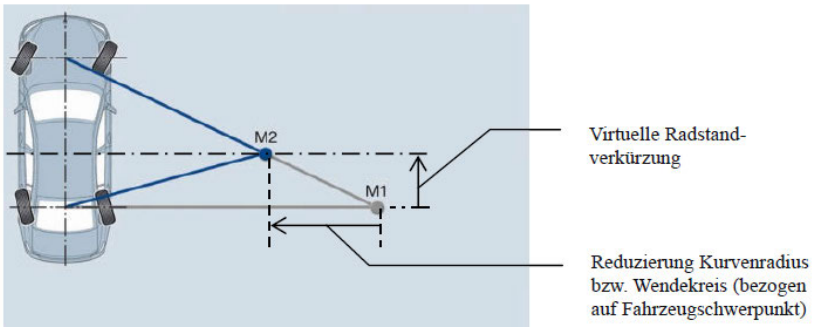

Abbildung 2.9: Vergleich zwischen Ackermann- und Allradlenkung[20]

2.6 ROS2 Framework

ROS2 ist ein quelloffenes Framework für die Entwicklung von Robotik-Software. Es bietet eine Sammlung von Tools, Bibliotheken und Konventionen, welche die Entwicklung von Robotik-Anwendungen erleichtern. ROS unterstützt die Skalierung von einfachen bis hin zu komplexen Robotersystemen, indem es eine verteilte Architektur verwendet. Das Framework wird in vielen Anwendungen eingesetzt, darunter autonome Fahrzeuge, Drohnen, die industrielle Automatisierung und Robotikforschung. Robotersysteme bestehen aus einer Mechanik und einer steuerenden Software. Die Entwickler von ROS2 stellten dabei sicher, dass es auf verschiedenen Plattformen wie Windows 10, Linux und MacOS installiert werden kann. Der Kerngedanke von ROS beruht auf „*Don't reinvent the wheel*“[17]. Die Idee ist es, dass bereits über Software gelöste Probleme in einer Sammlung bereitgestellt werden. ROS2 stellt daher eine Vielzahl von Tools und Bibliotheken zur Verfügung, die die Entwicklung von Robotik-Software erleichtern, wie beispielsweise Treiber für Sensoren und Aktuatoren, 3D-Visualisierungswerkzeugen und Simulationsumgebungen. Um das Framework hat sich seit seiner Einführung eine große Community entwickelt, welche Open-Source-Projekte zum Beispiel über github veröffentlicht. So können Hersteller von Motoren ihre Software zur Ansteuerung von Motortreibern veröffentlichen, so dass sich die Inbetriebnahmezeit erheblich verkürzt.

Im Folgenden wird die für diese Arbeit relevante Anwenderebene von ROS2 erläutert.

2.6.1 Anwendung von ROS2

ROS bietet ein Netzwerk von Prozessen, die miteinander kommunizieren und Daten austauschen können. Das Austauschen von Daten wird über ein Publisher- und Subscribersystem realisiert. Die relevanten Basiskomponenten eines ROS-Projektes sind *Nodes*, *Publisher*, *Subscriber*, *Topics*, *Services* und *Actions*[21].

Ein *Node* bildet eine Sinneinheit einer bestimmten Funktionsgruppe und könnte beispielweise einen Sensor oder einen Aktor darstellen.

Ein *Publisher* in einem Node stellt eine Nachricht asynchron über einen definierten Datentyp bereit.

Ein *Subscriber* in einem Node empfängt asynchron eine Nachricht mit einem definierten Datentyp.

Ein *Topic* stellt die Verbindungsstelle zwischen Publisher und Subscriber dar.

Ein *Service Server* stellt eine synchrone Kommunikationsschnittstelle dar, welche den Empfang und das Versenden der Daten quittiert.

Ein *Action Server* stellt eine asynchrone Kommunikationsschnittstelle dar, welche den Empfang und das Versenden der Daten quittiert.

In Abbildung 2.10 sind die Bausteine eines ROS-Programms aufgeführt. Das gezeigte Beispiel sendet von Node zu Node eine Nachricht über einen Topic. Die Nodes unten rechts und unten links können den Topic abonnieren und die vom Node gesendete Nachricht empfangen.

Der *Service Client* stellt eine Empfangsanforderung an den *Service Server*. Dieser antwortet dem *Service Client* mit einer Nachricht[16].

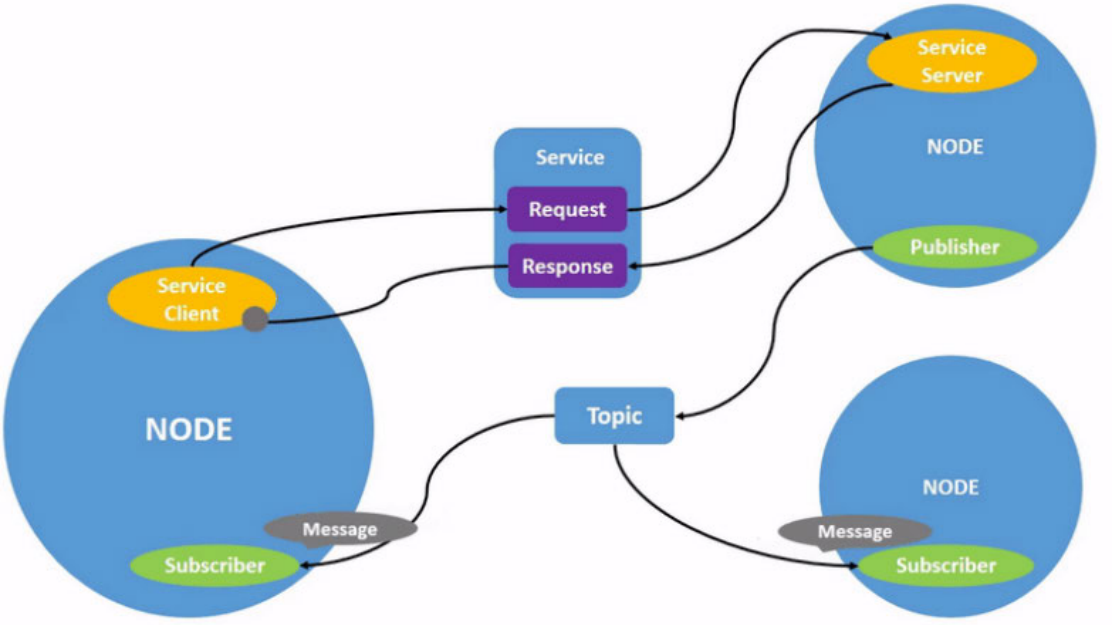

Abbildung 2.10: Schematischer Aufbau eines ROS2-Programms[16]

Für das Schreiben und Laden eines Nodes mit *Subscribern* und *Publishern* orientiert man sich an der Anleitung für ROS2 Foxy[19].

2.6.2 Simulationsumgebung Gazebo

Gazebo ist ein Tool für physikalische Simulation. Es wurde entwickelt, um Algorithmen, Regelungen oder Steuerungen eines Roboters zu testen, bevor sie in das reale System eingebettet werden können. So kann der Roboter bereits programmiert werden, obwohl die Hardware nicht vollständig aufgebaut ist. Zusätzlich können sicherheitskritische Bewegungen getestet werden, ohne eine Gefährdung darzustellen. In Gazebo können beispielsweise Gravitation, Wind und kinetische Parameter eingegeben werden, um möglichst realitätsnah zu simulieren. Das Tool bietet außerdem die Möglichkeit, Kollisionen mit Objekten zu testen. Die Eingabe der Simulationsparameter erfolgt in SI-Einheiten. Mithilfe einer Roboterbeschreibungsdatei, Unified Robot Description Format (URDF), oder einer Simulationsbeschreibungsdatei, Simulation Description Format (SDF), kann eine Simulationsumgebung aufgebaut werden. Dadurch wird es möglich, einen im Computer Aided Design (CAD) erstellten Roboter in eine virtuelle Umgebung einzubinden. Der entscheidene Vorteil bei der Nutzung von Gazebo ist, dass auf das Roboter-Hauptprogramm aufgesetzt werden kann und dieses direkt für die Simulation verwendet wird.

Für diese Arbeit wird Gazebo benutzt, um eine Steuerung für den Roboter zu entwickeln und zu testen. Der Playstation 4 Controller dient als Eingabemedium zum Testen der Fahrsteuerung in der Simulation. Zusätzlich wird der zu entwickelnde Algorithmus zur Vermeidung von Bodenschäden getestet. Zu berücksichtigen ist, dass Gazebo keine Bodenverdichtung und abrasives Verhalten des Untergrundes darstellen kann. Stattdessen kann der Kraftschlussbeiwert μ parametriert und das Lenkverhalten untersucht werden.

Bei der Erstellung eines Robotermodells wird mit geometrischen Körpern (*links*) und Verknüpfungselementen (*joints*) gearbeitet. Mit diesen wird angegeben, wie sich eine Komponente des Roboters zu einer anderen Komponente verhält. Das könnte zum Beispiel ein Roboterarm an einer Basis oder ein Rad an einem Chassis sein. Das grundlegende Funktionsprinzip besteht darin, dass jede Komponente ein eigenes Koordinatensystem besitzt, welche in Relation zueinander stehen.

Die Dateiformate SDF und URDF sind nach dem Extensible Markup Language (XML)-Standard aufgebaut. Der Codeauszug 2.1 zeigt den Aufbau eines *links*, also eines Volumenkörpers. Es könnte hier zum Beispiel das Grundgestell eines Roboters sein. In *tags* werden die visuellen (<*visual*>) und physikalischen Eigenschaften (<*inertial*>) festgesetzt.

Eine beispielhafte Implementierung eines *links* wird im Folgenden beschrieben:

```
<link name="base_link">
  <visual>
    <origin xyz="0 -${base_offset} 0" rpy="0 0 0"/>
    <geometry>
      <mesh filename="file://$(find package)/ />
    </geometry>
    <material name="yellow">
      <color rgba="0.8784 0.6627 0.6627 1.0"/>
    </material>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="${pi/2} ${pi/2} 0 "/>
    <geometry>
      <box size="${length} ${width} ${height}"/>
    </geometry>
  </collision>

  <inertial>
    <origin xyz="0 ${height} 0" rpy="${pi/2} 0 0"/>
    <mass value="${mass}"/>
    <inertia ixx="${(1/12) * mass * ((width * width) + (height *
        height))}" ixy="0.0" ixz="0.0" iyy="${(1/12) * mass * ((length
        * length) + (height * height))}" iyz="0.0" izz="${(1/12) * mass
         * ((length * length) + (width * width))}"/>
  </inertial>
</link>
```

Codeauszug 2.1: Beispielcode zur Erstellung eines *links*

Im *tag* <*visual*> werden visuelle Eigenschaften des Roboters gesetzt. Es kann sowohl ein einfaches 3D-Modell modelliert, als auch Standard Triangle Language (STL)- oder Digital Asset Exchange File (DAE)- Dateien geladen werden. Im *tag* <*collision*> werden äußere Eigenschaften des Roboters gesetzt. Diese sind nötig, um Kollisionen zu berechnen.

Hier ist es empfehlenswert, einfache geometrische Körper, zum Beispiel Quader, zu erstellen, da sich die Berechnungszeit der Simulation bei der Verwendung von komplexen Geometrien stark erhöht.

Im *tag* <*inertial*> werden die Massenträgheitsmomente des Roboters beschrieben. Dieser *tag* ist von großer Relevanz für die Reaktion von Kräften auf den Simulationskörper. Zeile

21 im Codeauszug 2.1 beschreibt die Massenträgheit einer erstellten Box. Codeauszug 2.2 zeigt die Erstellung eines Verknüpfungselementes (*joints*).

```
<joint name="steering_joint" type="revolute">
  <parent link="base_link"/>
  <child link="axis"/>
  <origin xyz="1 0 0" rpy="0 0 0"/>
</joint>
```

Codeauszug 2.2: Beispielcode zur Erstellung eines *joints*

Dieser Codeauschnitt beschreibt das Verknüpfungselement *steering_joint*. Es ist der link *axis* (Achse) mit dem link *base_link* (Grundgestell) über eine rotierenden Eigenschaft verknüpft. Das Keyword *origin* beschreibt die translatorische *xyz* und rotatorische *rpy* Ausrichtung zueinander.

Mit diesen Kenntnissen ist es nun möglich, sukzessiv ein Robotermodell aufzubauen.

Für die Erstellung einer Gazebo Simulationsumgebung sind mindestens zwei Nodes nötig. Ein Node, welcher Gazebo startet und die Simulationsumgebung lädt. Dazu gehört das Robotermodell selbst, aber auch die Umgebung. Einen zweiten Node, welcher Steuerbefehle an die Verknüpfungselemente sendet. Dieser bildet beispielsweise die Schnittstelle zwischen Simulation und einer Lenkung.

2.6.3 ROS2 Control

ROS2 Control ist ein weiteres Framework, das für die Steuerung und Regelung von Robotersystemen entwickelt wurde. Die Besonderheit des Frameworks liegt in der vergleichsweise schnellen Hardwareintegration, welche durch eine festgelegte Vorgehensweise möglich ist[18]. Ebenso kann Gazebo mit dem ROS2 Control Framework verknüpft werden.

Die zwei wichtigsten Komponenten aus dem Framework sind die Controller und der Controllermanager. Die Controller sind eine vorgefertigte Softwarekomponente für eine spezifische Anwendung. Tabelle 2.5 zeigt einen Auszug verfügbarer Controller. Der Controller *joint_state_broadcaster* gibt beispielsweise die Istposition eines Motors wieder. Der Controller *position_controllers* dagegen kann eine Solldrehzahl an einen Antrieb senden.

Der Controllermanager verwaltet Controller. Er prüft ihre Zustände und leitet die Lese- oder Schreibbefehle an die Hardware weiter.

Name	Beschreibung	Mögliche Verwendung
imu_sensor_broadcaster	Senden von Messwerten eines Beschleunigungssensors	Erfassen von Ist-Beschleunigungen des Roboters
joint_state_broadcaster	Senden von Ist-Zuständen der Roboter *joints*	Erfassen von Istdrehzahlen eines Motors oder einer Position
position_controllers	Senden von Sollpositionen	Ansteuern einer Sollposition eines Schrittmotors
velocity_controllers	Senden von Sollgeschwindigkeiten	Ansteuern eines Motors durch die Vorgabe einer Solldrehzahl

Tabelle 2.5: Übersicht einer Auswahl verfügbarer Controller[15] in ROS2

Das Laden der Controller und des Controllermanagers findet auf die gleiche Art statt, wie das Laden eines Nodes. Zusätzlich sind in der URDF-Datei des Roboters die *joints* mit dem Controller zu verlinken. Der Codeauszug 2.3 zeigt eine beispielhafte Implementierung in der URDF-Datei für die Nutzung des Roboters unter Gazebo. Innerhalb des Codes können Positionsgrenzen einer Achse (param name=min/max) eingegeben werden, welche die Achse nicht überschreiten.

```
1  <ros2_control name="GazeboSystem" type="system">
2  <hardware>
3  <plugin>gazebo_ros2_control/GazeboSystem</plugin>
4  </hardware>
5  <joint name="fl_wheel_joint">
6  <command_interface name="velocity">
7  <param name="min">-1.57</param>
8  <param name="max">1.57</param>
9  </command_interface>
10 <state_interface name="velocity"/>
11 <state_interface name="effort"/>
12 </joint>
```

Codeauszug 2.3: Beispielhafte Verlinkung eines Roboterjoints mit dem Controller velocity und effort

3 Methodik

Nachdem im vorherigen Kapitel die Grundlagen für die Bearbeitung dieser Arbeit gelegt wurden, findet in diesem Kapitel eine Übersicht der zu bearbeitenden Teilaufgaben. Abbildung 3.1 stellt visuell die zu erfüllenden Aufgaben dar. Die Bearbeitung erfolgt in zwei Hauptaufgaben; das Erstellen einer Simulation (siehe Kapitel 4) und das Programmieren der Roboterbasis (siehe Kapitel 5).

Die Simulation ermöglicht eine Überprüfung der Funktionsfähigkeit der Fahrsteuerung der Roboterbasis, noch vor der eigentlichen Integration in dieser. Dafür muss eine virtuelle Roboterbasis bereitgestellt werden, die in die Simulation geladen werden kann. Ein Fahralgorithmus der bodenschonend fährt, ist zusätzlich zu entwickeln und zu implementieren. Für die Simulation sind außerdem Umgebungsparameter festzusetzen. Als letzten Bearbeitungspunkt im Aufgabenpaket Simulation, wird die Fahrsteuerung simuliert.

Das zweite Aufgabenpaket, die Roboterbasis, beinhaltet die Implementierung von CANopen und die Entwicklung einer seriellen Schnittstelle. Ausgehend vom MicroGiant-Board werden über CANopen die Antriebssoll- und istwerte ausgetauscht. Dies stellt die Low-Level-Verarbeitung dar. Die serielle Schnittstelle soll die Kommunikation zwischen dem MicroGiant-Board und dem Jetson ermöglichen. Die Inbetriebnahme und Bewertung der Motoren bildet den Abschluss dieser Aufgabe.

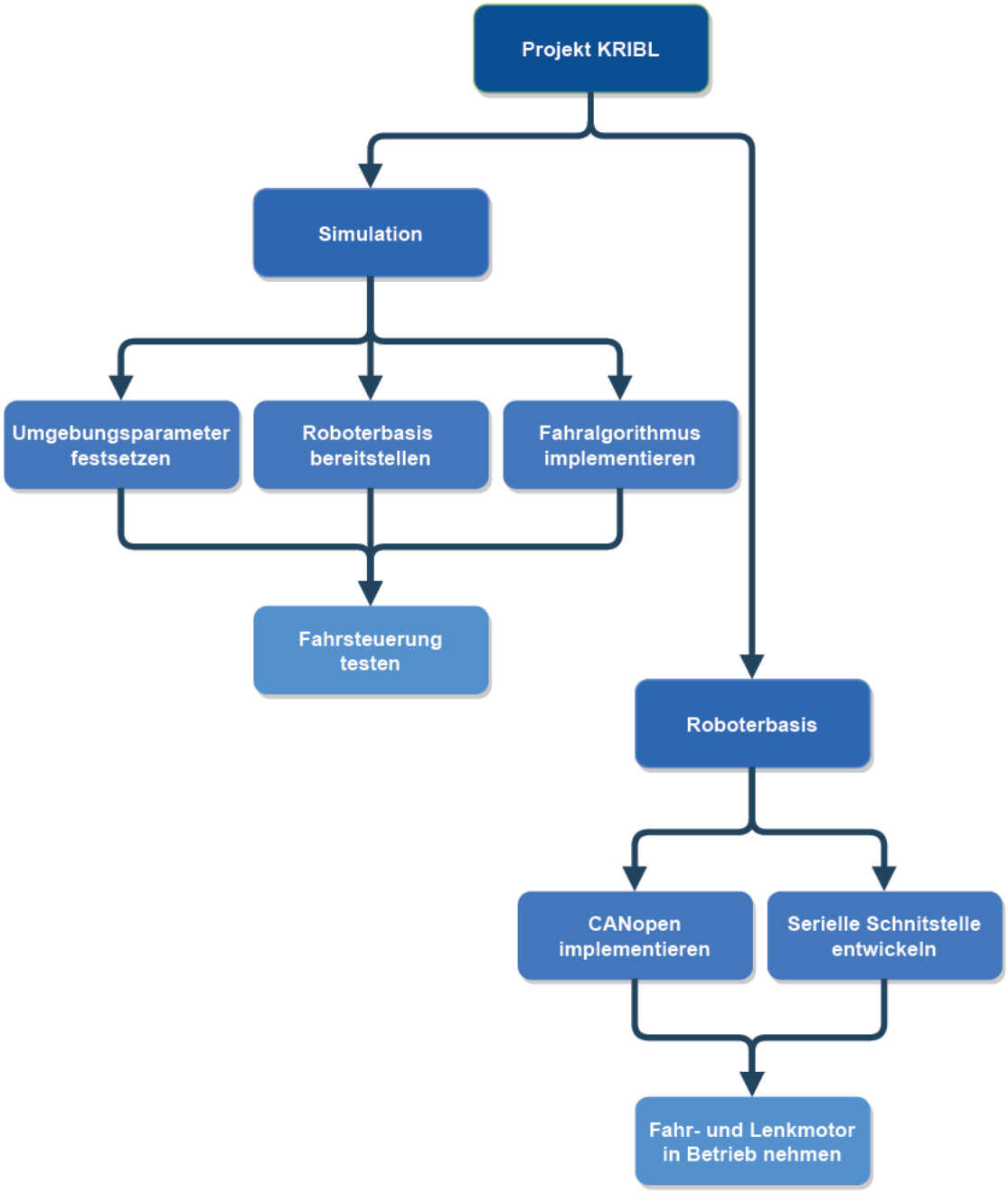

Abbildung 3.1: Übersicht der zu realisierenden Aufgaben

4 Simulationsumgebung zur Entwicklung der Fahrsteuerung

Für die Entwicklung der Fahrsteuerung wird eine Simulationsumgebung mit Gazebo aufgebaut. Für die Erstellung der Simulationsumgebung wird eine Roboterbeschreibungsdatei (URDF-Datei), eine konfigurierte Welt, CAD-Modelle der Komponenten und eine *launch*-Datei die alles öffnet, nötig. Abbildung 4.1 zeigt in welchen Abhängigkeiten die einzelnen Strukturen zueinander stehen.
Die Datei *robot_ description* stellt die Basis der Simulation dar. In dieser wird beschrieben aus welchen *links* der Roboter besteht und wie sie aussehen. Die Optik ist durch CAD-Dateien beschrieben. Eine Verknüpfung zwischen der *robot_ description* und der Fahrsteuerung *robot_ control* ist nötig, um den Roboter in der Simulation zu fahren. Weiterhin muss ein Untergrund geladen werden, worauf der Roboter fährt. Dafür wird eine *world*-Datei erstellt, mit einem Untergrund der einem Ackerboden ähnelt. Die in Abbildung 4.1 gezeigte *launch*-Datei wird die Simulationsumgebung und die konfigurierte Welt laden.

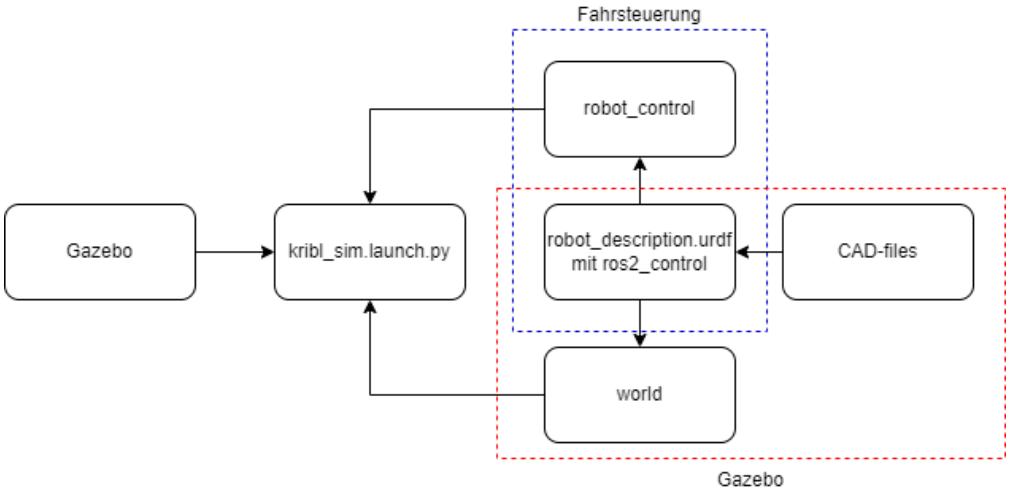

Abbildung 4.1: Struktur zum Aufbau der Gazebo Simulationsumgebung

Das Erstellen der *robot_ description* wird in Abschnitt 4.1 erläutert. Die Implementierung und Entwicklung der Fahrsteuerung *robot_ control* findet in Abschnitt 4.2 statt.

4.1 Bereitstellen der Roboterbasis in Gazebo

Für die Erstellung der Gazebo Simulation wurde das Projekt von *Robotmania*[22] als Vorlage für die Fahrsteuerung verwendet. Um in Gazebo das Roboterbasismodell zu laden, muss die URDF-Datei erst erstellt werden. Für die Erstellung der URDF-Datei werden zunächst Macros für die *links* erstellt. Im Macro für den *base_ link* wird das in Abbildung 4.2 gezeigte Modell geladen. Die Steuerachse besteht aus einem Zylinder, der Radaufnahme und dem Rad selbst. Diese sind in Abbildung 4.2 zu erkennen. Das Keyword *$side* beschreibt die jeweilige Seite des Roboters.

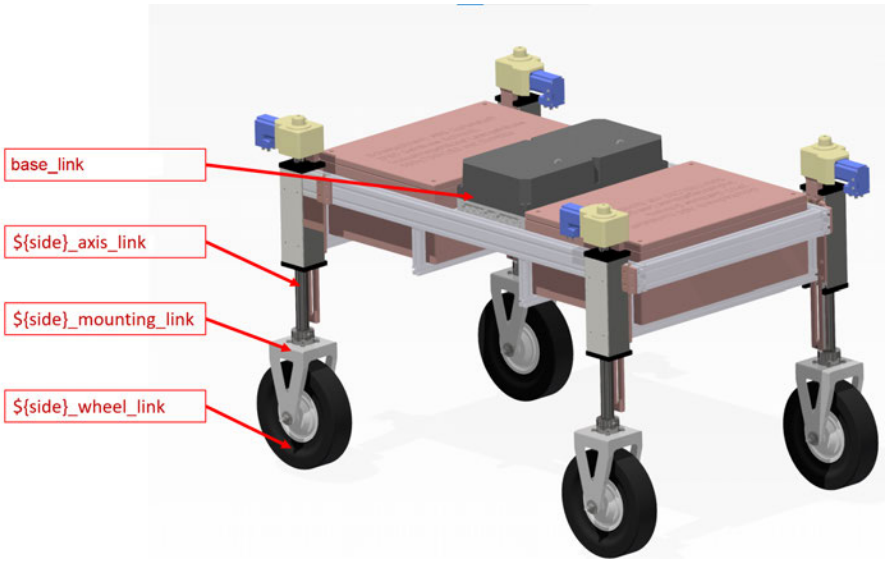

Abbildung 4.2: Darstellung der zu erstellenden *links*

Anschließend werden die Abhängigkeiten der Geometrien zueinander mithilfe der *joints* festgelegt. Für die Darstellung der Abhängigkeiten dient folgende Übersicht:

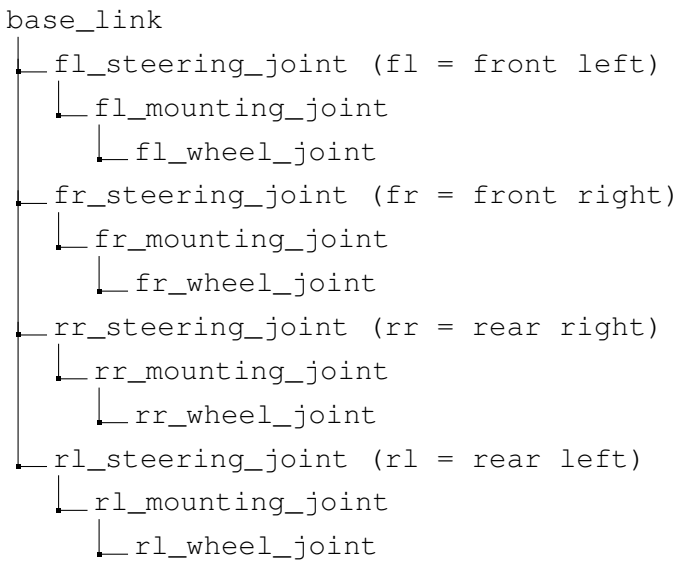

Abbildung 4.3: Abhängigkeiten der *links* zueinander

Der *base_link* bildet die Basis. An ihr sind alle *steering_joints* verknüpft. An jedem *steering_joint* sind die *mounting_joints* und an jedem *mounting_joint* die *wheel_joints* verknüpft.
Da die Zuweisung der Achsen, Radaufnahmen und Räder sich stetig wiederholt, wird das Tool XML Macros (xacro) verwendet. Mit diesem Tool kann eine allgemeingültige Struktur erstellt werden, welche durch die Zuweisung von Variablen spezifisch wird.

Zum Beispiel kann der in Unterabschnitt 2.6.2 beschriebene Code um ein xacro-tag erweitert werden. Es ergibt sich Codeauszug 4.1.

```
1  <xacro:macro name="wheel" params="side radius width pos_x pos_y
       pos_z mass">
2  <collision>
3  <origin xyz="0 0 0" rpy="0 ${pi/2} 0 "/>
4  <geometry>
5    <cylinder radius="${radius}" length="${width}"/>
6  </geometry>
7 </collision>
8  </xacro:macro>
```

Codeauszug 4.1: Erstellung eines Macros durch die Nutzung von xacro

Dieser Code lässt sich in der Roboter-URDF-Datei mehrfach ausführen und somit auf weitere Achsen übertragen. Eine Änderung von Parametern innerhalb des Macros wirkt sich auf alle zu erstellen Komponenten aus. Es wird für jede der genannten Komponenten ein Macro geschrieben. Codeauszug 4.2 zeigt die Implementierung im Code des Macros in der URDF-Datei.

```
1 <xacro:axis side="fl" radius="${axis_radius}" width="${axis_width}"
    pos_x="${axis_pos_x}" pos_y="${axis_pos_y}" pos_z="${axis_pos_z}
    " mass="${axis_mass}" />
2
3 <xacro:axis side="fr" radius="${axis_radius}" width="${axis_width}"
    pos_x="-${axis_pos_x}" pos_y="${axis_pos_y}" pos_z="${axis_pos_z}
    " mass="${axis_mass}" />
```

Codeauszug 4.2: Beispielhafte Implementierung zur Erstellung zweier Achsen mithilfe von xacro

Im Anhang unter Abbildung A.9 sind alle Nodes dargestellt, die bei der Simulation verwendet werden. Die in der Simulation geladene Roboterbasis ist in Abbildung 4.4 zu erkennen.

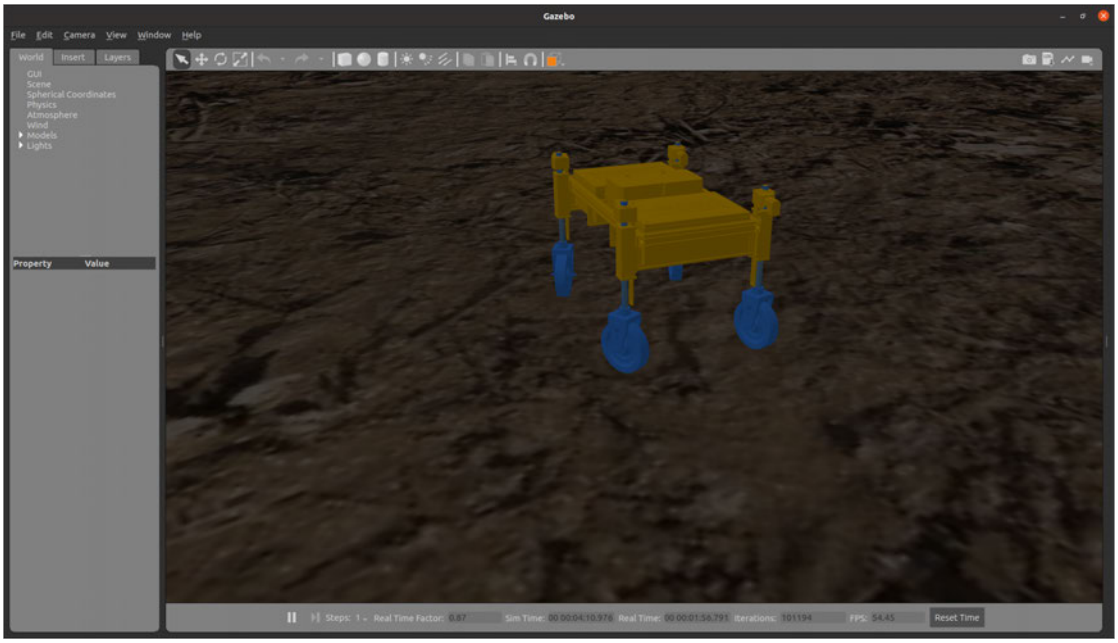

Abbildung 4.4: Gazebo Simulationsumgebung mit Basisroboter

4.2 Implementierung eines Fahralgorithmus

Wie im Abschnitt 2.3 beschrieben, erzeugt ein großer Lenkwinkel einen großen Schlupf. Sobald der gewünschte Lenkwinkel erreicht ist und keine Lenkwinkeländerung auftritt, verringert sich der Querschlupf. Die Bodenbeschädigungen finden nun ausschließlich durch den Antriebsschlupf statt. Daher gilt es, die Lenkwinkeländerung stets gering zu halten. Dies kann mit zwei Methoden erreicht werden:

1. Eine Fahrmethode wählen, welche einen geringen Kurvenradius benötigt.
   Damit wird der Lenkwinkel β verringert.

2. Begrenzung des zulässigen Querschlupfes.
   Dadurch ergibt sich ein maximal zulässiger Winkel.

Auf diese Methoden wird im Folgenden genauer eingegangen.

4.2.1 Auswahl der Lenkmethode

Wie im Abschnitt 2.5 beschrieben, kann durch den Einsatz einer Allradlenkung ein kleinerer Kurvenradius bei gleichem Lenkwinkel im Vergleich zur Ackermannlenkung realisiert werden. Oder anders ausgedrückt: Mit der Allradlenkung ist ein kleinerer Lenkwinkel nötig, um den gleichen Kurvenradius zu fahren. Aus diesem Grund wird die Allradlenkung implementiert.

Die Raddrehzahl n, mit der Einheit $\frac{rad}{s}$ für die Fahrsteuerung, und der Lenkwinkel β werden wie folgt berechnet[22]:

$$n = \frac{\sqrt{(vel_x \pm (\frac{\omega \cdot t}{2}))^2 + \frac{\omega \cdot L}{2}^2} \pm \omega \cdot s}{r} \tag{4.1}$$

Das Plus in der Gleichung steht für eine Fahrzeugseite, das Minus für die andere. Die Solldrehrate ω mit der Einheit $\frac{rad}{s}$ ist je nach Wunschrichtung mit einem Minus für links oder einem Plus für rechts behaftet.

$$\beta = \arctan(\frac{\omega \cdot L}{2 \cdot vel_x \pm \omega \cdot t})\,[rad] \tag{4.2}$$

vel_x	Sollgeschwindigkeit in X-Richtung $[\frac{m}{s}]$
ω	Solldrehrate aus Sicht des Fahrzeugmittelpunkts $[\frac{rad}{s}]$
t	Spurweite [m]
L	Radstand [m]
s	Lenkrollradius [m]
r	Radradius [m]

4.2.2 Begrenzung des Querschlupfes

Der Lenkwinkel β wird aus dem Winkel der Fahrzeugrichtung und dem Lenkwinkeleinschlag gebildet. Bei Erreichen der Sollrichtung ändert sich der Lenkwinkeleinschlag deshalb zu 0° und eine Beschädigung des Bodens wird nur vom Antriebsschlupf verursacht. Bei jeder Lenkwinkeländerung wird ein Querschlupf erzeugt, welcher Schäden am Untergrund verursacht. Nun kann betrachtet werden, dass ein langsames Rad bei konstanter Lenkwinkeländerung ein höheres Maß an Verdrängung der Erde erzeugt als ein schnell laufendes Rad. Dies lässt sich dadurch erklären, da der Reifenlatsch (Reifenaufstandsfläche) länger auf einer Stelle verweilt und der Untergrund an dieser Fläche abgeschert wird. Aus diesem Grund wird ein Algorithmus entwickelt, welcher die Reifenumfangsgeschwindigkeit und die Lenkwinkeländerung in Zusammenhang bringt.

Zunächst wird der maximale Lenkwinkel ermittelt. Aus dem Abschnitt 2.3 geht hervor, dass der optimale Schlupf oder auch der kritische Schlupf auf normalen Straßenverhältnissen zwischen $0,1$ und $0,3$ liegt. Jedoch sollte der Schlupf auf Ackerboden möglichst gering gehalten werden. Daher wird er für diese Arbeit zu $S_\beta = 0,1$ angenommen. Der maximale Lenkwinkel eines Rades β_{max} ergibt sich aus der Umstellung der Gleichung 2.5.

$$\beta_{max} = \arctan\left(S_\beta\right) \tag{4.3}$$

$$\beta_{max} = \arctan\left(0,1\right) = 0,1\,rad \mathrel{\widehat{=}} 5,7\,^\circ \tag{4.4}$$

Weiter muss ermittelt werden, aus wieviel Reifenlatsche eine Radumfangsfläche besteht. Dadurch erhält man die Anzahl an zulässigen Lenkbewegungen pro Radumfang.

Der Reifenlatsch A_T lässt sich vereinfacht durch die Gewichtskraft F_z und den Reifendruck p_t ermitteln[4]. Weitere Effekte wie die Luftkompression, die Strukturkomponente des Gummis und die Rundhaltekraft, werden für die vereinfachte Annahme nicht beachtet. Ebenso wird der Reifen ohne Stollen angenommen.

$$A_T = \frac{F_z}{p_t} \tag{4.5}$$

Durch die Angabe der Reifenbreite b_R lässt sich die Sehnenlänge l eines Rades bestimmen. Die Sehnenlänge stellt die Seitenlänge des Reifenlatsches dar. Abbildung 4.5 verdeutlicht die gesuchte Seitenlänge.

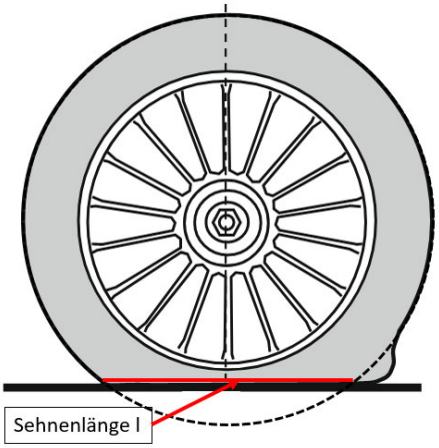

Abbildung 4.5: Darstellung der Sehnenlänge eines Rads ausgelöst durch eine Radlast[6]

$$l = \frac{A_T}{b_R} \tag{4.6}$$

Folgend wird bestimmt, wie oft sich die Sehnenlänge in einer Radumdrehung befindet. Dazu wird der Radumfang U ermittelt.

$$U = 2\pi r \tag{4.7}$$

Zusammen mit dem Radumfang U und der Sehnenlänge l erhält man einen Faktor γ, der zeigt, wie oft eine Lenkwinkeländerung pro Radumdrehung durchgeführt werden kann, um bodenschonend zu lenken.

$$\gamma = \frac{l}{U} \tag{4.8}$$

Um für die Steuerung einen zeitlichen Abstand zu ermitteln, ab welchem Zeitpunkt eine Lenkwinkeländerung erlaubt ist, kann über die Raddrehzahl n ein zeitlicher Abstand t_L ermittelt werden.

$$t_L = \frac{\gamma}{n} \tag{4.9}$$

Für eine gute Manövrierfähigkeit wird der maximale Lenkwinkel aus Gleichung 4.3 für jede Lenkwinkeländerung zugelassen. Zusätzlich kann die Bodeneinsinktiefe h beachtet werden. Dieser Parameter ist auf dem Feld schwer zu bestimmen, da sich der Ackerboden sehr variabel verhält. Die Sehnenlänge l wird durch die folgende Formel berechnet[8].

$$l_{Neu} = 2\sqrt{h \cdot (2 \cdot r - h)} \tag{4.10}$$

Da die Einsinktiefe einen größeren Einfluss auf die Sehnenlänge l hat als ein normal belastetes Rad im Stand, wird die Gleichung 4.10 verwendet. Zusammengesetzt kann nun folgende Formel in den Code implementiert werden:

$$t_L = \frac{2\sqrt{h \cdot (2 \cdot r - h)}}{n \cdot U} \tag{4.11}$$

Für die Lenkvorgänge mit einem maximalen Lenkwinkel ist es nötig, eine neue zulässige Drehrate der Roboterbasis zu bestimmen. Dafür wird Gleichung 4.2 nach ω umgestellt:

$$\omega_{max} = \frac{tan(\beta) \cdot 2 \cdot vel_x}{L \pm tan(\alpha) \cdot t} \tag{4.12}$$

Diese Formel wird bei der Berechnung der neuen Sollpositionen verwendet.

4.2.3 Implementierung

Nachdem zuvor die Grundlage für den Algorithmus gesetzt wurden, erfolgt in diesem Kapitel die Implementierung. In Abbildung 4.6 wird der implementierte Algorithmus dargestellt. Der Code ist in der *robot_ control.py* zu finden.
Im ersten Schritt wird die Sollposition der Lenkmotoren mit Gleichung 4.1 berechnet. Es wird die Differenz aus einer alten und der neuen Sollposition ermittelt. Ist die Differenz kleiner als der erlaubte Lenkwinkel, wird bereits so gesteuert, dass der zulässige Querschlupf nicht überschritten wird. Andernfalls wird der Zeitfaktor aus Gleichung 4.11 neu kalkuliert.
Wenn der Zeitfaktor unterschritten wird und der „alte“ Reifenlatsch noch Kontakt zur Erde hat, wird ausschließlich die alte Sollposition verwendet und es findet keine Lenkwinkeländerung statt. Bei Überschreiten der Zeit wird ermittelt, in welchem Lenkstatus die Roboterbasis steht.
In Abhängigkeit zur aktuellen Radstellung und der neuen Wunschposition wird die neue Sollposition berechnet. Abschließend wird die neue Sollposition für den nächsten Zyklus als alte Position zwischengespeichert. Vereinfacht lässt sich sagen, dass eine Lenkwinkeländerung immer dann statt findet, wenn die Zeit aus dem Zeitfaktor abgelaufen ist. Das stellt sicher, bei jeder Lenkwinkeländerung eine neue Reifenaufstandsfläche zu verwenden.

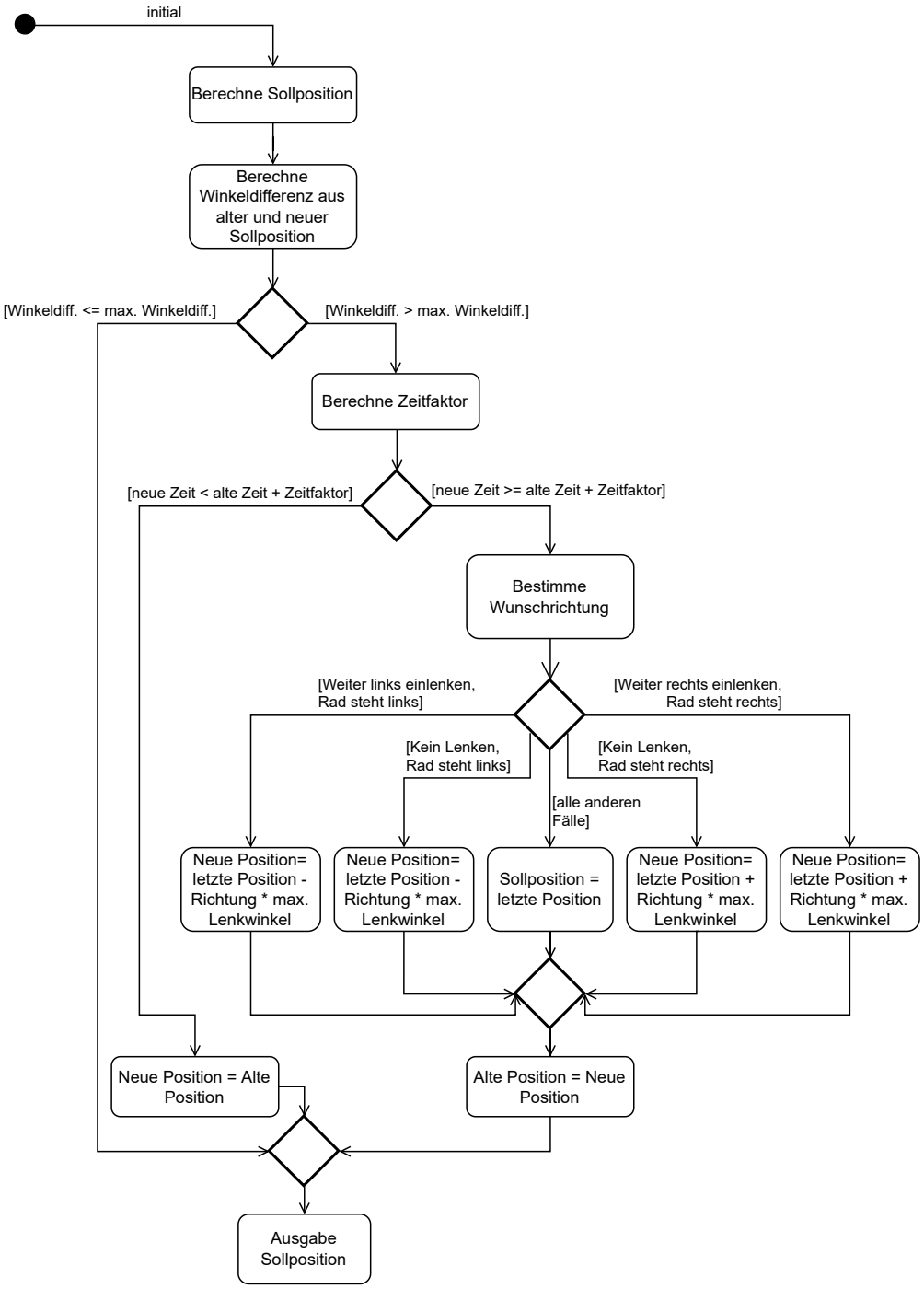

Abbildung 4.6: Schematische Darstellung des Fahralgorithmus zur Bodenschonung

4.3 Festsetzen von Umgebungsparametern

Für die Erstellung der Simulation sind Parameter festzusetzen. Neben geometrischen Parametern für das Robotermodell werden essentielle Parameter der Simulationsumgebung gesetzt. Die gewählten Umgebungs- und Algorithmusparameter sind in Tabelle 4.1 einsehbar. Die geometrischen Größen, welche aus den technischen Zeichnungen hervorgehen, sind im Anhang unter Tabelle A.1 zu finden. Die Einsinktiefe h des Reifens ist aufgrund der Vielzahl an Bodenparameter schwer absehbar. Um eine gute Wirkung des Algorithmus zu erzeugen, wird die Einsinktiefe so gewählt, dass eine größere Sehnenlänge l entsteht als wenn die Sehnenlänge durch den Reifenlatsch und -breite berechnet werden würde. Sie wird daher auf 0,03 m gesetzt.
Der zulässige Schlupf wird wie in Abschnitt 2.3 erläutert zu 0,1 gesetzt. Die Simulation wird mit dem Kraftbeischlusswert 0,6 für Erde und 0,75 für asphaltierten Belag getestet. Die Geschwindigkeiten für

- die Arbeitsgeschwindigkeit ist auf $1,5\,\frac{m}{s}$ und
- die Arbeitsrotationsgeschwindigkeit ist auf $1,5\,\frac{rad}{s}$

festgesetzt. Für die Fahrantriebe wird zusätzlich eine zulässige maximal Drehzahl bestimmt. Diese muss für die Simulation, von der im Abschnitt 2.3 erwähnten Fahrgeschwindigkeit $2,5\frac{m}{s}$ auf $\frac{rad}{s}$ umgerechnet werden. Mit folgender Formel wird dazu die maximale Drehzahl n_{max} berechnet.

$$n_{max} = \frac{v_{max} \cdot 2\pi}{2\pi \cdot r} \tag{4.13}$$

$$n_{max} = \frac{2,5\,\frac{m}{s} \cdot \cancel{2\pi}}{\cancel{2\pi} \cdot 0,167\,m} = 14,97\,\frac{rad}{s} \tag{4.14}$$

Aus dem Datenblatt der Lenkmotoren[13] geht eine maximale Drehzahl n_{Lmax} von $200\,\frac{1}{min}$ hervor. Zusätzlich ist ein Getriebe mit einer Übersetzung von $i = 25$ verbaut. Für die maximale Winkelgeschwindigkeit der Lenkmotoren an der Welle n_{Wmax} wird berechnet:

$$n_{Wmax} = \frac{n_{Lmax} \cdot 2\pi}{i \cdot 60} \tag{4.15}$$

$$n_{Wmax} = \frac{200\,\frac{1}{min} \cdot 2\pi}{25 \cdot 60} = 0,84\,\frac{rad}{s} \tag{4.16}$$

Parameter	Wert
Einsinktiefe h	0,03 m
Zulässiger Schlupf	0,1
Kraftbeischlusswert μ_h	0,6/0,75
Max. Geschwindigkeit Roboter v	1,5 $\frac{m}{s}$
Max. Winkelgeschwindigkeit Roboter ω	1,5 $\frac{rad}{s}$
Max. Drehzahl n	$14,97\frac{rad}{s}$
Max. Winkelgeschwindigkeit Lenkmotor n_{Wmax}	$0,84\frac{rad}{s}$

Tabelle 4.1: Umgebungsparameter für die Simulation

4.4 Übersicht der ROS2-Programmfeatures

Dieser Abschnitt soll einen Überblick über die Funktionen und die Dateistrukturen des programmierten ROS2-Programms geben. Für diese Arbeit wurde das ROS2-Paket *kribl_4ws* mit weiteren Unterordnern erstellt. Diese beinhalten alle implementierten Funktionen sowie die Simulation dieser Arbeit.
Die Ordnerstruktur des ROS2-Arbeitsverzeichnisses ist im Anhang unter Abbildung A.1 zu finden. Im Ordner *description* sind alle Dateien zu finden, die den Roboter beschreiben. In der Roboterbeschreibungs *robot.urdf.xacro* werden alle *.xacros* zusammen geführt.

Im Ordner *launch* befinden sich die launch-Dateien. Die Simulationsumgebung kann mit *kribl_sim.launch.py* gestartet werden. Eine definierte Trajektorie wird mit *robot_drv_start.launch.py* verfahren. Die Roboterbasis selbst wird mit *kribl_sim.launch.py* initialisiert.

Der Ordner *modells* beinhaltet die .stl-Dateien für die äußere Erscheinung in der Simulationsumgebung.

Unter *scripts* befindet sich der Code für die Fahrsteuerung mit bodenschonenden Algorithmus (*robot_control.py*), die Trajektorie (*uart_drv_start.py*) und die serielle Kommunikation (*uart.py*).
Die Simulationsumgebung ist im Ordner *worlds* zu finden.

5 Implementierung und Inbetriebnahme des Basis-Robotersystems

5.1 Aufbau der Hardware

In Abbildung 5.1 wird der schematische Aufbau des Basis-Roboters dargestellt. Als zentrale Steuerungs- und Regelungskomponenten dieser Arbeit sind der Jetson, sowie das MicroGiant-Board, zu erwähnen. Mit dem Jetson wird der Soll-Geschwindigkeitsvektor errechnet. Die Sollwerte werden vom Jetson über eine Schnittstelle an das MicroGiant-Board weitergegeben. Das Board übersetzt die Sollwerte in eine Solldrehzahl für die Nabenmotoren und eine Sollposition für die Lenkmotoren. Die Kommunikation zwischen MicroGiant-Board und den Antrieben findet über das CANopen-Protokoll statt.

Die Nutzung von CANopen ergab sich durch den Einsatz des MicroGiant-Boards und der Echtzeitfähigkeit des Bussystems. Um eine Debugger-Möglichkeit während des Fahrbetriebes zu erhalten, wird ein Access Point eingerichtet. Dieser baut ein WLAN auf. Der Jetson wird über einen Switch an einen Router angeschlossen. Somit ist ein Remote-Zugang möglich.
Zusätzlich ist ein Intel Nuc Computer eingebaut um Daten oder weitere Berechnungen für KI-Prozesse auslagern zu können.
Für den ersten prototypischen Aufbau des Basis-Roboters werden die Sollgrößen über einen am Jetson verbundenen PlayStation 4-Controller erzeugt. Die Weiterleitung der Daten findet über Bluetooth statt.

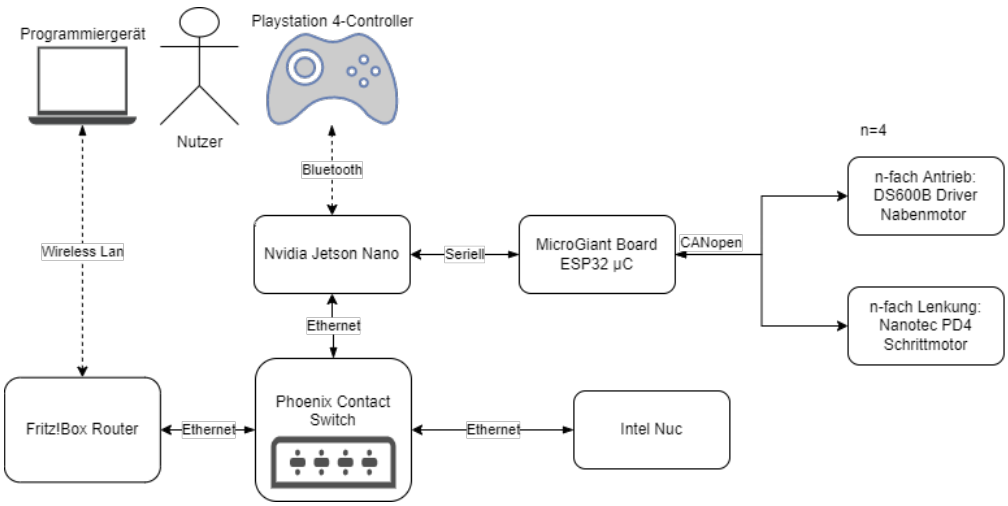

Abbildung 5.1: Schematischer Aufbau der verwendeten Hardware

5.2 CANopen

In diesem Kapitel wird auf die Implementierung von CANopen auf dem MicroGiant-Board eingangen. Zusätzlich wird die Inbetriebnahme der Motoren erläutert. Die Lenkmotoren sind werksseitig bereits mit PDO-Mappings ausgestattet. Bei den Nabenmotoren muss das PDO-Mapping noch durchgeführt werden. Das PDO-Mapping ist nötig, um

- den Control Mode (Motor soll im Geschwindigkeits- oder Positionierbetrieb fahren)
- die Geschwindigkeit oder die Positionierung und
- den Zustand »ein« oder »aus«

zu setzen. Darüber hinaus um

- die aktuelle Geschwindigkeit und Positionierung sowie
- den Zustand des Motors

auszulesen.

Für das Auslesen aller CAN-Nachrichten aus dem Telegramm wird ein USB zu CAN Converter der Firma PeakTech verwendet. Mit diesem und einer Auslesesoftware[1] ist es möglich, alle Nachrichten, die Bus-Teilnehmer senden und empfangen, zu lesen. Außerdem können CAN-Nachrichten händisch versendet werden.

[1] https://www.peak-system.com/PCAN-View.242.0.html, letzter Zugriff: 10.04.2023

5.2.1 Physikalischer Aufbau

Die verwendete CANopen-Topologie wird in Abbildung 5.2 gezeigt. Besonderes Augenmerk liegt auf der Verwendung der Abschlusswiderstände, um die Funktionalität des CAN-Busses zu gewährleisten.

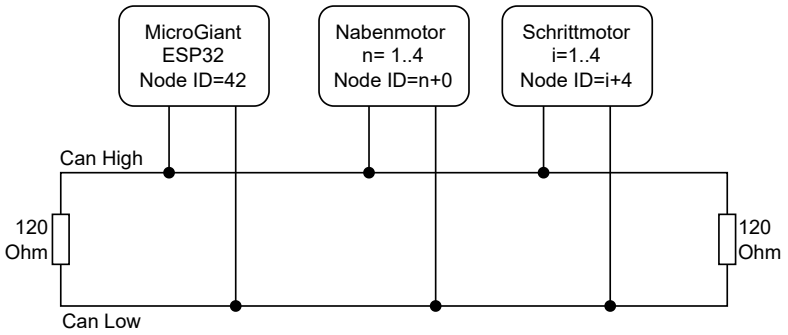

Abbildung 5.2: Can-Bus Topologie der Roboterbasis

Tabelle 5.1 stellt tabellarisch dar, welche konkrete Node-ID die Bus-Teilnehmer erhalten. Wichtig bei der Vergabe der Node-ID ist, dass jeder Bus-Teilnehmer eine einmalige ID erhält.

Einbauposition	Node-ID Lenkmotor	Node-ID Nabenmotor
Vorne Links	1	5
Vorne Rechts	2	6
Hinten Rechts	3	7
Hinten Links	4	8

Tabelle 5.1: Zuweisung der Node-IDs an die jeweiligen Antriebe

5.2.2 Implementierung der Anwendungsschicht

Für die Implementierung der Anwendungsschicht des CANopen-Protokolls wurde auf ein bereits bestehendes Projekt zurückgegriffen. Es handelt sich hierbei um das Projekt von A. Miller, welcher bereits CANopenNode für den ESP32 portiert hat[2].

Jedoch wurde dieses Projekt mit einem alten Softwarestand von CANopenNode und der Herstellerbibliothek *esp-idf*[3] erstellt.

Die Herausforderung liegt darin, das Projekt auf einen neuen Softwarestand zu heben. Aus der Dokumentation von CANopenNode geht hervor, dass für die Portierung die Dateien

- CO_driver.h
- CO_driver.c
- CO_driver_target.h

für den jeweiligen Mikrocontroller angepasst werden müssen.
Im ersten Schritt wurde die Vorlage aus dem CANopenNode Projekt verwendet. Die Dateien *CO_ driver.h*, *CO_ driver.c* und *CO_ driver_ target.h* wurden nun mit dem Projekt von A. Miller verglichen und Methoden sowie Datentypen in die Vorlage übernommen. Die Übernahme von Funktionen wurde mithilfe der Integrated Development Environment (IDE) *Visual Studio Code* bewerkstelligt. Dabei werden sukzessiv Fehler beseitigt, bis ein lauffähiges Programm entsteht.
Abbildung 5.3 beschreibt den Ablauf und die Nutzung der wichtigsten Methoden, um ein lauffähiges Programm zu erhalten. Die Abbildung stellt die Basiskonfiguration des MicroGiant-Boards dar. Methoden mit *CO_* sind für den CANopen-Stack zuständig. Alle weiteren Methoden wurden für diese Arbeit erstellt und werden in den nächsten Kapiteln näher beschrieben.

[2] https://github.com/nathanRamaNoodles/CANopen-ESP32-nodes/tree/master/Forks/Alexander_-Miller, letzter Zugriff: 26.02.2023
[3] https://docs.espressif.com/projects/esp-idf/en/latest/esp32/index.html, letzter Zugriff: 23.03.2023

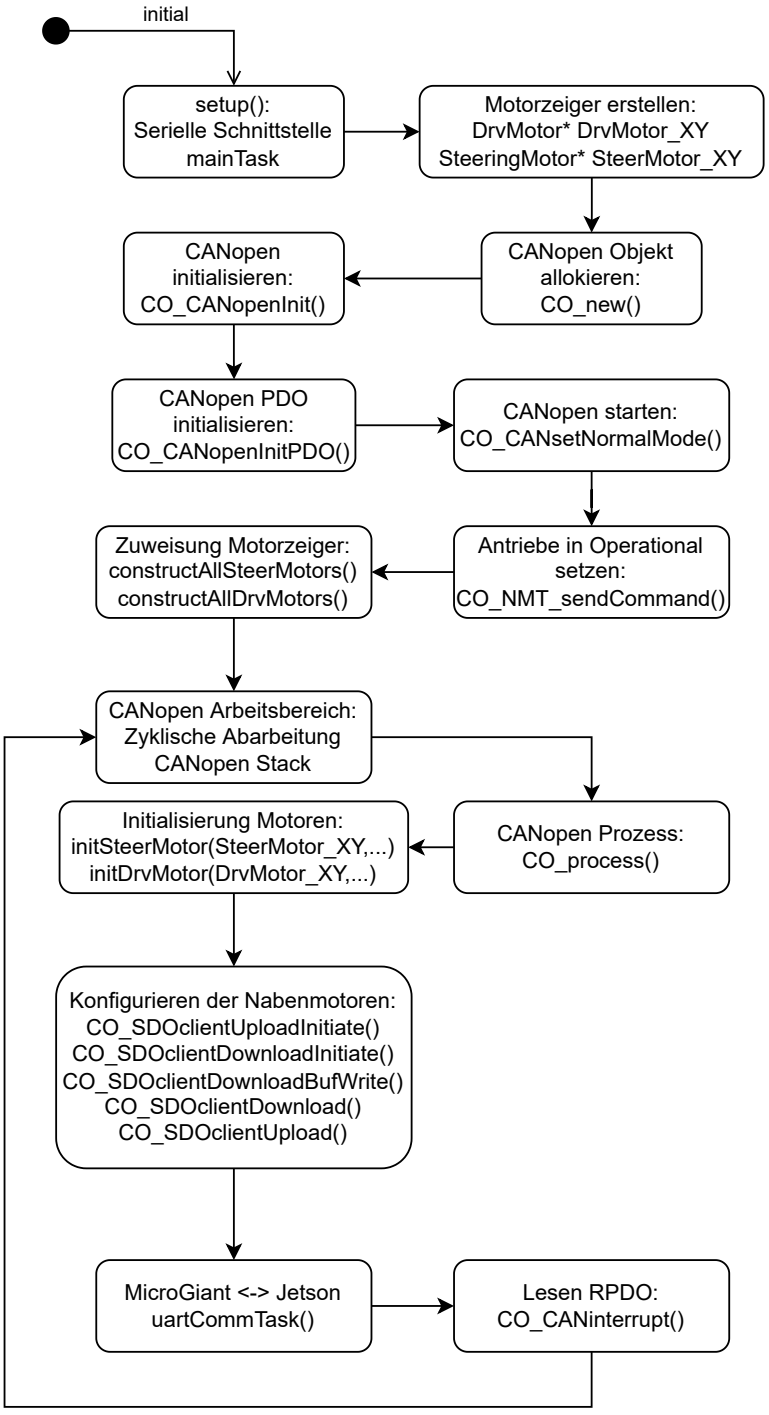

Abbildung 5.3: UML-Ablaufdiagramm der wichtigsten Funktionen des MicroGiant-Boards Hauptprogramms

Für die Low-Level-Implementierung wurden, wie in der objektorientierten Programmierung üblich, klassenähnliche Konstrukte erstellt. Allerdings ist die verwendete Sprache auf dem Mikrocontroller C. Die Programmiersprache C ist eine prozeduale Sprache. Sie ist prinzipiell nicht dafür gedacht klassenähnliche Strukturen aufzubauen. Durch Umstrukturierung ist es trotzdem möglich objektorientiert zu programmieren. Für diesen Anwendungsfall bietet es den Vorteil, dass jeder Motor eine eigene Sinneinheit bildet und mit seinen Variablen und Methoden separat verwaltet werden kann. Das Programmieren wird erleichtert und Funktionserweiterungen sind einfacher zu implementieren.

Die erstellten Klassen können in Abbildung A.2 und Abbildung A.3 eingesehen werden.

Beide Motorklassen haben Methoden, um die Register eines Motors aus dem Objectdictionary einem spezifischen Motor zuzuordnen (*constructMotor*). Mithilfe von *constructAllSteeringMotors* und *constructAllDrvMotors* werden alle Motoren den erstellen Registern zugeordnet.
Die Initialisierung erfolgt mit *initSteerMotor* und *initDrvMotor*. Mit der Initialisierungs-Methoden wird die Konfiguration der Register des Nabenmotors durchgeführt. Die Konfiguration wird in Unterabschnitt 5.2.3 erläutert.
Getter- und Setter-Methoden ermöglichen den Zugriff die Motordaten. Dazu zählen *getActualSteeringAngle* und *getDrvVelo*, womit die aktuelle Radposition oder Drehzahl eines Antriebes ausgelesen werden.
Zu den Setter-Methoden gehören die Methoden *setSteerTargetPos* und *setDrvVelo*, womit die Sollposition und die Solldrehzahl gesetzt werden.

5.2.3 Konfiguration des PDO-Mappings des Nabenmotors

Um die Nabenmotoren über PDO betreiben zu können, müssen diese mithilfe des SDO-Protokolls konfiguriert werden. Wie in Abschnitt 2.2 beschrieben, besteht das SDO-Protokoll aus einem Adressbereich und einem Datenbereich. Mithilfe der PCAN-View Software wird das Konfigurationstelegramm versendet. Aus dem Handbuch des Treibers für den Nabenmotor geht hervor, welche Register für die im Kapitelanfang erwähnten Funktionen nötig sind.

Es soll nun ein Beispiel folgen, wie ein PDO-Register konfiguriert wird. Für dieses Beispiel wird das Register 0x2352 ControlMode dem PDO-Register 0x1600 zugewiesen. Dafür muss als erstes das Register 0x1600 empfangsbereit gemacht werden:

Register	Bezeichnung	Zweck	Einheit	Datentyp
0x2352	ControlMode	Betriebsart des Motors hier: Geschwindigkeitssteuerung	1	UINT8
0x2020	VelocityRef	Setzen der Sollgeschwindigkeit	0,0001 $\frac{U}{min}$	INT32
0x20BB	MotorOnline	Einschalten des Motors	1	UINT8
0x2021	VelocityFdb	Lesen der Istgeschwindigkeit	0,0001 $\frac{U}{min}$	INT32
0x2013	MotorStatus	Zustand des Motors auslesen	1	UINT8

Tabelle 5.2: Register, welche ausgelesen werden sollen

Position im Telegramm	B0	B1	B2	B3	B4	B5	B6	B7
Wertigkeit		LB	HB		LB	HB	LB	HB
Wert	0x40	0x00	0x16	0x00	0x00	0x00	0x00	0x00

Tabelle 5.3: Aufbau eines Upload-Requests über SDO. B=Byte, LB=LowByte, HB=HighByte

Es ist besonders auf die Positionen des LowBytes und HighBytes im Telegramm zu achten. Diese entsprichen nicht der eigentlichen Lese-Richtung. Siehe dazu folgende Tabelle. Das Register 0x2352 erscheint in der Telegramm Reihenfolge als 52 23.

Nun kann das Register 0x2352 dem Register 0x1600 zugewiesen werden:

Position im Telegramm	B0	B1	B2	B3	B4	B5	B6	B7
Wertigkeit	-	LB	HB	-	LB	HB	LB	HB
Wert	0x40	0x00	0x16	0x01	0x08	0x00	0x52	0x23
Bezeichnung	SDO Download-Request	Register 0x1600		Subindex	8 bit Datenlänge Datentyp UINT8		Zu mappendes Register	

Tabelle 5.4: Aufbau eines Download-Requests um das Register 0x2352 auf das Register 0x1600 zu mappen. B=Byte, LB=LowByte, HB=HighByte

Auf diese Art und Weise können die PDOs eines CAN-Bus-Teilnehmers konfiguriert werden. Die Durchführung ist in Abschnitt A.3 zu sehen.

Da die Nabenmotoren nach jedem Neustart ihre Konfiguration verlieren, ist es theoretisch nötig, diese Schritte jedes Mal von Hand durchzuführen. Um das System mit möglichst

wenigen Handgriffen betriebsbereit zu setzen, wird die Konfiguration durch das MicroGiant-Board durchgeführt. Die essentiellen Methoden dafür sind im Anhang unter Tabelle **??** beschrieben.

5.2.4 Mapping der Prozessdatenobjekte auf das MicroGiant-Board

Damit die Antriebe mit dem Mikrocontroller betrieben werden können, muss ein Objectdictionary für den Mikrocontroller selbst angelegt werden. Das Zuweisen der Prozessdatenobjekte auf das Objectdictionary des MicroGiant-Boards erfolgt mithilfe des Tools *Object Dictionary Editor*[4].

Das Hauptfenster, in welchem allgemeine Eintstellungen vorgenommen werden können, ist in Abbildung A.6 zu erkennen. Es werden bereits vorgefertigte Profile geladen, welche in der CANopen Spezifikation Standard sind; dazu gehört das Profil 301.
Um das vollständige Objectdictionary anzulegen, ist es sinnvoll sich im Vorfeld eine Namenssyntax für die Register zu überlegen, da diese die Anwendung der Register im Code erleichtert. Tabelle A.4 und Tabelle A.3 zeigen übersichtlich den Aufbau.

Das vervollständigte Objectdictionary mit den zugehörigen Mappings ist in Abbildung A.7 und Abbildung A.8 zu erkennen. Letzendlich wird das Dictionary über die CANopen Exportfunktion in eine *OD.c* und *OD.h* exportiert und in das Arbeitsverzeichnis für den CANopen-Stack gespeichert.

Wichtige Anmerkung: Das Tool exportiert die Dateien *OD.c* und *OD.h* nicht fehlerfrei. Die nötigen Änderungen können im Unterabschnitt A.2.2 eingesehen werden.

5.2.5 Anwenden von CANopen auf dem MicroGiant-Board

In der verwendeten CANopenNode-Vorlage sind bereits die wichtigsten Teile des CANopen-Stacks implementiert, um mit CANopen arbeiten zu können. Der Ersteller des Stacks hat einen spezifischen Bereich für eigenen Code zugelassen. Demnach soll sich die Low-Level-Motorsteuerung im Schleifenkörper *while(reset == CO_RESET_NOT)* befinden. Neben der Low-Level-Motorsteuerung müssen zusätzlich drei Methoden (*CO_process*, *CO_CANinterrupt* und *vTaskDelay*) implementiert werden, welche in jedem Zyklus aufgerufen werden.

[4] https://github.com/robincornelius/libedssharp, letzter Zugriff: 01.04.2023

CO_process: Ist unter anderem für das Handling der SDO- und PDO-Protokolle zuständig.
CO_CANinterrupt: Ist für das Einlesen von Events im PDO-Protokoll zuständig.
vTaskDelay: Begrenzt die maximale Bearbeitungszeit der Schleife und stellt Hintergrundprozessen Rechenzeit zur Verfügung.

Somit können im Schleifenkörper aus dem Codeauszug 5.1 sämtliche Funktionen für den Basisroboter implementiert werden.

```
...
while (reset == CO_RESET_NOT){
  reset = CO_process(CO, false, timeDifference_us, NULL);
  // Implement code here

  CO_CANinterrupt(nullptr);
           vTaskDelay(MAIN_WAIT / portTICK_PERIOD_MS);
}
```

Codeauszug 5.1: Auszug aus dem Hauptcode

5.3 Kommunikation zwischen MicroGiant-Board und Jetson

Um die vom Jetson generierten Sollwerte an den Mikrocontroller zu übertragen und die Istwerte der Antriebe zurückzumelden, bieten die Controller verschiedene Protokolle an. Um eine Entscheidung für ein geeignetes Übertragungsprotokoll zu treffen, werden die Datenblätter nach entsprechenden Übertragungstechniken analysiert. Zunächst wird die zu übertragene Bitrate bestimmt. Ein Sende- und Empfangszyklus soll maximal 10 ms dauern. Mit dieser Zeit wurden in der Simulation bereits gute Ergebnisse erzielt.

$$Daten_{Sollwerte} = 32\,bit \cdot 8 = 256\,bit \tag{5.1}$$

$$Daten_{Istwerte} = 32\,bit \cdot 8 = 256\,bit \tag{5.2}$$

$$\sum Daten = 512\,bit \tag{5.3}$$

$$Bitrate = \frac{512\,bit}{0.01s} = 512000\frac{bit}{s} = 512\frac{kbit}{s} \tag{5.4}$$

Für die Wahl des Protokolls werden folgende Anforderungen gestellt:

- Echtzeitfähigkeit
- Robustheit hinsichtlich der Anschlüsse an den Controllern (unempfindlich gegenüber Erschütterungen)
- Übertragungsgeschwindigkeit mit einer Bitrate von 512 kbit pro Sekunde
- Einschätzung Inbetriebnahmedauer

Tabelle 5.5 zeigt eine kurze Gegenüberstellung der vorhandenen Kommunikationswege zwischen dem Jetson und dem MicroGiant-Board. Aus der Tabelle geht hervor, dass die Kommunikation über Ethernet grundsätzlich nicht der Anforderung der Echtzeitfähigkeit entspricht.
Das liegt unter anderem daran, dass es keine direkte Verbindung zwischen den Controllern gibt, sondern ein Umweg über den Switch genommen werden muss. Gegen I2C spricht die zusätzliche Verdrahtung und der Auslegungsaufwand der Pullup-Widerstände. SPI hat den Nachteil, dass es nur über den Pinheader der Controller angeschlossen werden kann[5][24].
UART setzt sich bei dieser Entscheidung durch, da die Controller über USB-Typ A (Jetson) und USB-Typ Micro (MicroGiant-Board) eine zuverlässige Verbindung gewährleisten. Zusätzlich wird die Implementierung in ROS2 durch die Bibliothek *pySerial*[5] erleichtert.

Protokoll	Echtzeit-fähig	Anschlussart	Übertragungs-rate	Einschätzung Inbetriebnahmedauer
SPI	Ja	4 Leitungen Pins <->Pins	$> 512\frac{kbit}{s}$	Schnell Anschluss über Pins
UART	Ja	USB<->USB	$> 512\frac{kbit}{s}$	Schnell gegebene Bibliothek USB-Buchse vorhanden
I2C	Ja	2 Leitungen Pins<->Pins	$> 512\frac{kbit}{s}$	Lange Auslegung von Pullup-Widerständen
Ethernet	Nein	RJ45->Switch->RJ45	$> 512\frac{kbit}{s}$	Mittel Konfiguration komplex

Tabelle 5.5: Gegenüberstellung der vorhandenen Kommunikationsmöglichkeiten

[5]https://pyserial.readthedocs.io/en/latest/pyserial.html, letzter Zugriff: 28.03.2023

5.3.1 Implementierung

Die Implementierung der seriellen Schnittstelle erfolgt semantisch sowohl auf dem MicroGiant-Board, als auch identisch auf der Jetson Seite. Der Ablauf ist in Abbildung 5.4 dargestellt. Prinzipiell wird eine Zeichenkette (String) erzeugt, welche die Sollposition und Solldrehzahlen für die Antriebe bereit hält. Dieser String wird vom Jetson zum MicroGiant-Board verschickt. Umgekehrt wird ein String von MicroGiant-Board mit den Istpositionen und Istdrehzahlen an den Jetson geschickt. Der Aufbau des Strings ist in beiden Fällen identisch und wird in Codeauszug A.3 beschrieben. Die Kommata machen die Werte voneinander unterscheidbar. Das „\n“ gibt darüber Auskunft, wann die Zeichenkette zu Ende ist.

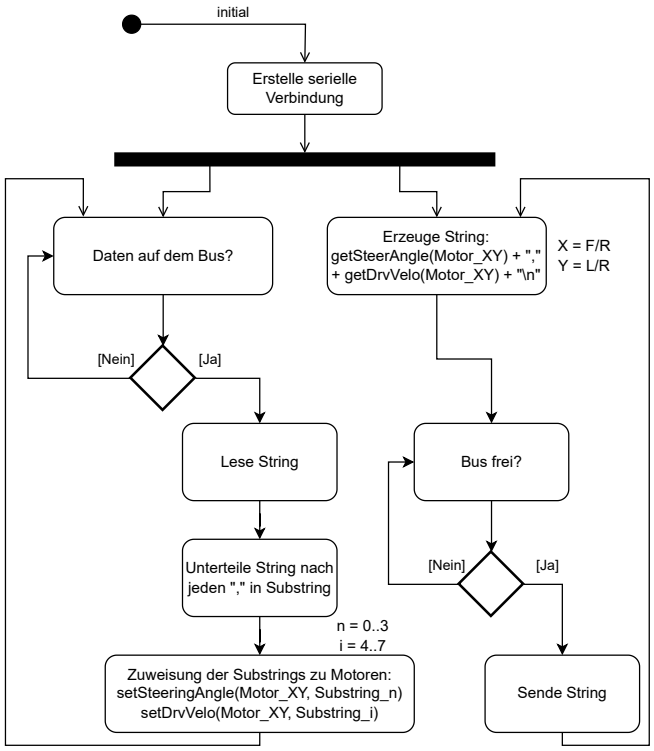

Abbildung 5.4: Schematische Darstellung des Ablaufes der seriellen Schnittstelle. Die Semantik wird sowohl für das MicroGiant-Board als auch für das ROS2 Programm übernommen

Für die Nutzung des USB-Ports unter Ubuntu ist es nötig, den Port für Lese- und Schreibrechte freizugeben. Dazu wird im Verzeichnis */etc/udev/rules.d* eine Datei mit dem Namen *50-ttyusb.rules* erstellt.

```
sudo nano /etc/udev/rules.d/50-ttyusb.rules
```

In dieser wird nun das genutzte Ubuntu-Subsystem für Laufwerke (*tty*), eine Händler-ID und Produkt-ID durch das Einfügen der nächsten Zeile vergeben.

```
SUBSYSTEM=="tty", ATTRS{idVendor}=="<vendor_id>", ATTRS{idProduct}==
    "<product_id>", MODE="0666"
```

Nach dem Einstecken des MicroGiant-Boards und mit dem Befehl *lsusb* werden alle USB-Geräte mit der Händler- und Produkt-ID aufgelistet. Für das MicroGiant-Board ergibt sich daher folgende Zeile:

```
SUBSYSTEM=="tty", ATTRS{idVendor}=="10c4", ATTRS{idProduct}=="ea60",
    MODE="0666"
```

Mit dieser Konfiguration und dem in Codeauszug A.3 gezeigten Aufbau, lassen sich die Strings in seine einzelnen Variablen zerlegen und die Kommunikation zwischen beiden Geräten ist vorbereitet.

5.4 Inbetriebnahme

Aufgrund von Lieferengpässen kann die Roboterbasis bis zum Ende dieser Arbeit nicht fertiggestellt werden. Es fehlen torsionsfähige Motorkabel zum Anschluss der Nabenmotoren und Gleichspannungssicherungen. Aus diesem Grunde erfolgt die Inbetriebnahme der Motoren isoliert vom Basisroboter als Komponententest.

Beide Motoren werden mithilfe einer Halterung an einem Tisch befestigt, sodass die Wellen frei drehen können. Mithilfe eines Netzteils wird der Nabenmotor, der Lenkmotor und das MicroGiant-Board mit Elektrizität versorgt. Über eine serielle Schnittstelle auf dem MicroGiant-Board erfolgt die Sollwertvorgabe. Bei diesem Versuchsaufbau wurde keine torsionsfähige Motorleitung für den Nabenmotor verwendet.

5.4.1 Nabenmotor

Für die Inbetriebnahme des Nabenmotors ist es nötig, bei jedem Neustart eine Konfiguration durchzuführen. Dies geschieht durch das Hochfahren des Mikrocontrollers und des im Unterabschnitt 5.3.1 beschriebenen Codes.

Der Motor wird getestet, indem eine beliebige Solldrehzahl eingestellt und die Istdrehzahl aufgenommen wird. Es werden die Auslieferungsswerte für die Beschleunigungs- und Verzögerungswerte beibehalten. Diese sollten, wenn nötig, bei Testfahrten im fertigen System angepasst werden, da ein bspw. zu starkes Verzögern die Fahrstabilität der Roboterbasis gefährden könnte. Der Nabenmotor wird während der Inbetriebnahme unbelastet betrieben. In Abbildung 5.5 ist die Soll- und Istdrehzahl eines Nabenmotors dargestellt. Ausgegeben wird die Istdrehzahl durch den Motortreiber. Auffällig ist ein Nichterreichen bzw. kontinuierliches Pendeln der Istdrehzahl um den Sollwert mit zwei konstanten Werten. Die reale Drehzahl am Rad wurde mit einem Handtachometer gemessen. Sie entsprach dabei nach einer minimalen Einregelzeit der Solldrehzahl. Vom Hersteller werden keine Angaben zur Genauigkeit und Auflösung des Mess- und Steuersystems gemacht. Die Ursache für die fehlerhafte Ausgabe konnte nach Korrespondenz auf die fehlerhafte Software des Motortreibers zurückgeführt werden.

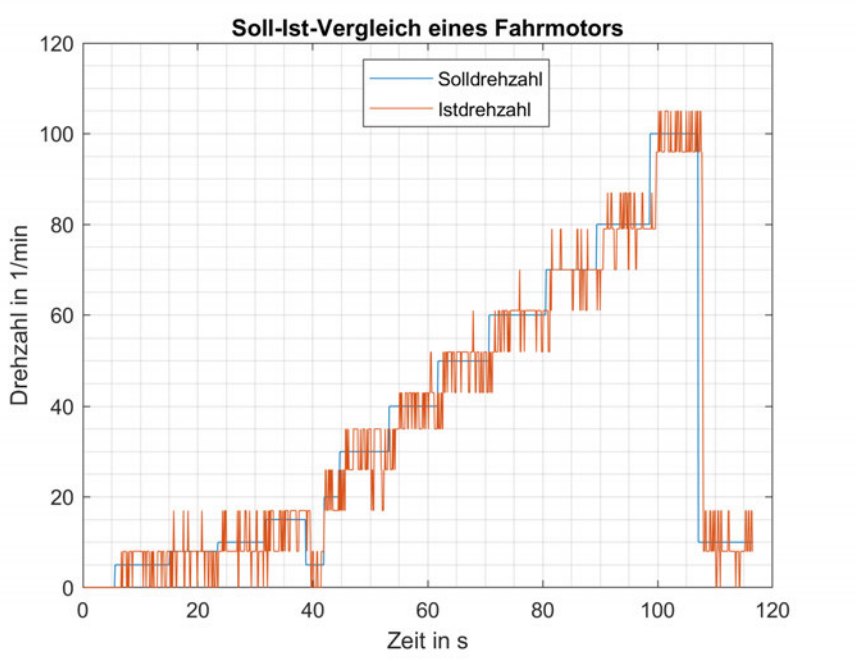

Abbildung 5.5: Vergleich der Soll- und Istdrehzahl eines Nabenmotors

5.4.2 Lenkmotor

Analog zum Nabenmotor wird der Lenkmotor im nicht eingebauten Zustand getestet. Es wird eine Sollposition in Grad vorgegeben und die ausgegebene Istposition mithilfe des Absolutwertgebers ausgelesen. Abbildung 5.6 zeigt, wie der Lenkmotor auf eine vorgegebene Größe reagiert. Es ist nach circa zehn Sekunden zu erkennen, dass der Motor grundsätzlich die vorgegebene Istposition erreichen kann. Aus dem Zeitbereich 11 - 15 Sekunden wird ersichtlich, dass das Programm und der Motor mit einer Sollwertänderung während eines anderen Fahrbefehls umgehen kann und den neuen Sollwert annimmt.
Im Bereich 20 - 30 Sekunden wird ersichtlich, dass der Motor kleine Wertänderungen verarbeiten kann.
Die Verzögerung von circa einer Sekunde zwischen Sollwertsprung und Istwertsprung bei Sekunde 6 entsteht durch die Verzögerung aus dem verwendeten seriellen Monitor. Dieser hat für den Roboterbasis keine negativen Auswirkungen, da dieser nicht verwendet wird.

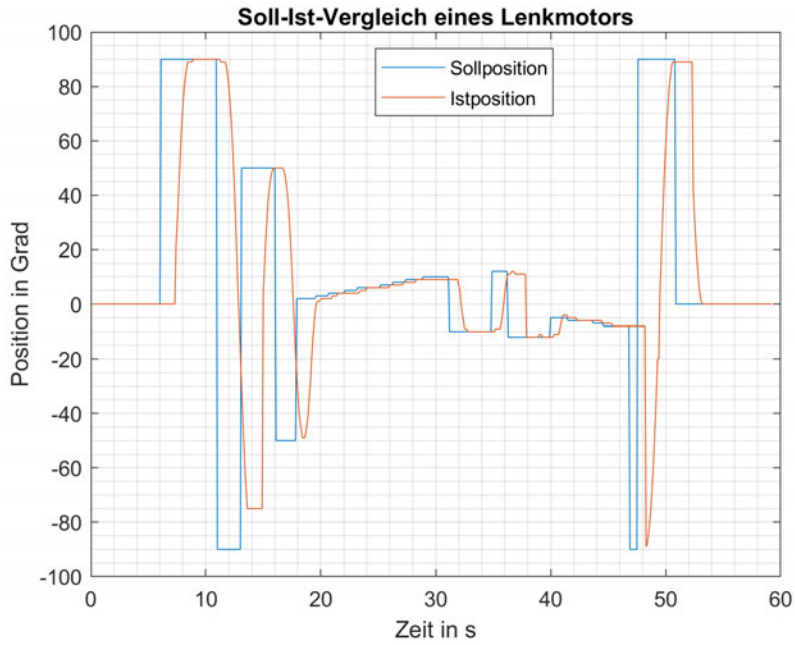

Abbildung 5.6: Vergleich der Soll- und Istposition eines Lenkmotors

Für die Inbetriebnahme der Roboterbasis muss für jede Achse die 0-Grad Position eingestellt werden. Dieses lässt sich durch das Herstellertool parametrieren.

5.5 Theoretische Inbetriebnahme

Da eine praktische Inbetriebnahme der Roboterbasis nicht stattfinden kann, wird das ROS2-Programm für die Inbetriebnahme vorbereitet. Das MicroGiant-Board ist durch den im Abbildung 5.3 beschriebenen Ablaufes bereits vollständig für die Inbetriebnahme der Roboterbasis programmiert. Sobald eine serielle Verbindung zwischen dem Jetson und dem MicroGiant-Board aufgebaut wird, werden die Soll- und Istwerte ausgetauscht.

Für das ROS2-Programm ist es nötig, das Skript für

- die Fahrsteuerung *(four_ws_control.launch.py)*
- die Schnittstelle zum Playstation 4-Controller (in der Fahrsteuerung bereits enthalten)
- die serielle Kommunikation *(uart.launch.py)*

zu starten, sobald der Jetson hochgefahren ist. Die genannten Skripte werden mithilfe des Skripts *kribl_bringup.launch.py* gestartet.

Aus Sicherheitsgründen wird für die Ersteinschaltung und für die weitere Inbetriebnahme auf das Starten der Skripte direkt beim Einschalten der Roboterbasis verzichtet. Das könnte zu ungewollten Verfahrbewegungen im derzeitigen Entwicklungsstand führen. Stattdessen wird die *launch*-Datei händisch über die Kommandozeile unter ROS2 gestartet:

```
1 ros2 launch kribl_4ws kribl_bringup.launch.py
```

Codeauszug 5.2: Befehl um das Fahren mit der Roboterbasis zu ermöglichen

Unter Abbildung A.10 sind die Nodes dargestellt, die für die reale Roboterbasis verwendet werden. Diese erscheinen nach dem Starten des *launch*-Befehls.

Abschließend kann der PlayStation 4-Controller über Bluetooth angeschlossen werden und die Roboterbasis sollte sich in Bewegung setzen.

6 Vermessung und Bewertung der Bahntreue

Die Bahntreue kann aufgrund des nicht fertigen Aufbaus nur durch Simulation getestet werden. Es sollen die Auswirkungen des implementierten Fahralgorithmus getestet werden. Neben der allgemeinen Fahrbahntreue soll untersucht werden, welche Auswirkung die Variation des Kraftbeischlusswertes in der Simulation hat. Dafür wird der Kraftbeischlusswert μ_h für einen festen Untergrund $\mu_h = 0,75$ und für einen losen Untergrund $\mu_h = 0,60$ verwendet. Der zulässige Querschlupf beträgt $S_\alpha = 0,1$.

6.1 Messmethodik

Um die Auswirkungen des Algorithmus auf das Roboterbasismodell zu erkennen, ist es sinnvoll, den zurückgelegten Weg seit Start, auch Odometrie genannt, zu messen. Vor allem sollen Lenkvorgänge getestet werden. Es wird eine beliebige Trajektorie mit Kurven festgelegt.

Die Solltrajektorie läuft 30 Sekunden und besteht aus

1. 3 Sekunden X-Richtung $v = 1,5\frac{m}{s}$, Rotation $\omega = 0,0\frac{rad}{s} \rightarrow Geradeausfahrt$
2. 6 Sekunden X-Richtung $v = 1,5\frac{m}{s}$, Rotation $\omega = 2,25\frac{rad}{s} \rightarrow Linksfahrt$
3. 6 Sekunden X-Richtung $v = 1,5\frac{m}{s}$, Rotation $\omega = 2,25\frac{rad}{s} \rightarrow Rechtsfahrt$
4. 6 Sekunden X-Richtung $v = 1,5\frac{m}{s}$, Rotation $\omega = 1,5\frac{rad}{s} \rightarrow Linksfahrt$
5. 3 Sekunden X-Richtung $v = 1,5\frac{m}{s}$, Rotation $\omega = 1,5\frac{rad}{s} \rightarrow Rechtsfahrt$
6. 6 Sekunden X-Richtung $v = 1,5\frac{m}{s}$, Rotation $\omega = 0,0\frac{rad}{s} \rightarrow Geradeausfahrt$

Um die Odometrie aufzunehmen, wird das Plugin *libgazebo_ros_p3d* in der *ros2_control.xacro*-Datei angelegt. Als Bezug wird die allgemeine Karte (*map*) angenommen. Der zu messende Körper ist die Basis *base_link*, wobei aus dem Modellmittelpunkt gemessen wird. Mithilfe des Plotjugglers[1] kann das Topic mit den Odometriedaten abonniert und in eine .csv-Datei gespeichert werden.

6.2 Messergebnisse

Abbildung 6.1 und Abbildung 6.2 zeigen die Odometrie in der Draufsicht des gefahrenen Weges der Roboterbasis. Die Startkoordinate ist bei (0|0); die Endkoordinate variiert je nach Fall.
Ausgehend vom Startpunkt ist zu erkennen, dass Lenkeingriffe bei der Nutzung des Algorithmus verzögert stattfinden. Während die blaue Kurve bereits im ersten halben Meter einlenkt, startet die orange Kurve erst ab dem zweiten gefahrenen Meter. Beide Graphen zeigen, dass die Nutzung des Algorithmus zur Bodenschonung für ähnliche Kurvenradien sorgt. Unterschiedlich sind die Spitze-Tal-Werte in X-Richtung. Eine markante Wendung stellen dabei die Kurven um die Koordinate $(2|-4)$ in Abbildung 6.1 dar. Ohne die Nutzung des Algorithmus fährt die Roboterbasis einen Meter weniger in X-Achsenrichtung im Vergleich zur Kurve mit dem Algorithmus. In Abbildung 6.2 ist eine Differenz zwischen den zwei Kurven von $5.4\,m - 0.4\,m = 5\,m$ am Kurvenmaxima der dritten Wendung auffällig.

In Tabelle 6.1 werden die jeweiligen Kurven hinsichtlich ihrer Odometrie bei variablem Kraftbeischlusswert verglichen.

Kurvenart	m.A	o.A	Abweichung	m.A	o.A	Abweichung
Kraftschlussbeiwert	0,60		absolut in m	0,75		absolut in m
1. Kurve Maximum X-Richtung in m	2,4	2,8	0,4	1,9	1,7	0,2
2. Kurve Maximum X-Richtung in m	-0,8	-1,0	0,2	0,5	-1,1	1,6
3. Kurve Maximum X-Richtung in m	0,3	-0,2	0,5	5,3	0,8	4,5

Tabelle 6.1: Vergleich der simulierten Kraftbeischlusswerte. m.A = mit Algorithmus, o.A = ohne Algorithmus

[1] https://github.com/facontidavide/PlotJuggler, letzter Zugriff: 07.04.2023

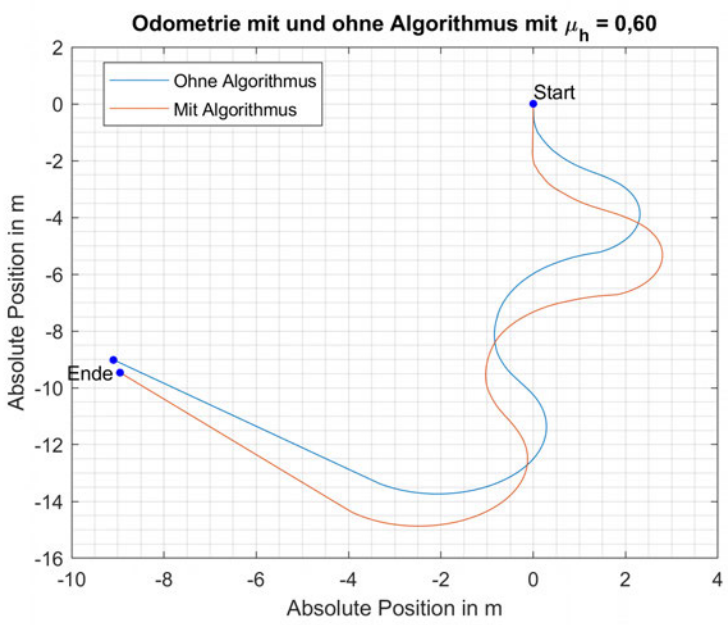

Abbildung 6.1: Vergleich der Odometrie mit und ohne Bodenschon-Algorithmus bei einem $\mu_h = 0,60$

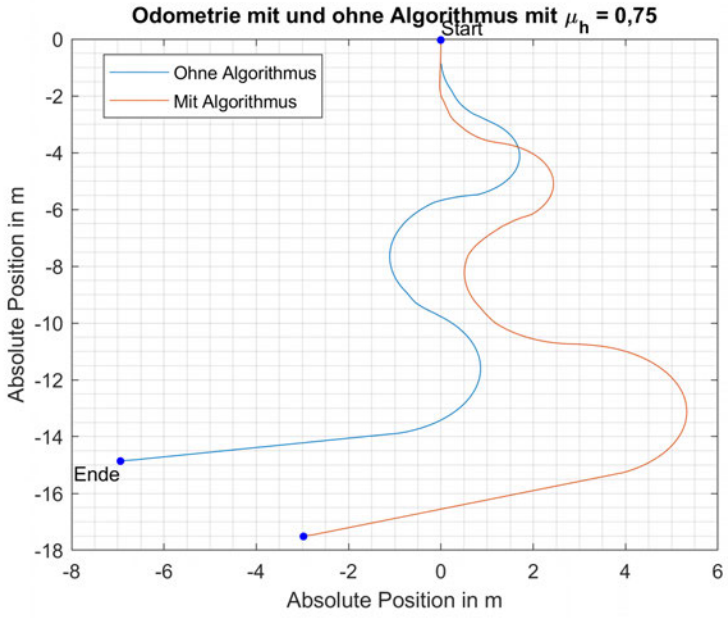

Abbildung 6.2: Vergleich der Odometrie mit und ohne Bodenschon-Algorithmus bei einem $\mu_h = 0,75$

6.3 Bewertung

Nach der Durchführung der Simulation werden die aufgenommenen Messergebnisse bewertet. Hauptaugenmerk liegt vor allem auf der Bodenschonung. Aus Abschnitt 6.2 geht hervor, dass die Nutzung des Algorithmus zu einem verzögerten Start der Lenkung führt. Das ist mit der Gleichung 4.11 zu erklären, denn diese sorgt bei einer anfänglich geringen Drehzahl des Rades für eine höhere Verzögerungzeit der Lenkung. Dieses ist positiv zu bewerten, denn eine kleine Lenkwinkeländerung sorgt für geringere plastische Verschiebung loser Erde.

Positiv ist auch, dass der Algorithmus auf einem Untergrund mit geringerer Traktion ($\mu_h = 0,6$) eine geringere absolute Abweichung (0,5 m zu 4,5 m) im Vergleich zu einem Boden mit höherer Traktion (0,75) erzeugt. Ein Grund für dieses Ergebnis könnte sein, dass auf einem Boden mit hoher Traktion größere Lenkwinkeländerungen aufgenommen werden können und das Fahrzeug in die entstprechende Richtung lenkt. Während bei gleicher Trajektorie aber geringerer Traktion, der Reifen in Kurven stärker durchdreht und die Kraft schlechter in den Boden eingeleitet werden kann. Dadurch fährt das Fahrzeug nicht in die Sollrichtung.

Weiterhin wird ersichtlich, dass die Nutzung des Algorithmus zu einer deutlichen Abweichung zur eigentlichen Trajektorie führt. Erkennbar ist das vor allem in Abbildung 6.2 mit einer absoluten Abweichung in X-Achsenrichtung von 5 m. Der Untergrund wird zwar geschont, der Roboter befände sich aber an einem anderen Ort als gewollt.

Insgesamt zeigt sich, dass ein theoretisch bodenschonenderes Fahren möglich ist. Für die Verifizierung des Algorithmus ist jedoch ein praktischer Test vonnöten.

6.4 Ableitung von Verbesserungsvorschlägen

Aufgrund der hohen Abweichung zwischen Soll- und Istposition ist es empfehlenswert ein Regelungssystem für die Positionierung des Basisroboters zu implementieren. Mit einer Regelung könnte durch Gegenlenkmaßnahmen eine Zieltrajektorie mit kleiner Abweichung erreicht werden.

Es könnte zusätzlich über das Anwendungsgebiet nachgedacht werden. Bei der Fahrt ohne Regelung über einer gerade ausgesäte Saatreihe wird eine Isttrajektorie mit wenig Abweichung zur Solltrajektorie benötigt. Wohingegen der Algorithmus bei einer autonomen Fahrt auf dem Feld, mit der Suche nach Unkräutern und mit vielen Lenkvorgängen dazu beitragen kann, den Boden zu schonen.

Da bisher nur ein statischer Wert für den Schlupf angegeben ist, sollte eine Methode eingeführt werden, um den realen Schlupf aufzunehmen. Darauf aufbauend lässt sich ein optimaler Schlupf bestimmen, an dem die größte Traktion stattfinden kann. Damit könnte der Algorithmus bei besseren Bodenverhältnissen einen größeren Lenkwinkel zulassen, wodurch eine kleinere Abweichung zur Solltrajektorie entstehen könnte.

Bei dieser Arbeit wurde der Fokus auf den Querschlupf gelegt. Aus dem Kammschen Kreis geht hervor, dass der Querschlupf in der üblichen Fahrpraxis immer in Verbindung mit dem Antriebsschlupf einhergeht. Daher ist es nötig, zusätzlich den Antriebsschlupf zu ermitteln, um den Boden auch während des Antriebschlupfes zu schonen.

7 Fazit

Diese Arbeit lässt sich in zwei Arbeitspakete teilen. Zum einen die Implementierung einer Fahrsteuerung in der Low-Level-Schicht, zum anderen die Entwicklung eines Algorithmus zur Vermeidung von Bodenschäden bei Lenkvorgängen.
Anfänglich wurde die verwendete Hardware vorgestellt und anschließend das Know-How angeeignet, um die vorhandenen Antriebe über das CANopen-Protokoll anzusteuern. Für die Entwicklung eines bodenschonenden Algorithmus mussten die Kräfte und die Zusammenhänge zur Krafteinleitung in den Boden analysiert werden. Zusätzlich war eine Einarbeitung in das ROS2-Framework nötig, um Fahrbefehle zu erzeugen und eine Simulationsumgebung zu erstellen.
Mit dem MicroGiant-Board werden die empfangenen Sollgrößen über CANopen an die Antriebe geleitet. Um das zu ermöglichen, wurde eine Open-Source-Bibliothek verwendet sowie ein alter Portierungsstand. Das entwickelte Programm auf dem Mikrocontroller umfasst eine aktualisierte Bibliothek. Die nötigen Fahrrichtungsbefehle wurden von einem weiteren Mikrocontroller (Jetson), erzeugt. Auf diesem wurde ein ROS2-Programm implementiert, welches neben der eigentlichen Fahrsteuerung auch eine Simulation des Basisroboters selbst enthält. Mit der Simulation konnte die Fahrsteuerung vor dem Aufbau der Roboterbasis getestet werden. Dazu wurde ein PlayStation 4-Controller über Bluetooth mit dem Jetson verbunden und in der Simulation der Roboter damit gesteuert. Neben der Fahrsteuerung wurde der Algorithmus getestet und ausgewertet. Der selbst entwickelte Algorithmus fährt theoretisch bodenschonend. Diese Eigenschaft muss in ausführlichen praktischen Tests nachgewiesen werden. Praktische Tests waren zum Ende dieser Arbeit nicht möglich, da die Roboterbasis nicht vollständig aufgebaut war. Gründe dafür waren Lieferengpässe von elektrischen Komponenten. Mit dem Ende dieser Arbeit wurde jedoch die Low-Level-Fahrsteuerung sowie, das ROS2 Programm für die Roboterbasis vorbereitet und ein theoretisch funktionstüchtiger Algorithmus für die bodenschonende Navigation entwickelt und virtuell getestet.

8 Ausblick

Für die Roboterbasis wurde mit dieser Arbeit bereits eine gute Grundlage geliefert. Obwohl die Basis noch nicht aufgebaut ist, bestehen bereits Optimierungspotenziale. Negativ zu bewerten ist zurzeit die feste Schlupfvorgabe. Für zukünftige Arbeitspakete sollte eine Methode implementiert werden, welche die reale Odometrie des Roboters aufzeichnen kann. Damit ließe sich der optimale Schlupf bestimmen, was eine bessere Anpassung an reale Gegebenheiten ermöglicht. Gängige Methoden sind beispielsweise der Einbau einer Inertial Measurement Unit (IMU) wie in der Kraftfahrzeugtechnik. Aus der Raddrehzahl und dem realen Geschwindigkeitvektors des Fahrzeuges kann der Antriebsschlupf und der Querschlupf ermittelt werden. Durch die Vorgabe eines Schlupfkennfeldes und des erstellten Algorithmus könnte ein optimiertes System zum bodenschonenden Fahren bei gleichzeitigem Einhalten der Solltrajektorie geschaffen werden.
Optimierungsbedarf besteht durch die Wahl der Nabenmotoren. Die Rückgabewerte der Drehzahl entsprechen nur in seltenen Fällen der realen Drehzahl. Stattdessen pendelt die Ausgabe über oder unter dem tatsächlichen Istwert. Für regelungstechnische Aufgaben stellen solche Sprünge eine Herausforderung dar.
Auch bei den Lenkmotoren gibt es eine Möglichkeit zur Optimierung. In dieser Roboterbasis ist der Drehgeber zur Positionsbestimmung direkt am Motor installiert. Das verbaute Getriebe besitzt bauartbedingt ein Getriebespiel von $\pm 1°$. Bei kleinen Lenkwinkeländerungen kann es vorkommen, dass diese tatsächlich nicht stattfinden. Eine Lösung könnte ein installierter Absolutdrehgeber auf der Antriebswelle sein. Nach Rücksprache mit dem Hersteller ließe sich dieser sogar direkt mit dem Lenkmotor koppeln. Die bereits implementierte Regelung des Motors könnte so als eigenständige Einheit verwendet werden.
Mit dieser Arbeit wird eine softwaretechnische Grundlage geliefert, um die Roboterbasis anzusteuern. Nach elektrischem Aufbau der Roboterbasis sollte sich die Inbetriebnahmedauer auf ein Minimum beschränken. Die Installation eines Roboterarmes oder anderen Werkzeugen stellt dabei kein Problem mehr dar.
Durch die Implementierung von ROS2 können mithilfe von Kameras und Navigationsalgorithmen komplexe Aufgaben wie das Zupfen von Unkraut realisiert werden.

Literaturverzeichnis

[1] Bendler, Thomas: *Motorrad/ Physik/ Kammsche Kreis.* – URL https://www.thbe.org/motorcycle/physik/kammsche_kreis/. – Zugegriffen am: 25.02.2023

[2] Blume, Hans-Peter ; Brümmer, Gerhard W. ; Horn, Rainer ; Kandeler, Ellen ; Kögel-Knabner, Ingrid ; Kretzschmar, Ruben ; Stahr, Karl ; Wilke, Berndt-Michael: *Scheffer/Schachtschabel: Lehrbuch der Bodenkunde* -. Berlin Heidelberg New York : Springer-Verlag, 2016. – ISBN 978-3-662-49960-3

[3] Bundesministerium für Ernährung und Landwirtschaft: *KRIBL – erschwingliche Robotik auch für kleine Betriebe.* – URL https://www.bmel.de/SharedDocs/Praxisbericht/DE/kuenstliche-intelligenz/KRIBL.html. – Zugegriffen am: 15.01.2023

[4] Ersoy, Metin ; Gies, Stefan: *Fahrwerkhandbuch - Grundlagen – Fahrdynamik – Fahrverhalten– Komponenten – Elektronische Systeme – Fahrerassistenz – Autonomes Fahren– Perspektiven.* Berlin Heidelberg New York : Springer-Verlag, 2017. – ISBN 978-3-658-15468-4

[5] eVision Systems GmbH: *I2C vs SPI Protokoll: Unterschiede und Ähnlichkeiten.* – URL https://evision-webshop.de/Total-Phase-Knowledge-Base/Promira/I2C-vs-SPI-Protokoll-Analysatoren-Unterschiede-und-Aehnlichkeiten. – Zugegriffen am: 26.03.2023

[6] Frömmig, Lars: *Grundkurs Rennwagentechnik - Einführung in das Zusammenwirken von Reifen, Fahrwerk, Aerodynamik, Differenzialsperren und Rahmen.* Berlin Heidelberg New York : Springer-Verlag, 2020. – ISBN 978-3-658-25044-7

[7] GmbH, KUNBUS: *Echtzeit-Ethernet-Kommunikation.* – URL https://www.kunbus.de/echtzeitkommunikation. – Zugegriffen am: 15.04.2023

[8] Gomeringer, Roland ; Heinzler, Max ; Kilgus, Roland ; Menges, Volker ; Stephan, Andreas ; Näher, Friedrich ; Oesterle, Stefan ; Scholer, Claudius ; Wieneke, Falko: *Tabellenbuch Metall* -. Haan-Gruiten : Verlag Europa Lehrmittel Nourney, Vollmer GmbH & CompanyKG, 2017. – ISBN 978-3-808-51678-2

[9] HP Wizard: *Tire friction and rolling resistance coefficients.* – URL http://hpwizard.com/tire-friction-coefficient.html. – Zugegriffen am: 04.04.2023

[10] John Deere: *Autonomie durch Automatisierung.* – URL https://www.deere.de/de/landtechnik/landwirtschaft-der-zukunft/. – Zugegriffen am: 14.01.2023

[11] Karkee, Manoj ; Zhang, Qin: *Fundamentals of Agricultural and Field Robotics* -. Singapore : Springer Nature, 2021. – ISBN 978-3-030-70400-1

[12] Mitschke, Manfred ; Wallentowitz, Henning: *Dynamik der Kraftfahrzeuge.* 4. Springer-Verlag, 2004. – ISBN 978-3-540-42011-8

[13] Nanotec Electronic GmbH Co. KG: *Technisches Handbuch PD4-E.* – URL https://en.nanotec.com/fileadmin/files/Handbuecher/Plug_Drive/PD4-E/fir-v2213/PD4E_CANopen_Technisches-Handbuch_V1.6.0.pdf?1663074312. – Zugegriffen am: 14.03.2023

[14] Open Robotics: *ROS 2 Releases and Target Platforms.* – URL https://www.ros.org/reps/rep-2000.html#foxy-fitzroy-may-2020-may-2023. – Zugegriffen am: 10.02.2023

[15] Open Robotics: *Ros2 Controller.* – URL https://github.com/ros-controls/ros2_controllers. – Zugegriffen am: 12.03.2023

[16] Open Robotics: *Understanding nodes.* – URL https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html. – Zugegriffen am: 14.01.2023

[17] Open Robotics: *Why ROS?.* – URL https://www.ros.org/blog/why-ros/. – Zugegriffen am: 25.02.2023

[18] Open Robotics: *Writing a new controller.* – URL https://control.ros.org/master/doc/ros2_controllers/doc/writing_new_controller.html. – Zugegriffen am: 01.03.2023

[19] Open Robotics: *Writing a simple publisher and subscriber (Python).* – URL https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Publisher-And-Subscriber.html. – Zugegriffen am: 15.04.2023

[20] Pfeffer, Peter ; Harrer, Manfred: *Lenkungshandbuch - Lenksysteme, Lenkgefühl, Fahrdynamik von Kraftfahrzeugen.* Berlin Heidelberg New York : Springer-Verlag, 2013. – ISBN 978-3-658-00977-9

[21] Rico, Francisco M.: *A Concise Introduction to Robot Programming with ROS2* -. Boca Raton, Fla : CRC Press, 2022. – ISBN 978-1-000-62981-1

[22] Robotmania: *4WS Theory.* – URL https://www.youtube.com/watch?v=VX53gAXafUA. – Zugegriffen am: 21.02.2023

[23] Schnell, Gerhard ; Wiedemann, Bernhard: *Bussysteme in der Automatisierungs- und Prozesstechnik - Grundlagen, Systeme und Anwendungen der industriellen Kommunikation.* Berlin Heidelberg New York : Springer-Verlag, 2019. – ISBN 978-3-658-23688-5

[24] Sercos International e.V.: *Echtzeit Ethernet.* – URL https://www.sercos.de/technologie/warum-ethernetechtzeit-ethernet/echtzeit-ethernet/. – Zugegriffen am: 26.03.2023

[25] Ubuntu Deutschland e.V.: *Python.* – URL https://wiki.ubuntuusers.de/Python/. – Zugegriffen am: 10.02.2023

[26] Wallentowitz, Henning ; Reif, Konrad: *Handbuch Kraftfahrzeugelektronik - Grundlagen - Komponenten - Systeme - Anwendungen.* Berlin Heidelberg New York : Springer-Verlag, 2006. – ISBN 978-3-8348-9121

[27] WayCon Positionsmesstechnik: *CANopen Manual.* – URL https://www.waycon.biz/fileadmin/draw-wire-sensors/CANopen-Manual.pdf. – Zugegriffen am: 15.03.2023

[28] W.Söhne: Reibung und Kohäsion bei Ackerböden. In: *Grundlagen der Landtechnik* (1953)

[29] W.Söhne: Beitrag zur Mechanik des Systems Fahrzeug-Boden unter besonderer Berücksichtigung der Ackerschlepper. In: *Grundlagen der Landtechnik* 17 (1963), S. 5–16

A Anhang

A.1 Simulation

A.1.1 Bedingungen für die Arbeitsumgebung

Durch die Wahl des Jetsons ergeben sich bezüglich des Betriebssystems Einschränkungen. Das offizielle Nvidia Image (Speicherabbild des Betriebssystems) basiert auf Ubuntu 18.04 Bionic Beaver. Für die Verwendung von ROS2 kann die Nutzung der Version 18.04 einschränkend sein, da einige Funktionen erst mit Python 3.8 funktionieren und diese erst mit einer neueren Ubuntu Version verfügbar sind[25][14]. Daher wird auf ein Open-Source-Image mit Ubuntu 20.04 zurückgegriffen[1]. Die verwendete ROS2 Version ist ROS2 Foxy. Dies ist die letzte Version, die für Ubuntu 20.04 unterstützt wird. Die Installation erfolgt nach der Anleitung unter https://docs.ros.org/en/foxy/Installation/Ubuntu-Install-Debians.html. Neben dem Betriebssystem sind zusätzliche Pakete zu installieren.

```
sudo apt install gazebo11
sudo apt install ros-foxy-gazebo-ros-pkgs
sudo apt install ros-foxy-gazebo-ros2-control
sudo apt install ros-foxy-ros2-control
sudo apt install ros-foxy-ros2-controllers
sudo apt install ros-foxy-xacro
```

Nach der Installation des Betriebssystems und der Pakete ist es empfehlenswert, der Kommandozeile einen Verweis auf die zu nutzenden Programme zu geben.

```
echo "source /usr/share/gazebo/setup.sh" >> /.bashrc
echo "source /opt/ros/foxy/setup.bash" >> /.bashrc
echo "source ~/ros2_ws/install/setup.bash" >> /.bashrc
```

Damit wird bei jedem Öffnen einer Kommandozeile automatisch das ROS2 Arbeitsverzeichnis, ROS2 Foxy und Gazebo verlinkt.

[1] https://github.com/Qengineering/Jetson-Nano-Ubuntu-20-image, letzter Zugriff: 12.03.2023

A.1.2 Ordnerstruktur ROS2-Programm

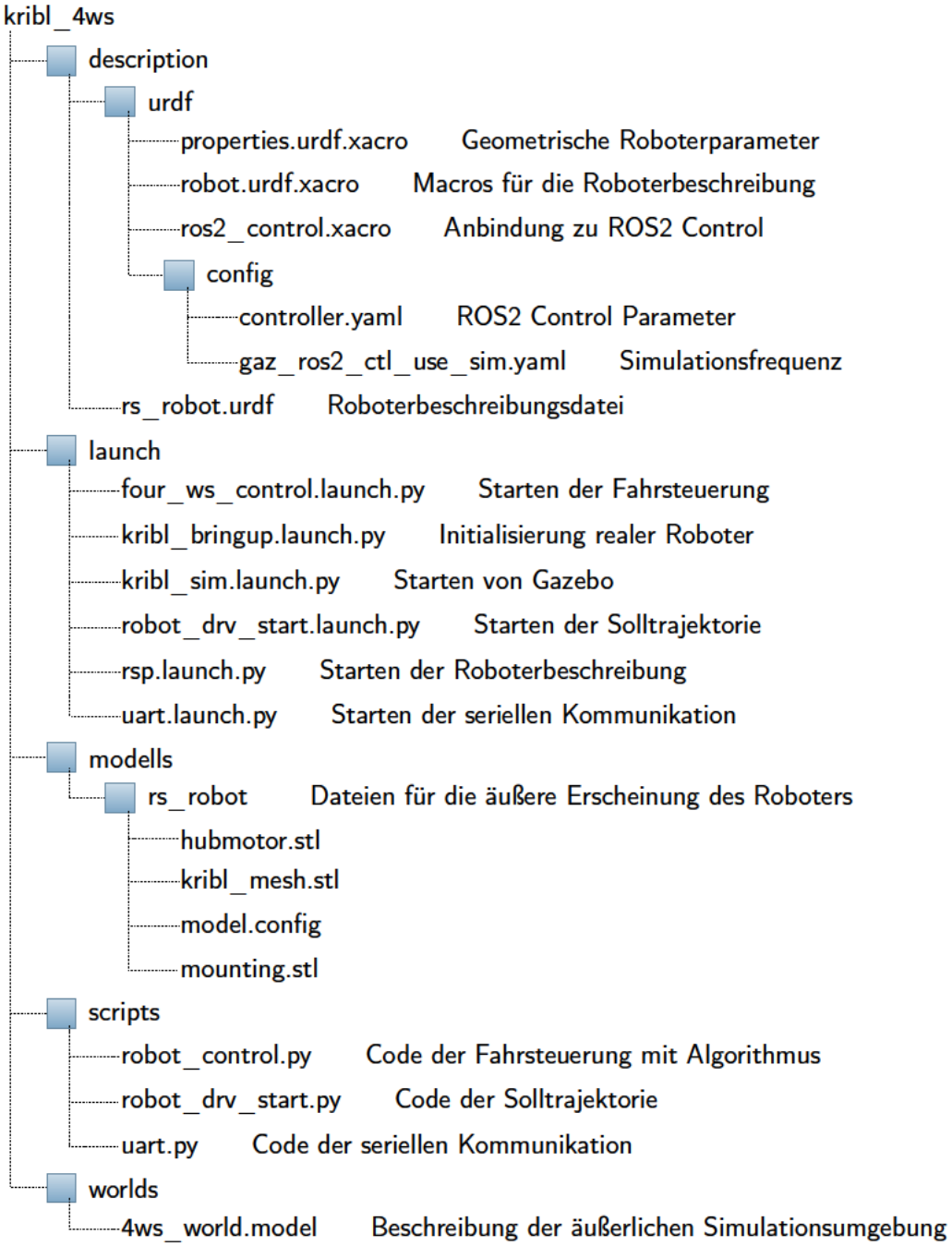

Abbildung A.1: Ordnerstruktur mit einer Aufgabenbeschreibung der jeweiligen Datei des ROS2-Programms

A.1.3 Simulationsparameter

Parameter	Wert	Bezeichnung
wheel_track	0,92 m	Spurweite
wheel_base	1,32 m	Radstand
wheel_radius	0,17 m	Radradius
wheel_steering_y_offset	0,06 m	Lenkrollradius
wheel_steering_track	0,80 m	Chassibreite

Tabelle A.1: Verwendete geometrische Größen des Robotermodells in der Simulation

A.2 Methoden für die SDO-Konfiguration der Fahrmotoren

Methode	Aufgabe
CO_SDOclientUploadInitiate()	Aktivieren des jeweiligen PDOs
CO_SDOclientDownloadInitiate()	Bekanntmachen, welches Register mit welcher Größe beschrieben werden soll
CO_SDOclientDownloadBufWrite()	Legt die Variable mit dem Wert, welcher in das Register geschrieben werden soll, in den Zwischenspeicher des CANopen Stacks
CO_SDOclientDownload()	Führt den Speichervorgang in den gewünschten Bus-Teilnehmer aus
CO_SDOclientUpload()	Führt den Lesevorgang aus dem gewünschten Bus-Teilnehmer aus

Tabelle A.2: Beschreibung der essentiellen Methoden zur Konfiguration der Fahrmotoren via SDO

A.2.1 Klassendiagramme

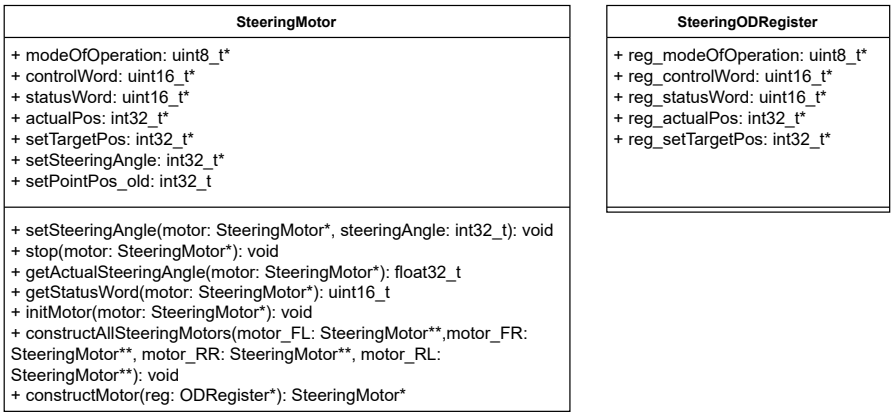

Abbildung A.2: Klassendiagramm der Lenkmotoren

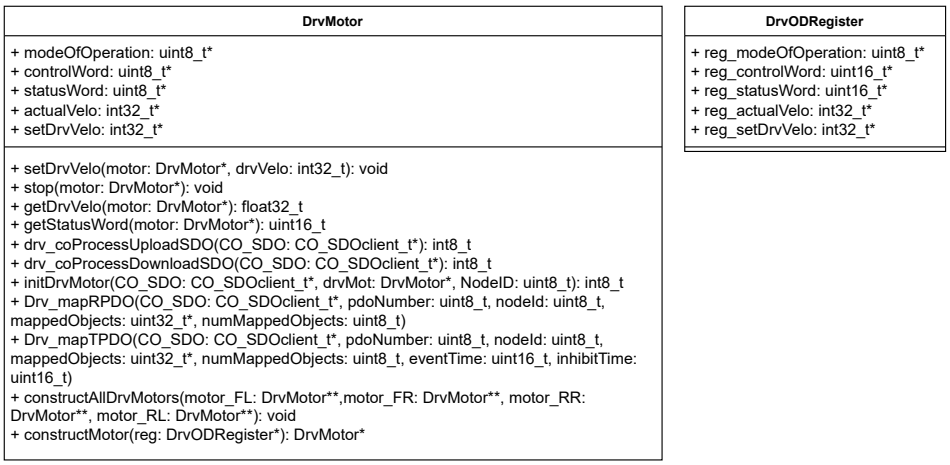

Abbildung A.3: Klassendiagramm der Fahrmotoren. Die Methoden Drv_mapRPDO, Drv_mapTPDO sind für das Konfigurieren der Motor PDOs zuständig.

A.2.2 Besonderheiten des Objectdictionary Editors

```
    .o_1F81_slaveAssignment = {
      ... .
    .dataElementSizeof = sizeof(uint32_t)
    },
    .o_1F82_requestNMT = {
      ...
    .dataElementSizeof = sizeof(uint8_t)
    },
```

Codeauszug A.1: Korrektur des Exportfehlers im Register 1F81 und 1F82

Beim Export der *OD.h* fehlen außerdem noch die in Codeauszug A.2 gezeigten Defines.

```
  // Sizes of OD arrays
  #define OD_CNT_ARR_1003 16
  #define OD_CNT_ARR_1010 4
  #define OD_CNT_ARR_1011 4
  #define OD_CNT_ARR_1016 8
  #define OD_CNT_ARR_1F81 127
  #define OD_CNT_ARR_1F82 127
```

Codeauszug A.2: Nachtragung von Defines

A.2.3 Aufbau des seriellen Strings

```
String outPutString = String(getActualSteeringAngle(SteerMotor_FL) +
    "," + getActualSteeringAngle(SteerMotor_FR) + "," +
   getActualSteeringAngle(SteerMotor_RR) + "," +
   getActualSteeringAngle(SteerMotor_RL) + "," + getDrvVelo(
   DrvMotor_FL)) + "," + getDrvVelo(DrvMotor_FR) + "," + getDrvVelo
   (DrvMotor_RR) + "," + getDrvVelo(DrvMotor_RL) + "\n";
```

Codeauszug A.3: Beispielhafter Aufbau des Strings

A.3 PCAN-View PDO Konfiguration

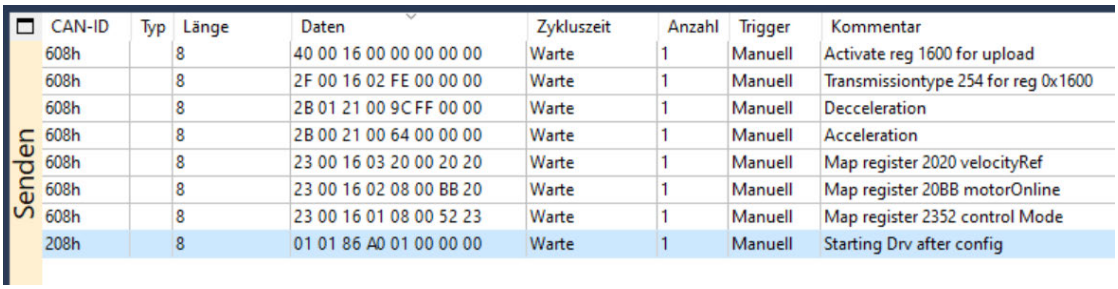

Abbildung A.4: Händische Konfiguration des Receive-PDOs mit PCAN-View

Abbildung A.5: Händische Konfiguration des Transmit-PDOs mit PCAN-View

A.4 Namenssyntax für die Erstellung des Objectdictionaries

Nabenmotor			
i = Node-ID 5-8	x = f/r f = front r = rear	y = r/l r = right l = left	
Register	Bezeichnung	Datentyp	PDO-Register
0x6i00	xy_SetDrvMotorOnline	UINT8	TXPDO 0x1Axx
0x6i01	xy_SetDrvControlMode	UINT8	
0x6i02	xy_SetDrvVelocityRef	INT32	
0x6i20	xy_GetDrvStatus	UINT16	RXPDO 0x18xx
0x6i21	xy_GetDrvVeloFdb	INT32	

Tabelle A.3: Namenssyntax des Objectdictionaries für die Nabenmotoren

Lenkmotor			
i = Node-ID 1-4	x = f/r f = front r = rear	y = r/l r = right l = left	
Register	Bezeichnung	Datentyp	PDO-Register
0x6i00	xy_SetSteerControlWord	UINT16	TXPDO 0x1A00.. 0x1Axx
0x6i01	xy_SetSteerOperationMode	UINT8	
0x6i02	xy_SetSteerTargetPos	INT32	
0x6i03	xy_SetSteerProfileVelo	UINT32	
0x6i04	xy_SetSteerTargetVelo	INT16	
0x6i20	xy_GetSteerStatusWord	UINT16	RXPDO 0x1800.. 0x18xx
0x6i21	xy_GetSteerOperationMode	UINT8	
0x6i22	xy_GetSteerActualPos	INT32	

Tabelle A.4: Namenssyntax des Objectdictionaries für die Lenkmotoren

A.5 Objectdictionary Editor

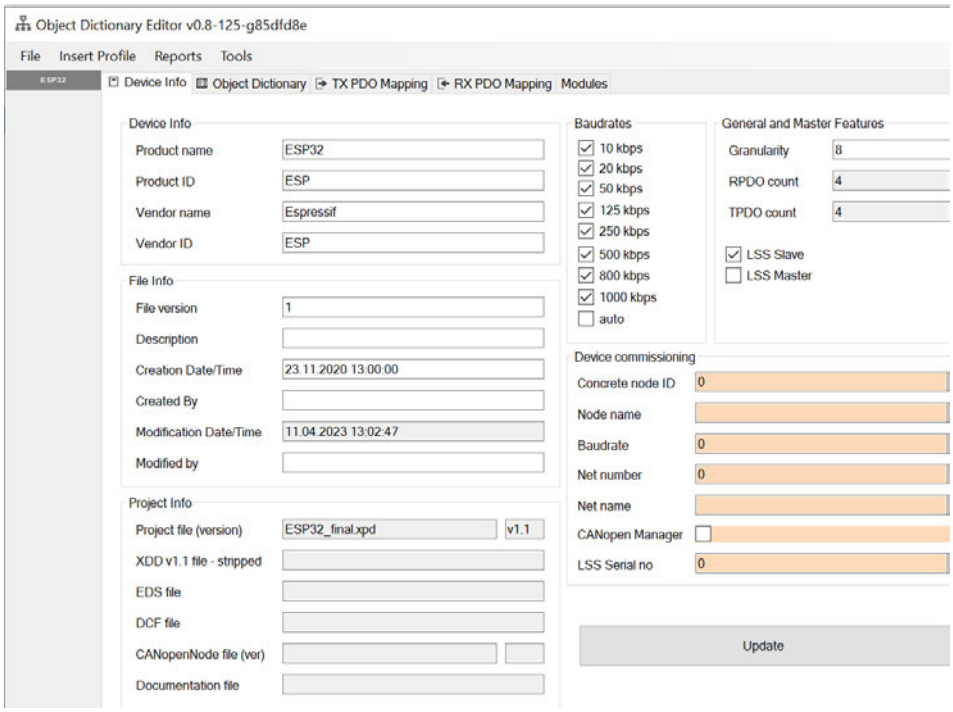

Abbildung A.6: Anwendung des Objectdictionary Editors. Hier ist das Hauptfenster zu sehen um allgemeine Geräteeinstellungen vorzunehmen.

Abbildung A.7: Anwendung des Objectdictionary Editors. Hier ist das Konfigurationsfenster für die Receive-PDOs zu erkennen

Abbildung A.8: Anwendung des Objectdictionary Editors. Hier ist das Konfigurationsfenster für die Transmit-PDOs zu erkennen

A.6 ROS2-Nodes

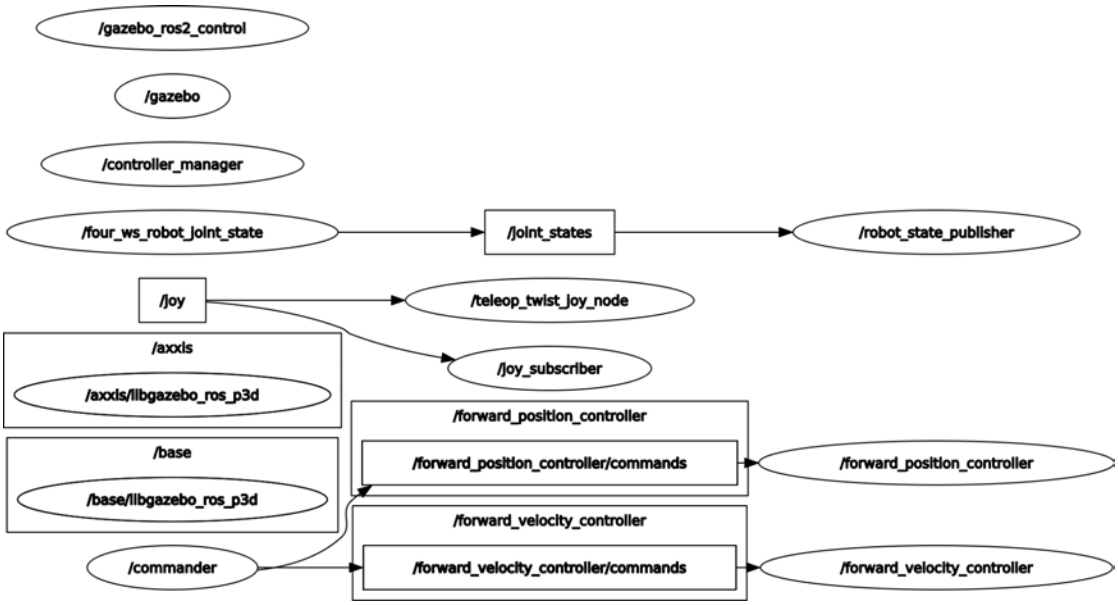

Abbildung A.9: Darstellung aller verwendeten Nodes in der Gazebo Simulation

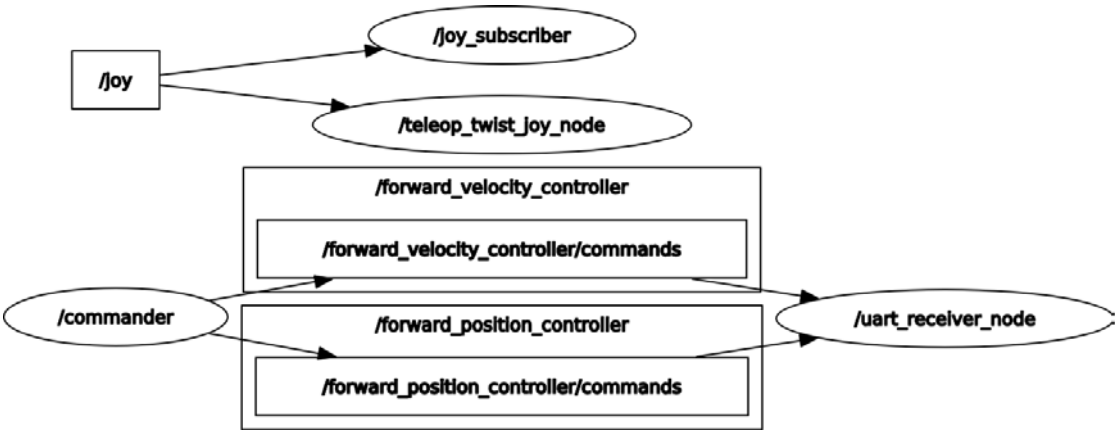

Abbildung A.10: Darstellung aller verwendeten Nodes in der realen Roboterbasis

A.7 Inhalt der CD

- Dieses Dokument als PDF
- **Ordner CanOpen_esp32:**
  Arbeitsverzeichnis der Low-Level-Implementierung für das MicroGiant-Board
- **kribl_4ws:**
  ROS2 Arbeitsverzeichnis mit Simulation und Fahrsteuerung
- **Software:**
  Ordner mit verwendeter Software with CANopenEditor und Herstellersoftware der Fahrmotoren
- **Manuals:**
  Handbücher für CANopen, der Fahr- und Lenkmotoren, des MicroGiant-Boards und des Jetson Nanos

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Dieses Blatt, mit der folgenden Erklärung, ist nach Fertigstellung der Abschlussarbeit durch den Studierenden auszufüllen und jeweils mit Originalunterschrift als letztes Blatt in das Prüfungsexemplar der Abschlussarbeit einzubinden.
Eine unrichtig abgegebene Erklärung kann -auch nachträglich- zur Ungültigkeit des Studienabschlusses führen.

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: Nickel Voelger

Vorname: Wolf Dieter

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit mit dem Thema:

Entwicklung einer Low-Level Fahrsteuerung zur Vermeidung von Schäden am Untergrund für mobile allradgelenkte Robotersysteme

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

Ort Datum Unterschrift