Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Tim Alcantara Ortega

Plattform-Adaption für das ROS Framework

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Tim Alcantara Ortega

Plattform-Adaption für das ROS Framework

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Martin Becke
Zweitgutachter: Prof. Dr. Franz Korf

Eingereicht am: 09.06.2022

Tim Alcantara Ortega

Thema der Arbeit

Plattform-Adaption für das ROS Framework

Stichworte

ROS, Raspberry Pi, API, C++, Ubuntu, 2-Phasen-Schrittmotoren

Kurzzusammenfassung

Im Rahmen dieser Bachelorthesis wird eine API für einen mobilen Roboter entwickelt. Mit dieser API wird es möglich sein, sämtliche Sensoren und Aktoren anzusteuern. Es wird auf die verbaute Hardware und verwendete Software eingegangen. Basierend auf der API soll die Steuerung des Roboters erfolgen und mithilfe der Odometrie soll die Position des Roboters festgestellt werden. Um die Funktion der API zu testen und zur Ermittlung der Genauigkeit der odometriebasierten Positionsbestimmung, werden mehrere Experimente durchgeführt und ausgewertet. Abschließend wird ein Ausblick für weitere Projekte geschaffen.

Tim Alcantara Ortega

Title of Thesis

Plattform-Adaption für das ROS Framework

Keywords

ROS, Raspberry Pi, API, C++, Ubuntu, 2-phase stepper motor

Abstract

Within the scope of this bachelor thesis, an API for a mobile robot will be developed. With this API it will be possible to control all sensors and actuators. The hardware and software used will be discussed. Based on the API, the robot will be controlled and the position of the robot will be determined with the help of odometry. To test the function of the API and to determine the accuracy of the odometry-based position determination, several experiments will be conducted and evaluated. Finally, an outlook for further projects is given.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Abkürzungsverzeichnis

API	Application Programming Interface
DIY	Do It Yourself
GHz	Gigahertz
GPIO	General Purpose Input/Output
HDMI	High Definition Multimedia Interface
LTS	Long Term Support (Langzeitunterstützung)
ROS	Robot Operating System
USB	Universal Serial Bus

1 Einleitung

Der Einsatz von Industrierobotern in Fabriken auf der ganzen Welt nimmt rasant zu. In immer mehr Bereichen werden Roboter eingesetzt, in den letzten fünf Jahren hat sich die Anzahl der eingesetzten Roboter in der Industrie fast verdoppelt [1]

Auch in privaten Haushalten wird der Einsatz von Robotern immer attraktiver. Staubsaugroboter fahren mittlerweile völlig autonom in der Wohnung, können Hindernisse erkennen und wenn der Akku fast leer ist, auch allein wieder zur Ladestation fahren. [2]

Die Hardware für solche Roboter wird immer günstiger und auch zuverlässiger. [3] Ein Raspberry Pi für die Steuerung ist heutzutage für 45,99€ erhältlich. [4] Dadurch ist es mehr Menschen möglich Software für Roboter zu entwickeln und es wird immer komplexere Software geschrieben, da diese aufeinander aufbauen kann. Hierdurch ist es möglich, dass viele Tätigkeiten in Kombination mit Robotern oder gar vollständig von Robotern ausgeübt werden. [5]

1.1 Motivation

Die Motivation dieser Bachelorthesis ist es, den so genannten „Loomos Little-Helper“ weiterzuentwickeln. Der „Loomos Little-Helper“ ist ein von voran gegangen Studierenden entwickelter[1] kleiner Roboter, der später andere Roboter, sowie den „Loomo“ unterstützen soll. Der „Loomos Little-Helper“ besteht aus zwei Antriebsmotoren, einem Multifunktionsarm und zwei Ultraschall-Sensoren, die vorne und hinten angebracht sind. Gesteuert wird der „Loomos Little-Helper“ von einem Einplatinenrechner, dem Raspberry Pi. Ein Vorteil des Raspberry Pi´s ist es, dass er leicht durch weitere Komponenten, wie Sensoren, erweitert werden kann.

Mit Open Source Frameworks wie ROS (Robot Operating System) wird es Softwareentwicklern einfacher gemacht, einzelne Softwarekomponenten wiederzuverwenden und somit wenig Aufwand beim Entwickeln neuer Programme zu haben. [6]

[1] GitLab - https://git.haw-hamburg.de/smart4cads/Multifunktions-Werkzeug

Durch die Modularität von ROS können hier Erweiterungen durch bereits bestehende Softwarelösungen durchgeführt werden. In vielen Fällen ist es möglich, mit wenig Zeitaufwand und kleinen Anpassungen, weitere Funktionen hinzuzufügen.

In der folgenden Arbeit wird auf die Adaption und Nutzung von ROS eingegangen.

1.1 Zielsetzung

Ziel dieser Arbeit ist es, den „Loomos Little-Helper“ um eine ROS Schnittstelle weiterzuentwickeln. Hierfür wird eine API (*Application Programming Interface*) benötigt, die zum Steuern des Roboters und zum Auslesen der Sensoren genutzt wird. Die Realisierung der API und der Softwarefunktionen des „Loomos Little-Helpers“ erfolgt mittels ROS, da bereits vorhandene ROS-Module zum Steuern des „Loomos Little-Helpers“ vorliegen. Somit können diese Module mit geringen Anpassungen übernommen werden.

Mit Hilfe von Experimenten soll ermittelt werden, wie die Genauigkeit der odometriebasierten Positionsbestimmung ist. Hierfür ist zusätzlich die Berechnung für die Odometrie zu implementieren. Des Weiteren muss die Ansteuerung der Schrittmotoren über die API erarbeitet werden. Die Schrittmotoren werden sowohl für die Fortbewegung als auch für die Ausrichtung des Multifunktionsarmes verwendet. Zusätzlich zu den Schrittmotoren sollen die bereits montierten Ultraschall-Sensoren durch die API nutzbar gemacht werden.

Zur Messung der Odometriefehler werden mehrere Experimente bei verschiedenen Geschwindigkeiten durchgeführt.

Es sollen reproduzierbare und aussagekräftige Ergebnisse für die Ansteuerung des „Loomos Little Helpers“ mittels ROS erzeugt werden. Dieser soll mit den bestehenden Programmen kommunizieren. Hierbei wird der Raspberry Pi des Roboters zum *ROS-Master* und stellt über sogenannte *Nodes* die verschiedenen Services zu Verfügung.

Anschließend sollen die Ergebnisse gegenübergestellt und ausgewertet werden.

1.2 Gliederung der Arbeit

Die Arbeit ist in folgende Abschnitte unterteilt:

Kapitel 2 – Grundlagen

Der „Loomos Little Helpers“ wird inklusive seiner Aktoren und Sensoren, sowie seiner Funktionen vorgestellt. Die odometriebasierte Positionsbestimmung und das eingesetzte *Framework Robot Operation System (ROS)*, welches für die Steuerung des Roboters eingesetzt wird, werden erklärt.

Kapitel 3 - Programmierung

In diesem Kapitel wird auf die Programmierung der einzelnen *ROS-Nodes* und auf ihre Funktionen eingegangen. Des Weiteren wird die Ansteuerung der Hardware beschrieben.

Kapitel 4 - Experimente

Der Aufbau der Experimente und deren Durchführung wird aufgeführt. Die Ergebnisse aus den Experimenten werden grafisch veranschaulicht und beschrieben.

Kapitel 5 – Betrachtung der Ergebnisse

Die Ergebnisse der vorangegangenen Experimente werden noch mal zusammengefasst und betrachtet. Aus den betrachteten Ergebnissen werden mögliche Fehlerquellen aufgezeigt und erläutert.

Kapitel 6 – Fazit und Ausblick

Zusammenfassung der erfolgten Arbeit und das daraus resultierende Fazit mit Ausblick auf mögliche Erweiterungen und Verbesserungsideen.

2 Grundlagen

Im Folgenden wird ein Überblick des Systems gegeben und der Roboter vorgestellt. Dieser wird im weiteren Verlauf der Arbeit auch „Loomos Little Helper“ genannt. Hierbei wird auf die verbauten Aktoren und Sensoren eingegangen. Des Weiteren werden die theoretischen Grundlagen betrachtet, die zum Verständnis und zur Durchführung der Experimente notwendig sind.

2.1 Übersicht des Systems

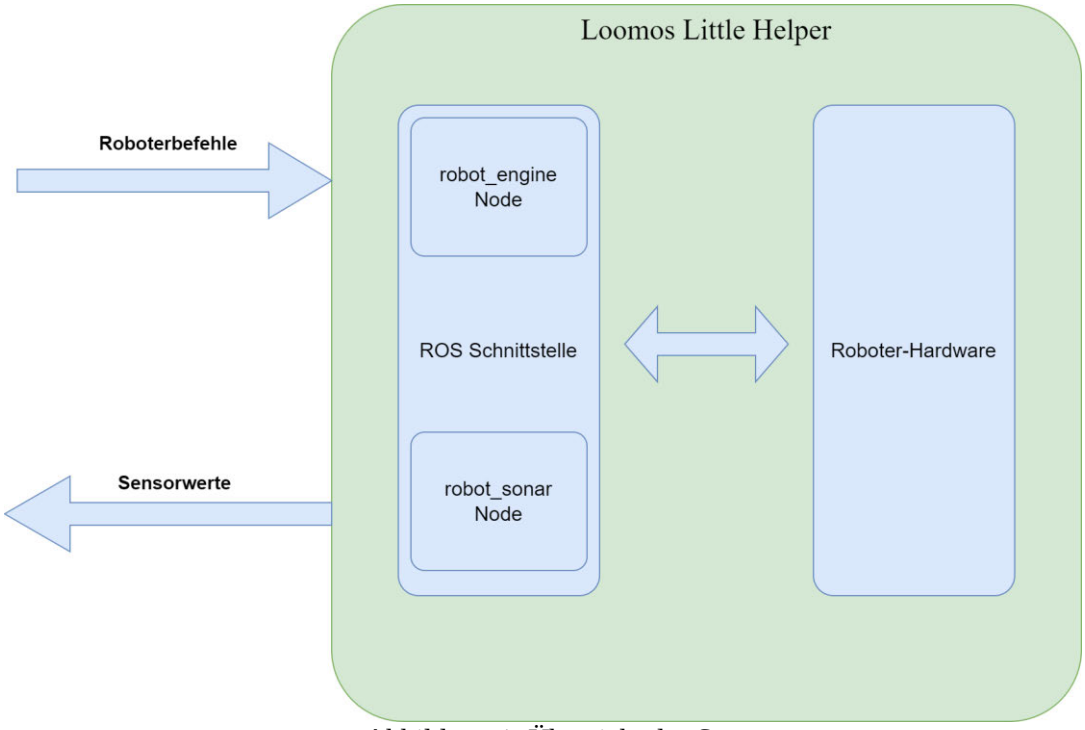

Abbildung 1: Übersicht des Systems

In Abbildung 1 ist der Überblick des Systems zu sehen. Die zu implementierende ROS Schnittstelle ist für die Kommunikation mit dem „Loomos Little Helper“ zuständig. Die Schnittstelle ist in zwei *Nodes* aufgeteilt. Die *robot_engine Node* ist fürs Empfangen der Roboterbefehle zuständig und verarbeitet diese und leitet diese an die entsprechende Roboter-Hardware weiter. Die *robot_sonar Node* liefert die Sensorwerte.

2.2 Roboter Hardware

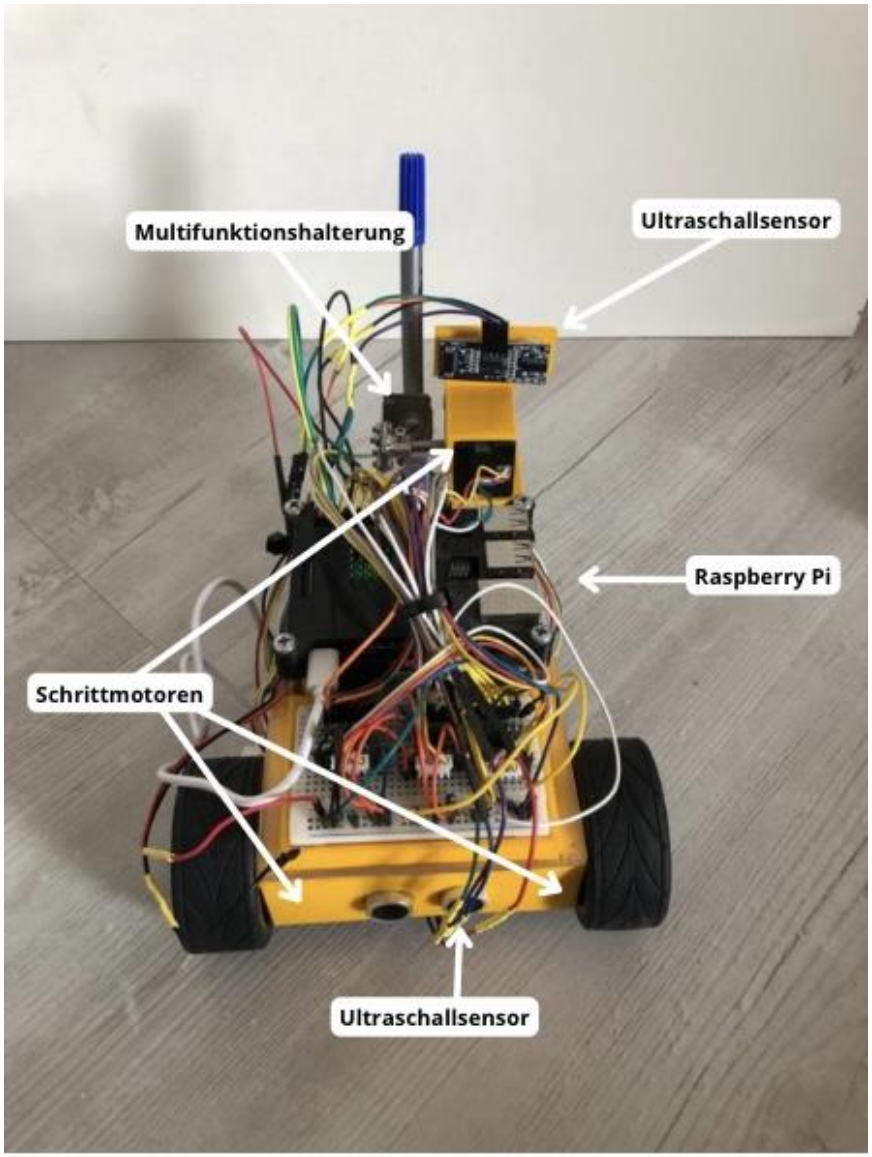

Abbildung 2: „Loomos Little Helper“

In der Abbildung 2 ist der „Loomos Little Helper“ zu sehen. Er verfügt über zwei Ultraschallsensoren, die jeweils vorne und hinten angebracht sind. Ein Roboterarm, der analog zu den beiden Hinterrädern, von jeweils einem Schrittmotor angetrieben wird. Gesteuert werden die Komponenten von einem Einplatinenrechner, dem Raspberry Pi.

2.2 Einplatinenrechner Raspberry Pi

Der Raspberry Pi ist ein sehr beliebter Einplatinenrechner. Grund dafür sind die vielen positiven Eigenschaften: geringe Anschaffungskosten, ausreichend leistungsfähig, viele Anschlüsse. Seit dem ersten Erscheinen des Raspberry Pi‘s wurden bereits mehr als 40 Millionen Stück verkauft. Besonders oft wird der Raspberry Pi für *Do It Yourself (DIY)* / Heimwerker Projekte eingesetzt. Auch für die Industrie wird der Raspberry Pi immer interessanter. Durch seine kompakte Bauweise kann er an Orten eingesetzt werden, an denen wenig Platz zur Verfügung steht, wie zum Beispiel Schaltschränken. [7]

Im „Loomos Little Helper“ wurde ein Raspberry Pi 3 Model B verbaut. Das Model B besitzt einen 1,2 GHz Quad Core Prozessor und 1 GB Arbeitsspeicher. Des Weiteren bietet der Raspberry Pi zahlreiche Anschlüsse für externe Hardware, darunter vier USB-Ports, HDMI, Ethernet und 40 GPIO Pins. Die GPIO Pins ermöglichen den Anschluss vieler Sensoren und Aktoren. Weitere Informationen können der Hersteller Webseite[2] entnommen werden.

2.3 Aktoren

Im Falle des „Loomos Little Helpers“ bilden die Aktoren den Antrieb für die Bewegung des Roboters in seiner Umgebung und den Arm, der für verschiedene Einsatzmöglichkeiten (Schaufel, Stift und Pieker) umgerüstet werden kann.

2.3.1 Antrieb

Der Antrieb des „Loomos Little Helper“ besteht aus zwei Schrittmotoren (Herst.-Nr:ST4118X1404-B) vom Hersteller Nanotec. Die beiden Schrittmotoren treiben unabhängig voneinander je ein Antriebsrad an. Die Antriebsachse liegt im Heck des Roboters, in der Front des Roboters befindet sich eine Lenkrolle. Durch den verschobenen Drehpunkt ist eine Drehung um den geometrischen Mittelpunkt des Roboters nicht möglich, sondern nur um die Antriebsachse.

[2] Raspberry Pi - https://www.raspberrypi.com/products/raspberry-pi-3-model-b/

Die Schrittmotoren haben einen Schrittwinkel von 1,8 Grad, mit dem sich der Roboter präzise bewegen kann. Weitere Daten können dem Datenblatt[3] entnommen werden.

2.3.2 Multifunktionshalterung

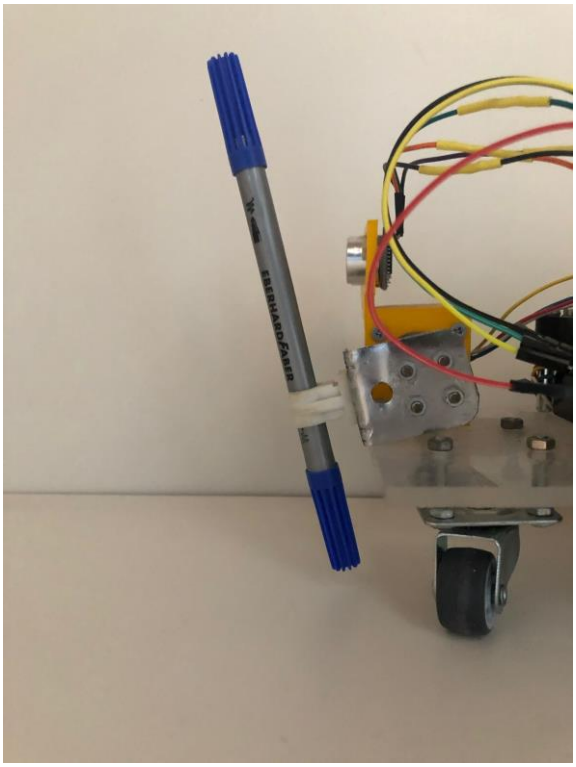

Abbildung 3: Multifunktionshalterung

Die Multifunktionshalterung des „Loomos Little Helper“ ist an der Front befestigt und wird, wie der Antrieb, von dem gleichen Typ Schrittmotor angetrieben. Die Multifunktionshalterung kann sich somit nur auf- und abwärts bewegen und wird dadurch im Winkel von 30 Grad verstellbar. Der Aufsatz der Multifunktionshalterung kann ausgewechselt werden. In der Abbildung 3 ist die Multifunktionshalterung, ausgerüstet mit einem Stift, zu sehen.

[3] Nanotec Schrittmotor Datenblatt - https://de.nanotec.com/fileadmin/files/Datenblaetter/Schrittmotoren/ST4118/X/ST4118X1404-B.pdf

2.3.3 Schrittmotortreiber

Der DRV8833 Dual H-Bridge Motor Driver[4] ist eine Platine, welche zur Ansteuerung von Schrittmotoren benutzt wird. Der DRV8833 kann sowohl im einphasigen Vollschritt (*Full Step*), zweiphasigen Vollschritt, sowie im Halbschritt-Modus betrieben werden. Im nachfolgenden werden alle drei Modi erklärt.

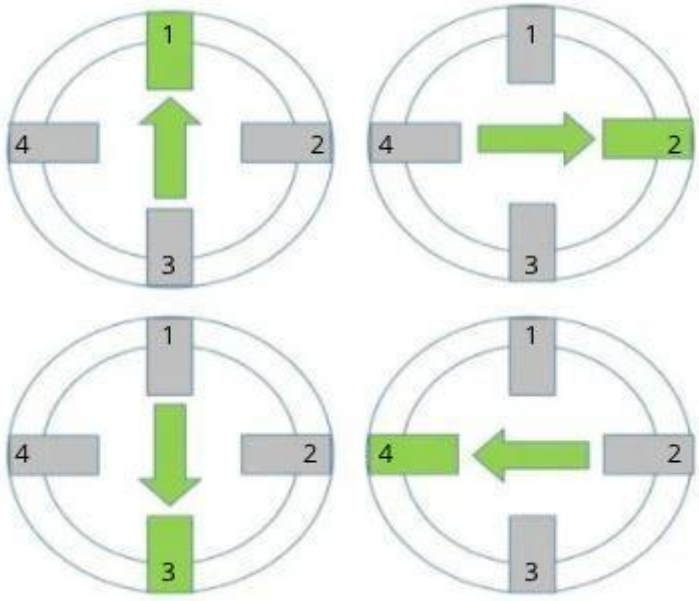

Abbildung 4: Funktionsweise einphasiger Full Step Modus (GreigRS, www.rs-online.com)

In Abbildung 4 ist der einphasige Vollschritt-Modus zu sehen. Hierbei wird immer nur eine Phase unter Spannung gesetzt. Dadurch wird ein Magnetfeld erzeugt, wodurch sich das Rad bewegt. Im ersten Schritt Phase 1, im zweiten Schritt Phase 2, im dritten Schritt Phase 3 und im vierten schritt Phase 4. Diese Reihenfolge wird kontinuierlich wiederholt, wodurch die Rotation des Motors entsteht. Um den Motor in die entgegengesetzte Richtung zu drehen, wird die Reihenfolge umgekehrt. Wenn Phase 1 startet, muss danach Phase 4 unter Spannung gesetzt

[4] DRV8833 Motor Driver Datenenblatt - https://www.ti.com/lit/ds/symlink/drv8833.pdf

werden, danach Phase 3 und zuletzt noch Phase 2. Durch die Wiederholung dieser Schritte entsteht die Rotation. Dieser Modus ist von allen Modi, der mit dem niedrigsten Treiber-Energieverbrauch.

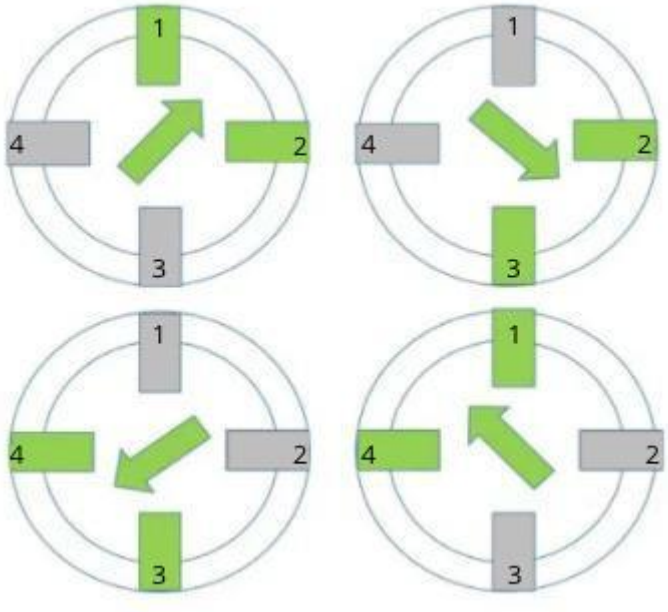

Abbildung 5: Funktionsweise 2-phasiger Full Step Modus (GreigRS, www.rs-online.com)

In Abbildung 5 ist der zweiphasige Vollschritt-Modus zu sehen. Im Gegensatz zum einphasigen Vollschritt-Modus werden hier immer zwei Phasen gleichzeitig unter Spannung gesetzt. Im ersten Schritt wird, anstatt nur Phase 1 unter Spannung zu setzen auch noch Phase 2 unter Spannung gesetzt. Im zweiten Schritt werden dann Phase 2 und 3, im dritten Schritt Phase 3 und 4 und im letzten Schritt Phase 4 und 1 unter Spannung gesetzt. Auch hier kann, wie im ersten Modus, durch Änderung der Reihenfolge die Drehrichtung geändert werden. In diesem Modus stehen zwei Phasen gleichzeitig unter Spannung wodurch das Drehmoment, als auch der Energieverbrauch erhöht wird.

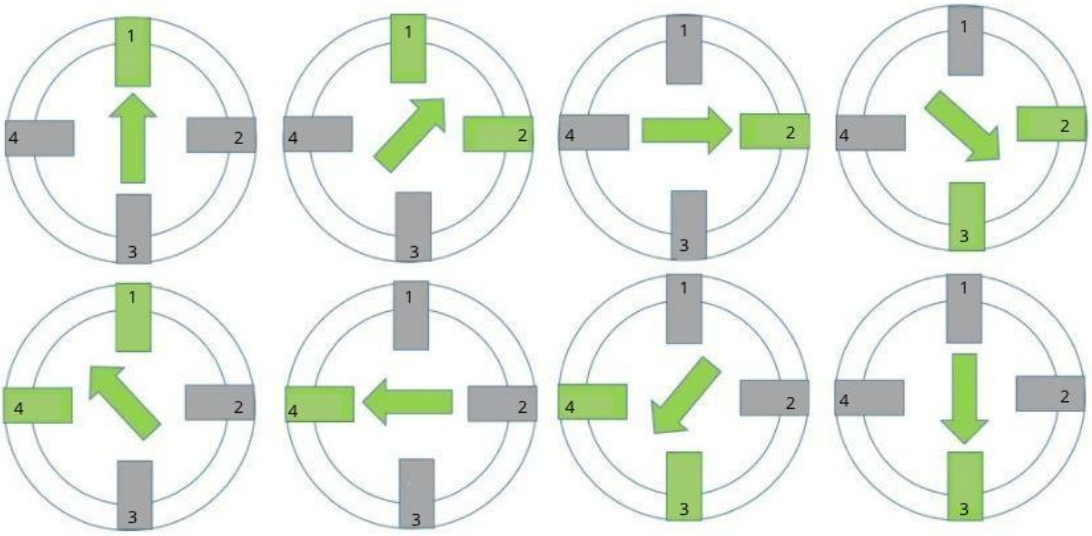

Abbildung 6: Funktionsweise Halbschritt-Modus (GreigRS, www.rs-online.com)

In Abbildung 6 ist der Halbschritt-Modus zusehen, der eine Mischung aus den beiden vorigen Modi ist. Durch die Mischung der beiden vorigen Modi sind jetzt doppelt so viele Schritte möglich. Hier wird im ersten Schritt nur Phase 1 unter Spannung gesetzt und im zweiten Schritt Phase 1 und 2. Im dritten Schritt dann wieder nur eine Phase (Phase 2), für die restlichen Schritte folgt der Modus dem vorangegangenen Muster. Durch diesen Modus werden die Schrittwinkel des Motors halbiert, was im Vergleich zum Vollschrittmodus, einen Drehmomentverlust bewirkt. [8]

In Abschnitt 3.2 wird nochmals auf die Implementierung eingegangen.

Die vorgestellten Funktionen sind für den Schrittmotor ST4118X1404 und den vorgesehenen Einsatzbereich des „Loomos Little Helper“ geeignet.

2.3.4 Sensoren

Sensoren spielen in der Robotik eine große Rolle, denn sie ermöglichen die Interaktion von Robotern mit ihrer Umwelt.

Der „Loomos Little Helper“ besitzt zwei Ultraschall-Abstandssensoren, wovon einer vorne und einer hinten installiert sind. Die Ultraschall-Abstandssensoren erkennen Hindernisse vor und hinter dem „Loomos Little Helper“. Bei den Ultraschall-Abstandssensoren handelt es sich um Joy-IT Ultraschall-Abstandssensoren HC-SR04. Die Abstandssensoren können Gegenstände im Bereich von 3-400 cm erfassen[5]. Mit Hilfe der Sensoren kann der Roboter sich zwar noch nicht im Raum orientieren, dafür werden Kollisionen und eventuelle Schäden verhindert.

Im Abschnitt 3.3 wird die Implementierung der Sensoren ausführlich erläutert.

2.4 Odometrie

Ein wichtiger Teil dieser Arbeit ist die Odometrie. Mit der odometriebasierten Positionsbestimmung lässt sich die Position anhand von Wegmessungen errechnen. Diese Methode hat sich in der Praxis, insbesondere für kurze Strecken, etabliert. [9] Mit Hilfe der Schrittmotoren kann genau bestimmt werden, wie viele Umdrehungen die Räder jeweils zurücklegen. Um die neue Position zu berechnen, wird abschließend nur noch der Radumfang und der Radabstand benötigt.

Diese Methode der Positionsbestimmung ist lediglich für kurze Strecken geeignet, da mögliche Fehler sich aufsummieren und es keinerlei Korrektur gibt. [9] Wie genau die odometriebasierte Positionsbestimmung für den „Loomos Little Helper“ funktioniert, zeigt sich in den Experimenten in Kapitel 4. Des Weiteren wird im Kapitel 5.1 zusätzlich auf mögliche Fehlerquellen eingegangen.

[5] Joy-IT Ultraschall Abstandssensor HC-SR04 Datenblatt

2.5 Robot Operating System

Die Entwicklung neuer Roboter ist oft mit viel Aufwand verbunden, da Hardware- und Softwarekomponenten integriert werden müssen. Meistens unterscheiden sich die Roboter nicht massiv voneinander, da viele Funktionen und Hardwarekomponenten identisch sind. [6]

Unter diesem Aspekt ist es erstrebenswert, die bereits entwickelten Softwarelösungen wieder zu verwenden. Dadurch kann auf die teure Entwicklung und Testphasen, welche sehr zeitaufwendig sind, verzichtet werden. Somit wird auch der finanzielle Aspekt berücksichtig, welcher in der Softwareentwicklung heutzutage immer mehr an Bedeutung gewinnt. [6]

Das Robot Operating System (ROS) ist ein Framework, das genau diesen Ansatz verfolgt. Es ist auf dem Markt weit verbreitet und das Interesse vieler Unternehmen daran steigt [10].

Alternativ zu ROS könnte auch der Nachfolger ROS2 verwendet werden. ROS2 hat mehrere Vorteile gegenüber ROS, wie zum Beispiel die Unterstützung von Echtzeit-Systemen und der Einsatz bei instabilen Netzwerken. [11]

Die Tatsache, dass der „Loomos Little Helper“ mit dem bereits vorhandenen „Loomo“ (der über ROS kommuniziert) interagieren soll, spricht für die Verwendung von ROS.

Bei ROS handelt es sich um ein *open-source Framework* für Roboter. Die Entwicklung begann 2007 am *Stanford Artificial Intelligence Laboratory* im Zuge des Projekts *STAIR* [12] . Ab 2009 wurde es durch das Robotikunternehmen *Willow Garage* weiterentwickelt. Seit April 2012 wird ROS von der Organisation *Open Source Robotics Foundation* (OSRF) [13]unterstützt.

Im Folgenden werden die Eigenschaften von ROS und die ROS-Kommunikation näher erläutert. Im Anhang befindet sich zusätzlich eine Installationsanleitung für ROS.

2.5.1 ROS Eigenschaften

ROS ist, wie der Name vermuten lässt, kein richtiges Betriebssystem. Offiziell wird es als „*meta-operating system*“ beschrieben. Es bietet viele Dienste, die von einem Betriebssystem erwartet werden, einschließlich Hardware-Abstraktion, Low-Level-Gerätesteuerung, Implementierung häufig verwendeter Funktionen, *Message-Passing* zwischen Prozessen und die Paketverwaltung. [14]

Außerdem bietet ROS auch Funktionen zum *Debuggen* von Programmen. ROS kann hierfür Sensordaten oder andere Nachrichten aufzeichnen und diese wieder abspielen. Das ermöglicht das *Debuggen* von Programmen, ohne die Experimente zu wiederholen. [15]

2.5.2 ROS-Kommunikation

Eins der Ziele von ROS ist es eine Sammlung von kleinen, möglichst eigenständigen Programmen zu erstellen. Diese Programme werden *Nodes* genannt. *Nodes* stellen verschiedene *Services* und *Topics* zur Verfügung, über die die Kommunikation stattfindet. Weil *Nodes* auch untereinander kommunizieren müssen, organisiert der zentrale ROS-Master die Kommunikation der einzelnen *Nodes* untereinander. [15]

2.5.3 ROS-Node

Nodes sind Prozesse, die einzelne Funktionen kapseln. Für die Kommunikation untereinander gibt es sogenannte *Topics* und *Services*.

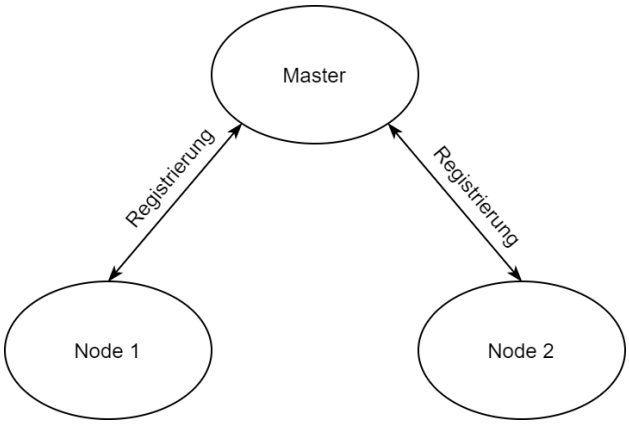

Abbildung 7: ROS Registrierung von Nodes

Damit die *Nodes* an dieser Kommunikation teilnehmen können, müssen diese sich beim *ROS-Master* registrieren und die angebotenen *Topics* und *Services* übermitteln. Der *ROS-Master* speichert die Registrierung (siehe Abbildung 7) und agiert als Vermittler zwischen den *Nodes*. Sobald die *Nodes* vom *Master* vermittelt wurden, findet die Kommunikation direkt zwischen den *Nodes* statt. [15]

2.5.4 ROS-Topic

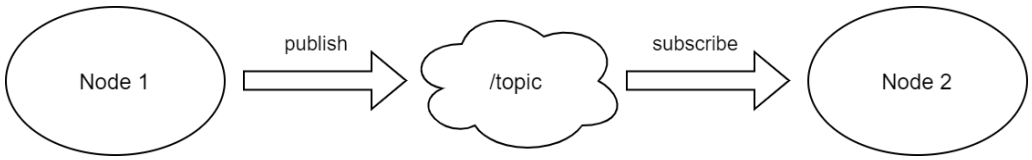

Abbildung 8: ROS Topic

Mit Hilfe von *Topics*, namentlich identifizierten Datenkanälen, kommunizieren *Nodes* und tauschen untereinander Daten aus. Die sendende *Node* legt für jedes *Topic* einen festen Nachrichtentyp fest. Jede *Node* kann auf solche *Topics* Daten senden (*publishen*) und auch empfangen (*subscriben*) (siehe Abbildung 8). Hierbei ist es für die *Nodes* unerheblich, mit wem sie kommunizieren, sie sind lediglich an den Daten aus dem *Topic* interessiert. Die Kommunikation findet hierbei asynchron statt. [15]

2.5.5 ROS-Service

Im Gegensatz zu *Topics* erfolgt die Kommunikation bei *Services* synchron. Ein *Service* besteht aus einem Nachrichtenpaar, einer Anfragenachricht (*request*) und einer Antwortnachricht (*reply*). Die *Node*, die einen *Service* aufruft, schickt somit einen *request* und wird so lange blockiert, bis die Anfrage bearbeitet wurde und ein *reply* gesendet wird. [15]

3 Programmierung

In diesem Abschnitt wird auf die Programmierung der einzelnen *ROS-Nodes*, die für die Ansteuerung der Schrittmotoren (robot_engine *Node*) und der Sensoren (robot_sonar *Node*) zuständig sind, eingegangen.

3.1 Aufbau der Programme

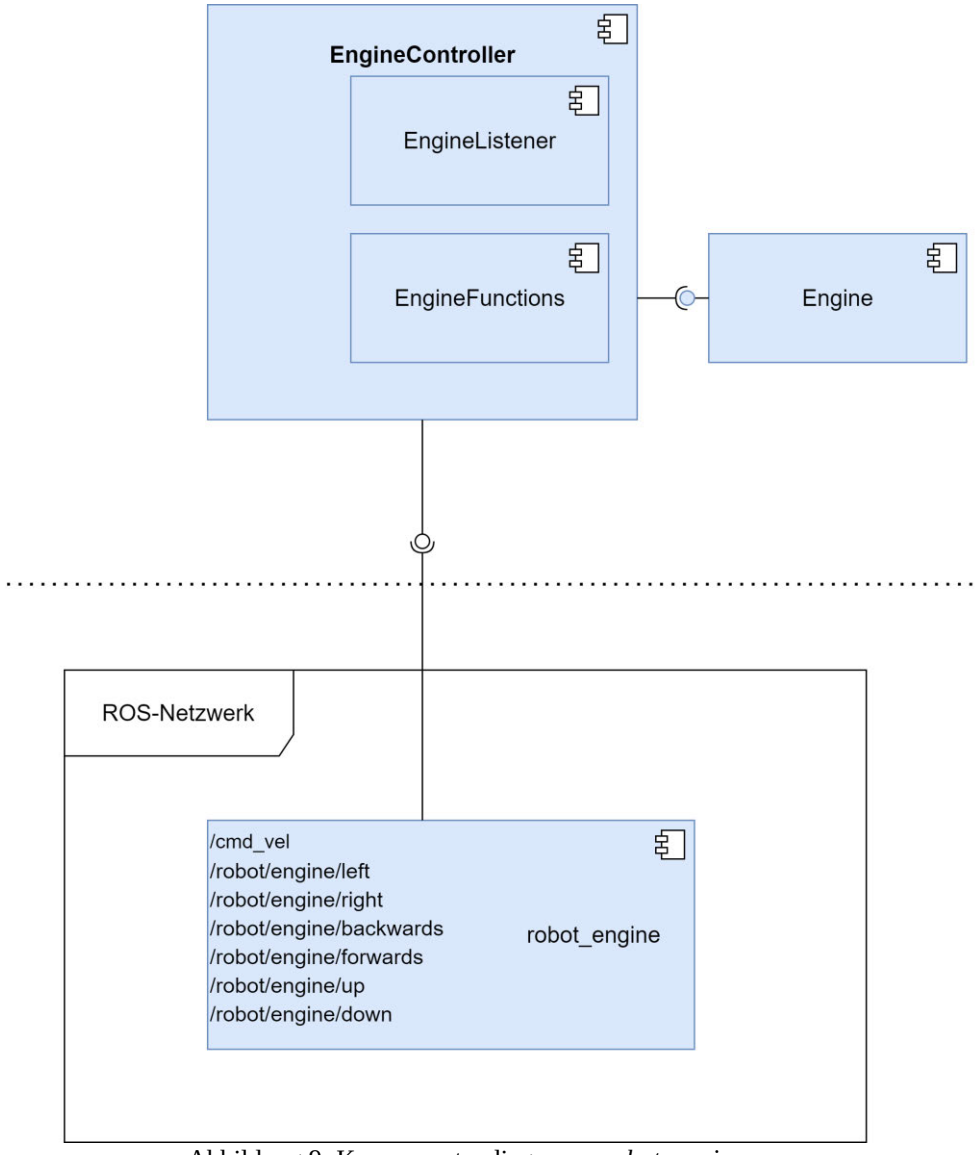

Abbildung 9: Komponentendiagramm *robot_engine*

In Abbildung 9 sind die einzelnen Komponenten der *robot_engine Node* zu sehen. Die beiden *Nodes* (*robot_sonar und robot_engine*) sind gleich aufgebaut.

Jede *Node* hat eine *Controller-*, *Listener-*, *Function-* und *Engine-* bzw. *Sensor*-Komponente. Die beiden *Nodes* sind komplett voneinander unabhängig und können auch einzeln gestartet werden. Der Controller der jeweiligen *Node* startet die einzelnen Komponenten, hierbei wird ROS initialisiert und der jeweilige *Listener* gestartet. Der *Listener subscribed* dann auf die für ihn wichtigen *Topics* und stellt gegebenenfalls *Services* zur Verfügung. Die ankommenden Daten werden an die *Functions*-Klasse weitergeleitet und dort verarbeitet. Die *Engine-* und die *Sensor-* Klasse sind für die direkte Ansteuerung der Hardware zuständig und werden in den nachfolgenden Abschnitten erklärt.

3.2 Ansteuerung der Schrittmotoren

Die Ansteuerung der Schrittmotoren wird von der *robot_engine Node* gesteuert.

Für die Hardwareansteuerung der Schrittmotoren ist die *Engine*-Klasse zuständig. Zum Initialisieren der Klasse werden die angeschlossenen GPIO-Pins benötigt. Vier der Pins werden für die einzelnen Schritte (*Steps*) benötigt, der fünfte Pin ist zum Einschalten des Schrittmotortreibers zuständig. Für die Implementierung wurde der *Full step* Modus ausgewählt, der mit einer Schrittweite von 0,11 cm arbeitet und damit das größte Drehmoment liefert.

Im *Listing* 1 sind die Funktionen für den 2-phasigen *Full step* Modus zu sehen. Hierfür wurde für jeden der vier *Steps* eine Funktion geschrieben. Mit dem Übergabeparameter *speed* wird die Geschwindigkeit des Motors übergeben. Die Funktion setGpio() setzt die jeweiligen GPIO-pins, um die Phasen anzusteuern.

```
void Engine::step1(int speed){
    setGpio(HIGH,LOW,HIGH,LOW);
    engineSpeed(speed);
}
void Engine::step2(int speed){
    setGpio(LOW,HIGH,HIGH,LOW);
    engineSpeed(speed);
}
void Engine::step3(int speed){
    setGpio(LOW,HIGH,LOW,HIGH);
    engineSpeed(speed);
}
void Engine::step4(int speed){
    setGpio(HIGH,LOW,LOW,HIGH);
    engineSpeed(speed);
}
```

Listing 1: Ausschnitt aus der Engine.cpp

Um einen Motor vorwärts drehen zu lassen, müssen die Funktionen *step* 1-4 in Reihenfolge aufgerufen werden und zum rückwärts drehen in entgegengesetzter Reihenfolge.

Den Aufruf der einzelnen *steps* übernimmt die Funktion *OneStep()*, siehe Listing 2, wodurch die Ansteuerung der Motoren vereinfacht wird. Durch Angabe der Richtung und des gespeicherten aktuellen *Steps* (in *currentStep*) kann der nächste *Step* ermittelt werden und mittels des *switch-case* gesetzt werden.

Durch die Abstraktion müssen lediglich die Geschwindigkeit und die Richtung, in der sich der Motor bewegen soll, angegeben werden.

Der Aufruf ROS_INFO gehört zum *logging* System von ROS. Hierbei gibt es verschiedene *Log Level*: DEBUG, INFO, WARN, ERROR und FATAL. Die Nachrichten werden über die Konsole ausgegeben und über das Topic /rosout *gepublished*. Standardmäßig ist das *Log Level* auf INFO gesetzt, was bedeutet, dass alle höheren Level ausgegeben werden, also alle bis auf DEBUG. Um das *Log Level* zu ändern, kann man über die Konsole folgenden Befehl ausführen:

```
rosservice call /node-name/set_logger_level ros.package-name level
```

```
void Engine::oneStep(bool direction, int speed){
    if(direction){
        if (currentStep>=4){
                currentStep = 1;
        } else {
                currentStep++;
        }
    } else{
        if (currentStep<=1){
                currentStep = 4;
        } else {
                currentStep--;
        }
    }

    switch(currentStep){
        case 1:
            step1(speed);
            break;
        case 2:
            step2(speed);
            break;
        case 3 :
            step3(speed);
            break;
        case 4 :
            step4(speed);
            break;
        default:
            ROS_INFO("Engine::oneStep switch: Wrong input");
    }
}
```

Listing 2: Ausschnitt aus der Engine.cpp oneStep()

3.3 Ansteuerung der Sensoren

Die Ansteuerung der Sensoren wird von der *robot_sonar Node* gesteuert. Die beiden Ultraschall-Sensoren werden über die Klasse *Sensor.cpp* angesprochen.

Für die Initialisierung der Klasse werden die angeschlossenen GPIO-Pins benötigt. Der Trigger-Pin wird zum Senden der Ultraschallimpulse angesteuert und der Echo-Pin erzeugt einen Impuls, sobald das reflektierte Signal empfangen wird.

Nachdem der *Service GetDistance* aufgerufen wird, wird eine Abstandsmessung gestartet. Hierzu wird ein *High-Impuls* auf den *Trigger-Pin* gegeben. Daraufhin sendet der Sensor acht Impulse mit je 40 kHz[6] aus. Sobald die Impulse gesendet wurden, wird der Echo-Pin auf „*High*" gesetzt und das Programm startet die Zeitmessung. Nachdem Ultraschallimpulse reflektiert und vom Sensor empfangen wurden stoppt die Zeitmessung. Mit der folgenden Formel kann daraus die Entfernung berechnet werden.

$$Entfernung = Geschwindigkeit * Zeit$$

Dadurch, dass der Ultraschallimpuls immer aus dem Weg hin zum Objekt und zurück zum Sensor besteht, muss die Formel noch angepasst werden. Um zu vermeiden, dass der doppelte Weg berechnet wird, ist die Formel wie folgt anzupassen:

$$Entfernung = \frac{Geschwindigkeit * Zeit}{2}$$

Aus der gemessenen Zeit wird die Strecke berechnet und der Wert zurückgegeben.

[6] Joy-IT Ultraschall Abstandssensor HC-SR04 Datenblatt

4 Experimente

In diesem Kapitel werden die Experimente ausgeführt und erläutert. Für die Vorbereitung wird beschrieben, welche Berechnungen der „Loomos Little Helper“ tätigt und wie der Versuchsaufbau gestaltet ist. Nachfolgend werden die Ergebnisse vorgestellt.

Durch diese Experimente soll festgestellt werden, wie präzise sich der „Loomos Little Helper“ mittels der Odometrie, welche in dem Programm verwendet wird, fortbewegen kann.

Der „Loomos Little Helper“ wird im ersten Experiment bei verschiedenen Geschwindigkeiten geradeaus fahren. Aufgrund räumlich begrenzter Umstände wurde eine Strecke von drei Metern gewählt. Dies ist die größtmögliche Distanz. Das Auftreten möglicher Fehler wird mit zunehmender Fahrzeit größer, was die Zuverlässigkeit der Ergebnisse steigert. Die zu fahrende Strecke und Geschwindigkeit wird mit ROS übermittelt, die tatsächlich gefahrene Strecke wird manuell nachgemessen.

Beim zweiten Experiment wird die Genauigkeit bei einer Rotation des „Loomos Little Helper“ überprüft. Eine Drehung um maximal 360 Grad ist ausreichend, damit der „Loomos Little Helper“ alle erforderlichen Aufgaben erfüllen kann. Eventuelle Abweichungen vom Sollwert werden manuell nachgemessen.

Jedes Experiment wird 5-mal wiederholt, um eine größtmögliche Variabilität von Ergebnissen zu erzielen.

4.1 Vorbereitung des ersten Experimentes

In diesem Experiment soll die Genauigkeit der odometriebasierten Positionsbestimmung beim geradeaus fahren unter verschiedenen Geschwindigkeiten bestimmt werden.

Der „Loomos Little Helper“ soll von der Startposition aus 300 cm vorwärtsfahren. Der Befehl wird über die ROS-Schnittstelle unter dem T*opic* /robot/engine/forwards übermittelt. Mit dem Reifendurchmesser von 6,7 cm ergibt sich ein Reifenumfang von 21,05 cm.

$$\text{Umfang} = \pi * \text{Durchmesser}$$

Aus der Schrittweite der Motoren von 1,8 Grad ergeben sich für eine volle Umdrehung von 360 Grad 200 Schritte.

$$Schritte\ volle\ Umdrehung = \frac{360°}{Schrittweite}$$

Durch den Umfang von 21,05 cm und den sich daraus ergebenden 200 Schritten für eine Umdrehung, kann errechnet werden, dass pro Schritt 0,10525 cm zurückgelegt werden. Bei einer zu fahrenden Strecke von 300cm ergeben sich 2850 Schritte (bzw. 14,25 Drehungen), die die Motoren ausführen müssen.

Diese Berechnungen macht der „Loomos Little Helper“ eigenständig. Im Programm werden dafür der Reifendurchmesser und die Anzahl der Schritte für eine vollständige Umdrehung hinterlegt. Über ROS kann dann die zu fahrende Strecke und die Geschwindigkeit übermittelt werden. Zu diesem Zweck wird auf dem *Topic* /robot/engine/forwards die zu fahrende Strecke und Geschwindigkeit *gepublished*. Im Anhang (A) befindet sich eine genaue Schnittstellenbeschreibung.

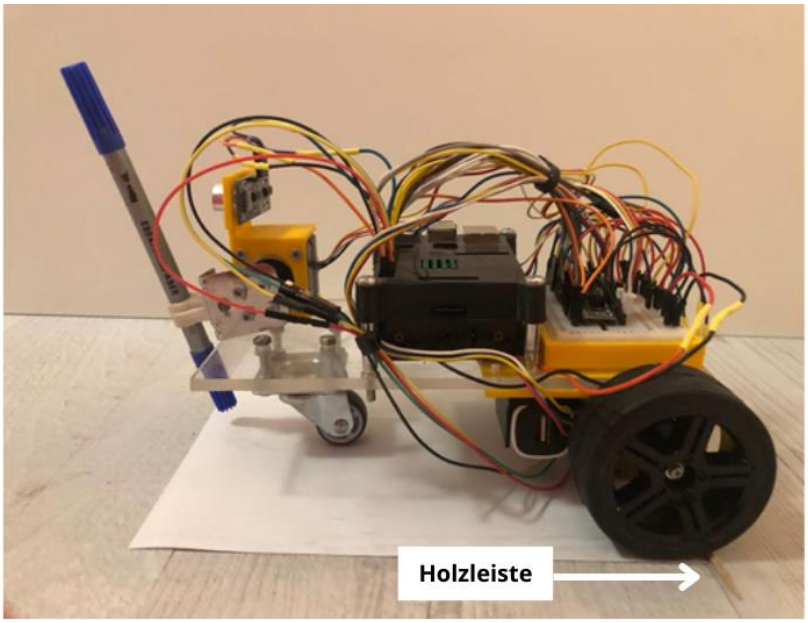

Abbildung 10: Startposition des " Loomos Little Helpers"

Für eine genaue Messung muss der „Loomos Little Helper“ immer von derselben Startposition starten, siehe Abbildung 10. Diese Startposition wurde mit Hilfe einer selbstgebauten Startbox sichergestellt. Im rechten Winkel zur Strecke wurde eine Holzleiste installiert, um die Hinterachse des „Loomos Little Helper“ für jede Versuchsfahrt identisch auszurichten.

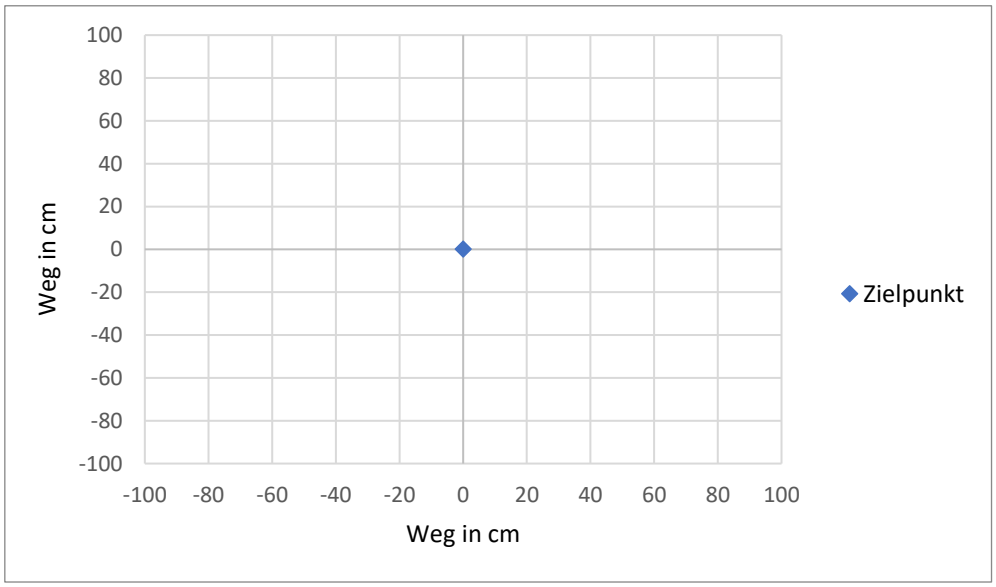

Abbildung 11: Koordinatensystem

Von dieser Startposition aus wurde ein Zielpunkt festgelegt, der genau 300 cm in Fahrtrichtung entfernt ist. Dieser Zielpunkt dient als Ursprung des Koordinatensystems, von dem aus die Odometriefehler gemessen werden (siehe Abbildung 11). Durch Einsatz verschiedener Geschwindigkeiten soll die Fahrgeschwindigkeit mit den wenigsten Odometriefehlern ermittelt werden.

4.2 Erwartungen des ersten Experimentes

Da bei der odometriebasierte Positionsbestimmung die meisten Fehler bei Drehungen entstehen [16], sollten folglich bei geraden Strecken weniger Fehler auftreten. Die Annahme geht von einer Abweichung von nicht mehr als 5 % aus, was bei einer Strecke von 300 cm einen maximalen Wert von 15 cm ergibt. Mit dem gewählten *Full step* Modus, welcher pro *Step* durch die Abmessung der Räder einen Weg von 0,11 cm zurücklegt, kann der angestrebte Weg mit hinreichender Genauigkeit zurückgelegt werden.

4.3 Ergebnisse des ersten Experimentes

Im Folgenden werden die Ergebnisse aus dem ersten Experiment dargestellt. Beim Starten der *ROS-Nodes* und der Übermittlung des Steuerungsbefehls für die Antriebsteuerung gab es keine Probleme und der „Loomos Little Helper“ hat sich in Richtung Zielposition bewegt. Für jede der fünf gefahrenen Geschwindigkeiten werden die Ergebnisse in einem Diagramm dargestellt und beschrieben.

1. **Geschwindigkeit 0,05 m/s**

   Beim ersten Versuch fährt der „Loomos Little Helper“ mit einer Geschwindigkeit von 0,05 m/s. Der Versuch wird viermal wiederholt. Die Ergebnisse werden in einem *Boxplot* dargestellt. Dadurch können Ausreißer und der Median direkt aus dem Diagramm abgelesen werden, was bei einem Punktediagramm nicht eindeutig möglich wäre. Die Skalierung der Diagramme wurde immer an die Ergebnisse angepasst, um eine bestmögliche Darstellung zu erreichen.

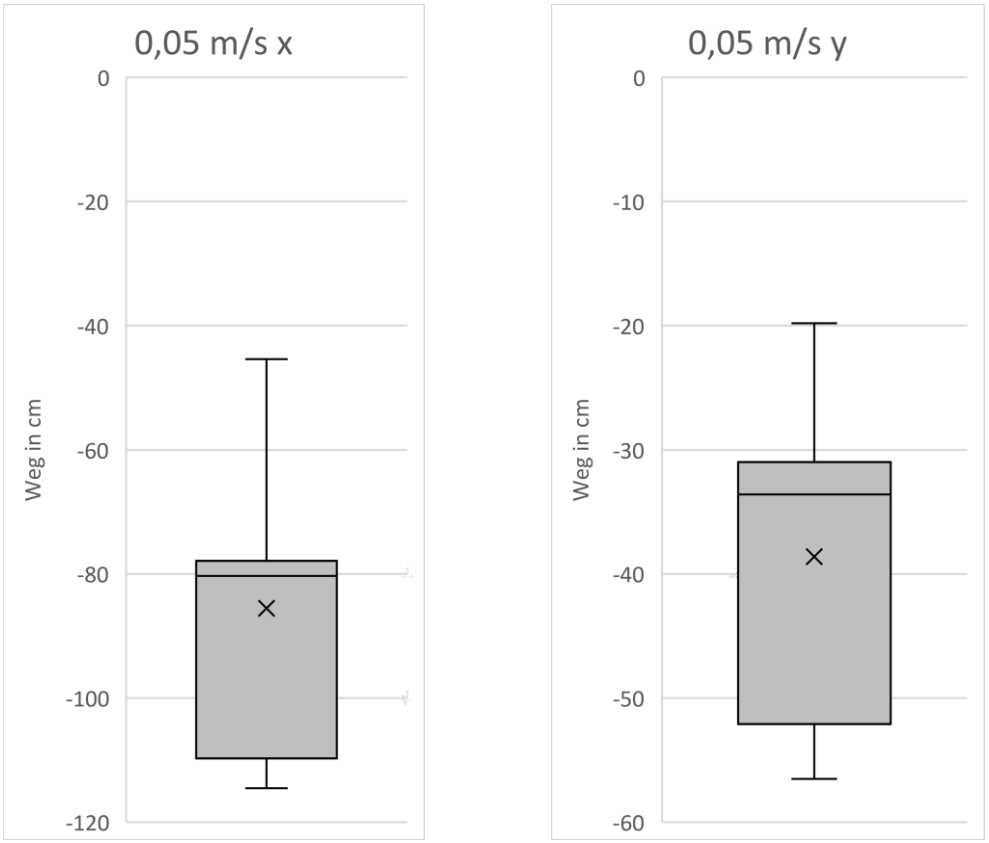

Abbildung 12: Ergebnisse des 1. Versuches im Boxplot dargestellt.

Bei diesen beiden Boxplots (Abbildung 12) werden die Abweichungen auf der x- und y-Achse dargestellt. Der Boxplot für die x-Achse geht von -120 cm bis 0 cm und der Boxplot für die y-Achse geht von -60 cm bis 0 cm. Die Box des Boxplots zeigt den Bereich an, in dem die mittleren 50 % aller Werte liegen. Bei diesem Versuch sind die Abweichungen mit einem maximalen Wert von -114,5 cm auf der x-Achse und einem maximalen Wert von -56,5 cm auf der y-Achse sehr groß. Die Maximalwerte werden durch die T-förmigen *Whisker* angezeigt, sofern es keine Ausreißer gibt. Der Median wird im Boxplot über die durchgezogene Linie in der Box dargestellt. Er liegt bei -80,3 cm auf der x-Achse und -33,6 auf der y-Achse. Der „Loomos Little Helper“ ist in Fahrtrichtung leicht nach Links gefahren. Der Interquartilsabstand beträgt für die x-Achse 31,8 cm und für die y-Achse 21,1 cm. Die mittlere Abweichung kann durch das „X“ im Boxplot abgelesen werden. Die prozentuale mittlere Abweichung liegt bei 28,52% auf der x-Achse und bei 12,87% auf der y-Achse.

2. Geschwindigkeit 0,1 m/s

Im zweiten Versuch wird die Geschwindigkeit des „Loomos Little Helper“ um 0,05 m/s erhöht auf 0,1m/s.

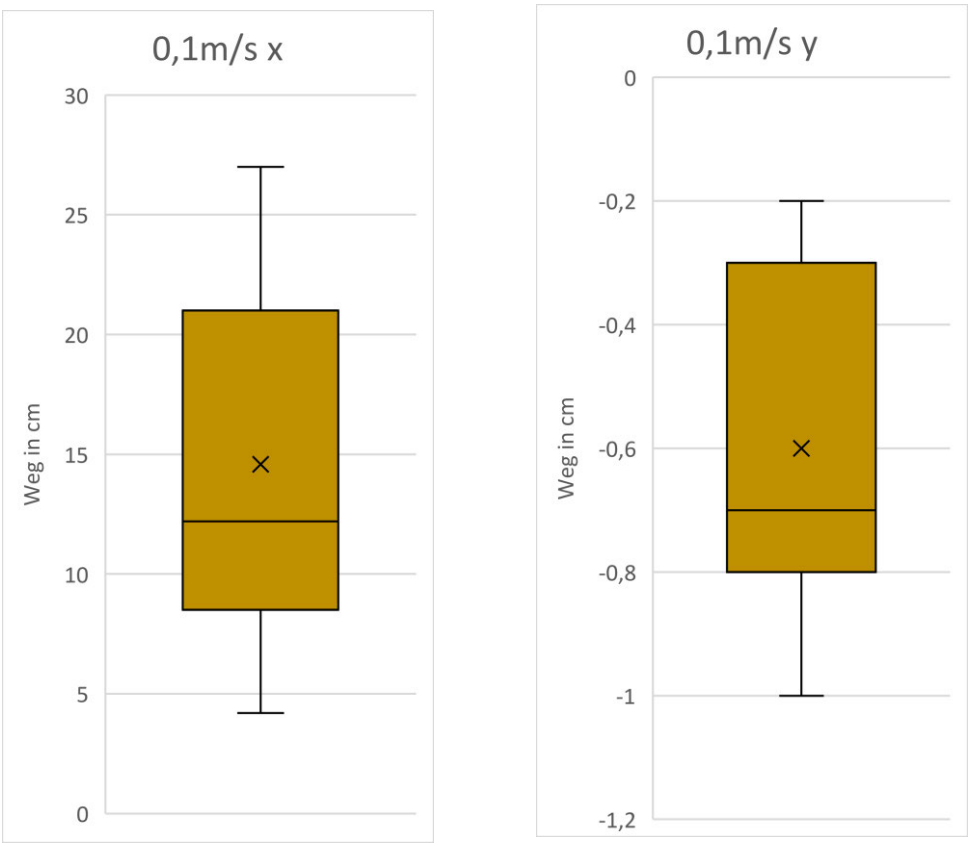

Abbildung 13: Ergebnisse des 2. Versuches im Boxplot dargestellt.

Wie beim ersten Versuch sind die gewonnenen Daten wieder in zwei Boxplots dargestellt. Der dargestellte Wertebereich des Boxplot für die x-Achse ist nun positiv und geht von 0 cm bis 30 cm . Im Boxplot für die y-Achse sind die Werte negativ und gehen von -1,2 cm bis 0 cm. Im Vergleich zum ersten Versuch sind die Ergebnisse sehr viel besser, da hier die maximale Abweichung auf der x-Achse bei 27 cm und auf der y-Achse nur bei -1 cm liegt. Der Medianwert liegt bei der x-Achse bei 12,2 cm und bei der y-Achse bei -0,7cm. Bemerkenswert ist an dieser Stelle, dass der „Loomos Little Helper“ nicht mehr in Fahrtrichtung nach links fährt, sondern ganz leicht nach rechts. Der Interquartilsabstand beträgt für die x-Achse 12,5 cm und für die y-Achse 0,5 cm. Die durchschnittliche Abweichung liegt bei 4,86% auf der x-Achse und bei 0,2% bei der y-Achse.

3. Geschwindigkeit 0,15 m/s

Im dritten Versuch wird die Geschwindigkeit des „Loomos Little Helper“ wieder um 0,05 m/s erhöht auf 0,15m/s.

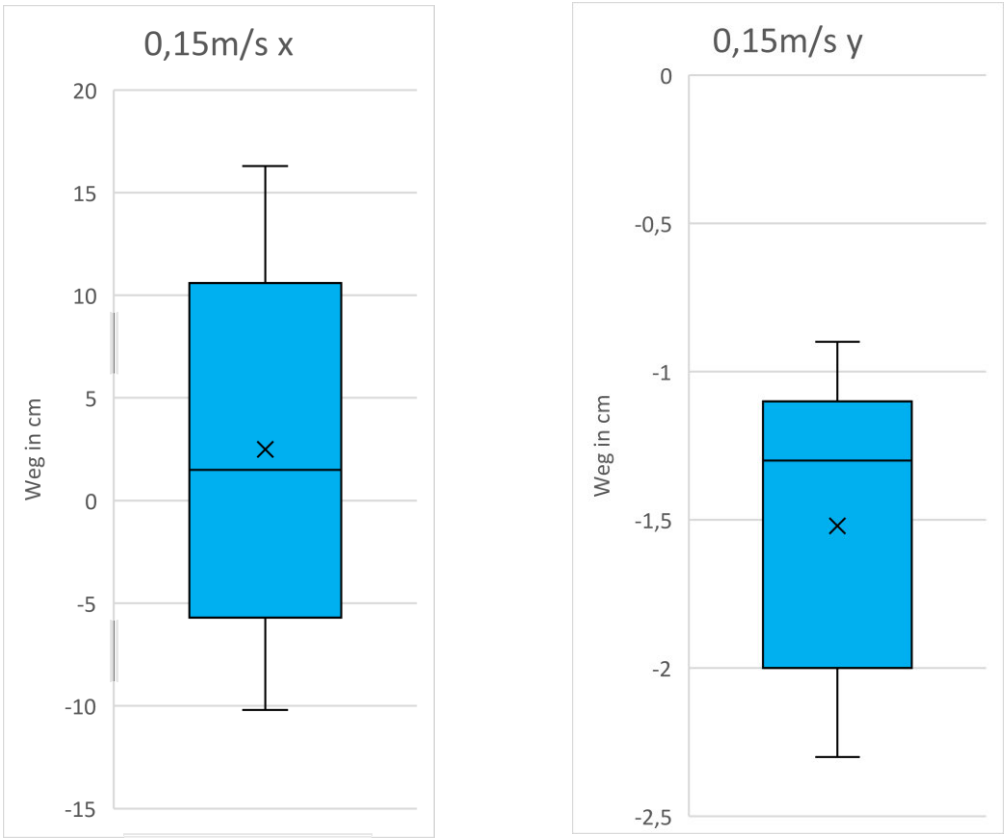

Abbildung 14: Ergebnisse des 3. Versuches im Boxplot dargestellt.

Die Ergebnisse sind ähnlich zum zweiten Versuch, wobei auffällig ist, dass die Werte auf der x-Achse zwischen positiv und negativ schwanken. Deshalb wird im Boxplot für die x-Achse der Wertebereich von -15 cm bis 20 cm dargestellt. Für die y-Achse ist der Wertebereich von -2,5 cm bis 0 cm. Die maximale Abweichung beträgt auf der x-Achse 16,3 cm und auf der y-Achse 2,3 cm. Der Medianwert liegt auf der x-Achse bei 1,5 cm und auf der y-Achse bei -1,3 cm. Der Interquartilsabstand beträgt für die x-Achse 16,3 cm und für die y-Achse 0,9 cm. Die durchschnittliche Abweichung liegt bei 2,95% auf der x-Achse und 0,51% auf der y-Achse, siehe Abbildung 14.

4. Geschwindigkeit 0,2 m/s

Im vierten Versuch wird die Geschwindigkeit des „Loomos Little Helper“ wieder um 0,05 m/s erhöht auf 0,2m/s.

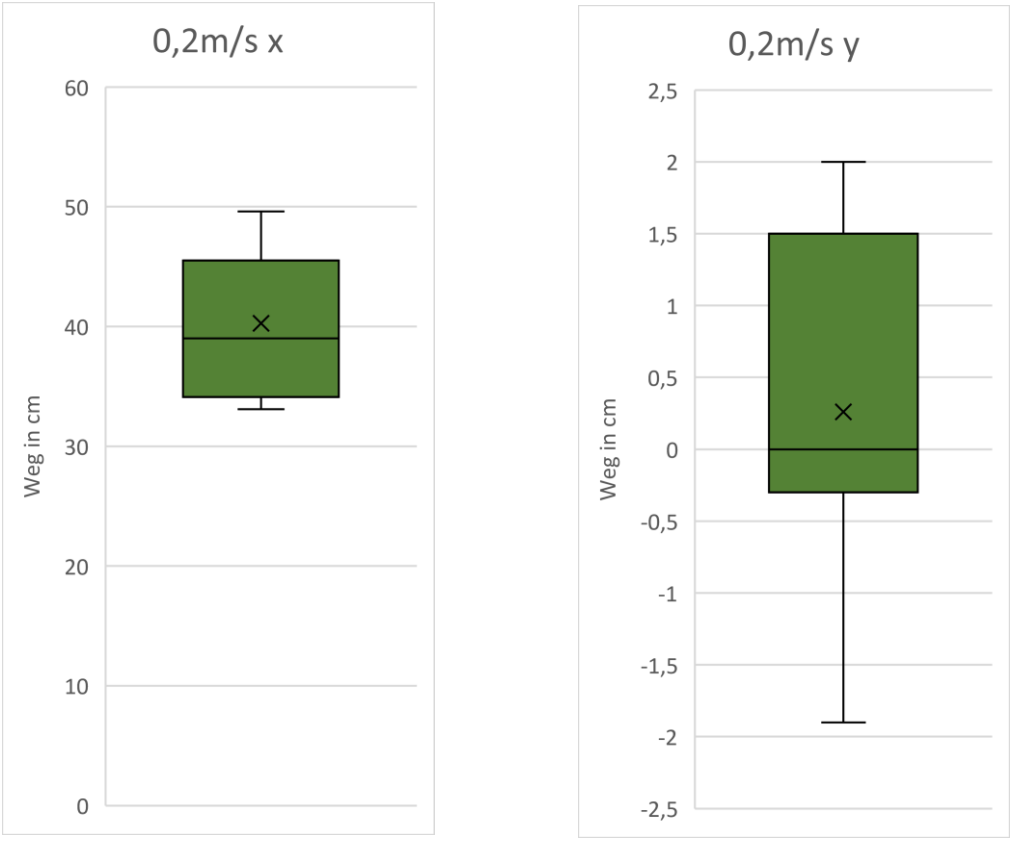

Abbildung 15: Ergebnisse des 4. Versuches im Boxplot dargestellt.

An diesen beiden Boxplots ist gut zu sehen, dass die Ergebnisse ab 0,2 m/s wieder schlechter werden. Der dargestellte Wertebereich geht auf der x-Achse von 0 cm bis 60 cm und auf der y-Achse von -2,5 cm bis 2,5 cm. Die maximale Abweichung liegt auf der x-Achse bei 49,6 cm und auf der y-Achse bei 2 cm. Auf Abbildung 15 ist auf der positiven y-Achse erkennbar, dass der „Loomos Little Helper“ bei dieser Geschwindigkeit teilweise über das Ziel hinausfährt. Das ist vermutlich auf den Bremsweg zurückzuführen. Der Medianwert liegt auf der x-Achse bei 39 cm und auf der y-Achse bei 0 cm. Der Interquartilsabstand beträgt für die x-Achse 11,4 cm und für die y-Achse 1,8 cm. Die durchschnittliche Abweichung liegt bei 13,42% auf der x-Achse und bei 1,5% auf der y-Achse.

5. Geschwindigkeit 0,25 m/s

Im fünften Versuch wird die Geschwindigkeit des „Loomos Little Helper“ wieder um 0,05 m/s auf 0,25m/s erhöht.

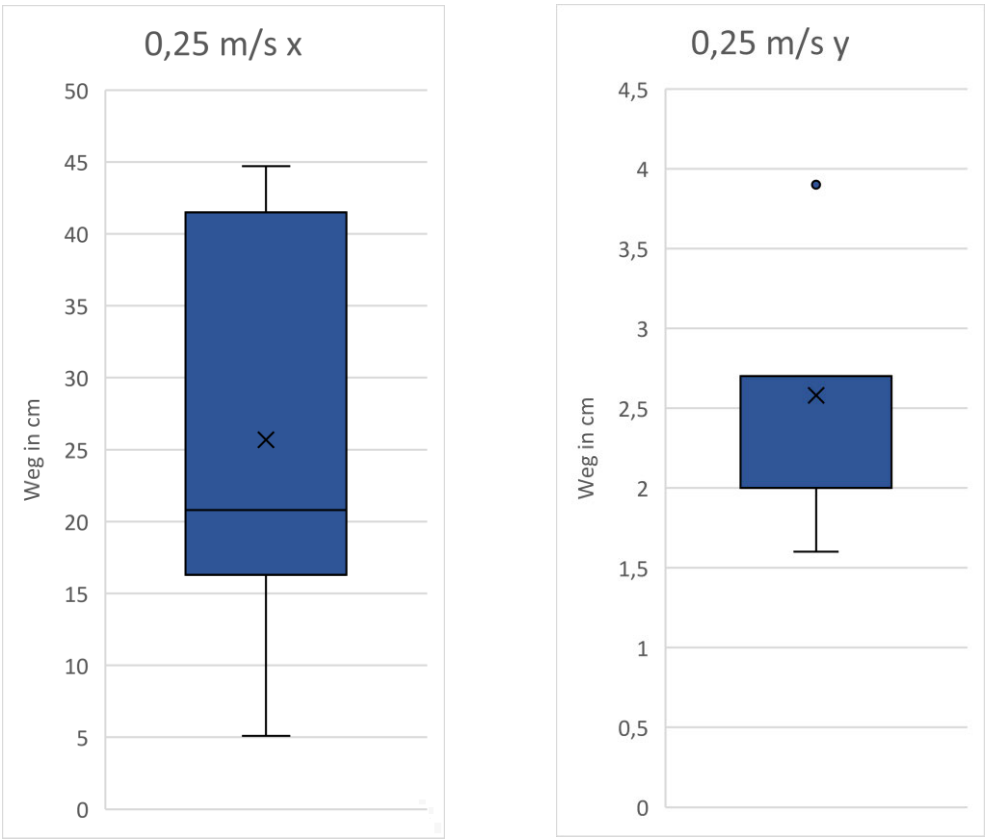

Abbildung 16: Ergebnisse des 5. Versuches im Boxplot dargestellt.

Der Wertebereich der beiden Boxplots in Abbildung 16 ist für die x-Achse von 0 cm bis 50 cm gewählt und für die y-Achse von 0 cm bis 4,5 cm. Wie auch schon beim vierten Versuch zu sehen war, ist auch hier erkennbar, dass der „Loomos Little Helper“ über das Ziel hinausgefahren ist. Die maximale Abweichung auf der x-Achse liegt bei 44,7 cm und auf der y-Achse sieht man einen Ausreißer mit einer Abweichung von 3,9 cm. Der Medianwert liegt auf der x-Achse bei 20,8 cm und auf der y-Achse bei 2,7 cm. Der Interquartilsabstand beträgt für die x-Achse 25,2 und ist damit im Vergleich zu Experiment 4 mehr als doppelt so groß. Für die y-Achse ist der Interquartilsabstand mit 0,7 cm nicht sehr groß. Die durchschnittliche Abweichung liegt bei 8,56% auf der x-Achse und bei 0,86% bei der y-Achse.

4.4 Vorbereitung des zweiten Experimentes

Im zweiten Experiment soll die Genauigkeit der Drehung bestimmt werden.

Der „Loomos Little Helper“ soll von der Startposition aus eine 360-Grad-Drehung vollziehen. Wie bereits im vorangegangenen Experiment, geschieht dies anhand der Ansteuerung der Motortreiber über die ROS-Schnittstelle und wird mittels der Odometrie berechnet. Um die Länge der zu fahrenden Strecke zu ermitteln, muss der Umfang des Spurkreises berechnet werden. Es wird angenommen, dass der Spurkreis mittig im Rad verläuft.

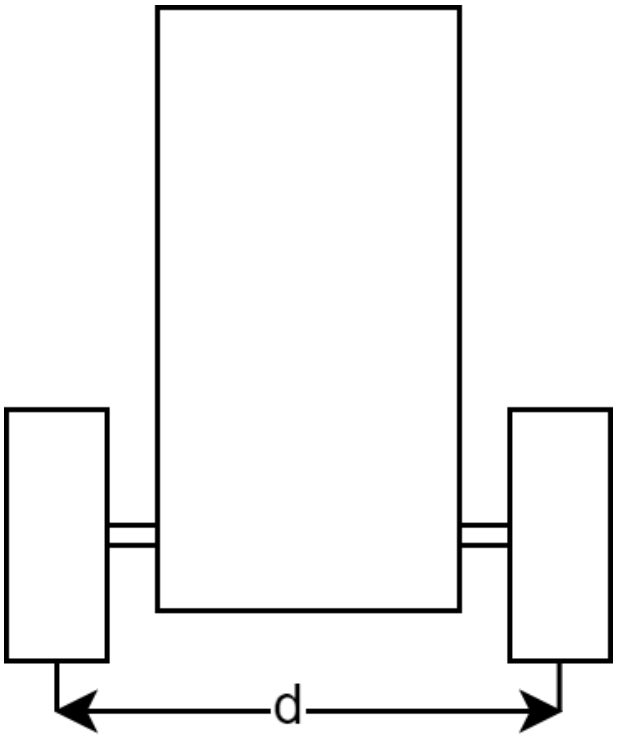

Abbildung 17: Draufsicht Loomos Little Helper

Der in Abbildung 17 eingezeichnete Durchmesser „d“ wird benötigt, um den Umfang des angenommenen Spurkreises des „Loomos Little Helpers“ zu berechnen. Dafür wird der Radabstand der beiden Hinterräder gemessen, der 11,21 cm beträgt. Mit dem Radabstand und der Radbreite von 2,8 cm lässt sich nun der Durchmesser berechnen.

$$Radabstand + 2 * \frac{Radbreite}{2} = \text{ Durchmesser}$$

Woraus folgt:

$$Radabstand + Radbreite = Durchmesser$$

Radabstand = 11,21 cm

Radbreite = 2,8 cm

$$Radabstand + Radbreite = 14{,}01\ cm$$

Mit dem Durchmesser lässt sich nun der Umfang berechnen.

$$\pi * Durchmesser = Umfang$$

$$\pi * 14{,}01\ cm = 44{,}01\ cm$$

Der Umfang ist die zu fahrende Strecke, um eine 360 Grad Drehung zu fahren.

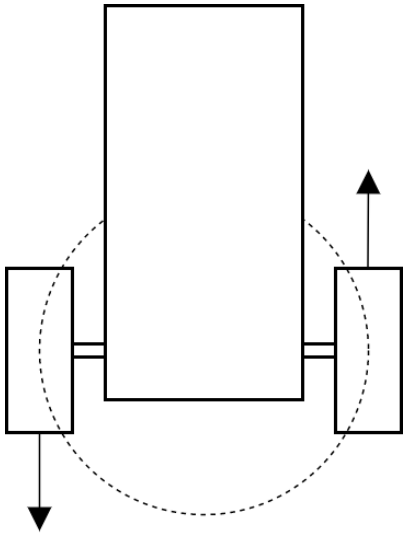

Abbildung 18: Linksdrehung des Loomos Little Helpers

In Abbildung 18 ist eine Linksdrehung abgebildet. Bei dieser fährt das rechte Rad nach vorne und das linke Rad nach hinten. Somit bleiben beide Räder auf der gestrichelten Linie und der „Loomos Little Helper“ vollzieht eine vollständige Drehung.

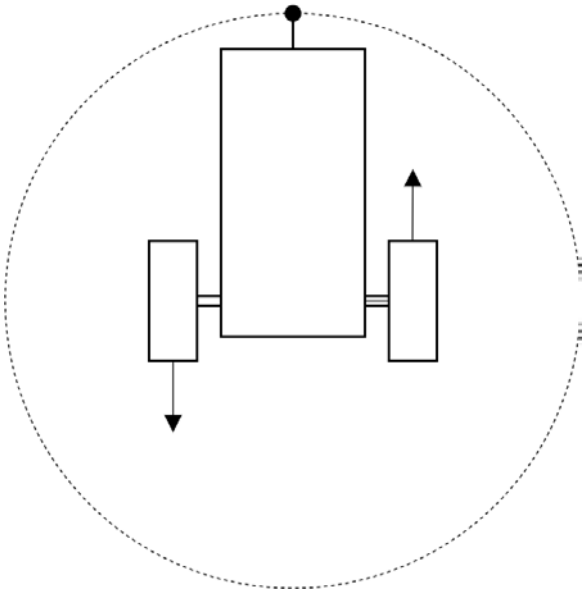

Abbildung 19: Aufbau des zweiten Experimentes

In Abbildung 19 ist der Aufbau des zweiten Experimentes zu sehen. Die Startposition wurde auf dem Boden eingezeichnet, damit der „Loomos Little Helper“ immer von der gleichen Position aus startet und somit gleiche Voraussetzungen für alle Versuche herrschen. Ausgehend von der Startposition soll der „Loomos Little Helper“ die 360-Grad-Drehung linksherum vollziehen und wieder in die Startposition zurückkehren. Der schwarze Punkt auf dem gestrichelten Kreis stellt einen Stift in der Multifunktionshalterung dar. An diesem Punkt wird das Ergebnis des Versuches gemessen. Dieses Experiment wird ebenfalls viermal wiederholt. Die Geschwindigkeit wird auf 0,15 m/s gesetzt.

4.5 Erwartungen Experiment Zwei

Wie im ersten Experiment wird eine Abweichung von maximal 5% erwartet. In diesem Experiment wird mit der Geschwindigkeit von 0,15 m/s die Drehung des „Loomos Little Helper“ durchgeführt. Bei einer Abweichung von nicht mehr als 5 % und bei Umdrehung von 360 Grad, entspricht dies einem maximalen Wert von 18 Grad.

4.6 Ergebnisse Experiment Zwei

Im Folgenden werden die Ergebnisse aus dem zweiten Experiment dargestellt. Das Starten der *ROS-Nodes* und die Übermittlung der Daten über das *Topic* /robot/engine/left verliefen problemlos und der „Loomos Little Helper“ hat sich in Bewegung gesetzt. Die Ergebnisse werden, wie im ersten Experiment, in einem Boxplot-Diagramm dargestellt und beschrieben.

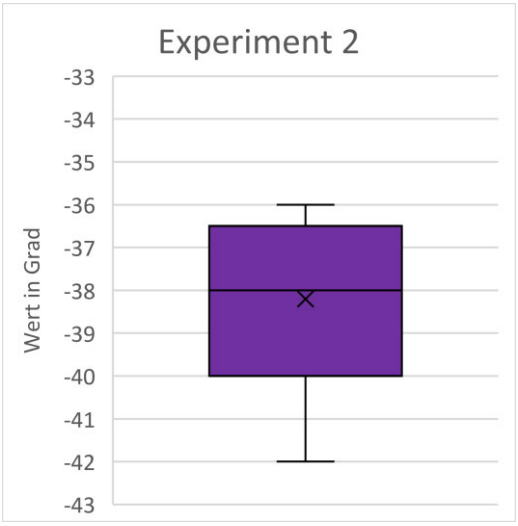

Abbildung 20: Ergebnisse des zweiten Experimentes im Boxplot dargestellt

Bei diesem Boxplot, dargestellt in Abbildung 20, werden die Abweichungen in Grad für eine vollständige Umdrehung erkennbar. Vollzieht der „Loomos Little Helper“ zum Beispiel eine Drehung von 330 Grad, ergibt sich eine Abweichung von -30 Grad. Der dargestellte Wertebereich reicht von -33 Grad bis -43 Grad. Bei diesem Versuch sind die Abweichungen mit einem maximalen Wert von -42 Grad sehr groß. Der Median liegt bei -38 Grad. Während der Versuchsdurchführung konnte nicht beobachtet werden, dass die Räder die Haftung verloren haben. Die durchschnittliche Abweichung liegt bei 10,61%.

Dieses Ergebnis deutet darauf hin, dass der angenommene Spurkeis nicht mittig im Rad verläuft. Aus den Messergebnissen lässt sich ein neuer Spurkreis berechnen und das Experiment wird mit den richtigen Parametern wiederholt werden.

4.7 Optimiertes Experiment Zwei

Aufgrund der Ergebnisse von Experiment Zwei, wurde entschieden das Experiment mit neu berechneten Parametern zu wiederholen. Aufgrund der zu erwartenden Streuung wurde anhand einer Mittelwertbetrachtung ein Drehwinkel von 321,8 Grad ermittelt. Mittels folgender Berechnung kann das Experiment erneut durchgeführt werden.

$$\frac{Fahrweg}{Fahrwinkel} * 360° = Sollweg$$

Bei einem Fahrweg von 44,01 cm muss dieser durch den gefahrenen Winkel von 321,8 Grad geteilt werden, um den tatsächlich gefahrenen Weg pro ein Grad zu ermitteln. Dieser wird mit 360 Grad multipliziert und ergibt somit den neuen Umfang.

$$\frac{44{,}01\ cm}{321{,}8°} * 360° = 49{,}23\ cm$$

Mit dem neu errechneten Umfang von 49,23 cm kann der genauere Durchmesser des Spurkreises berechnet werden.

$$\frac{Umfang}{\pi} = Durchmesser$$

$$\frac{49{,}23\ cm}{\pi} = 15{,}67\ cm$$

Mit dem neu berechneten Durchmesser von 15,67 cm sieht der Spurkreis wie in Abbildung 21 aus.

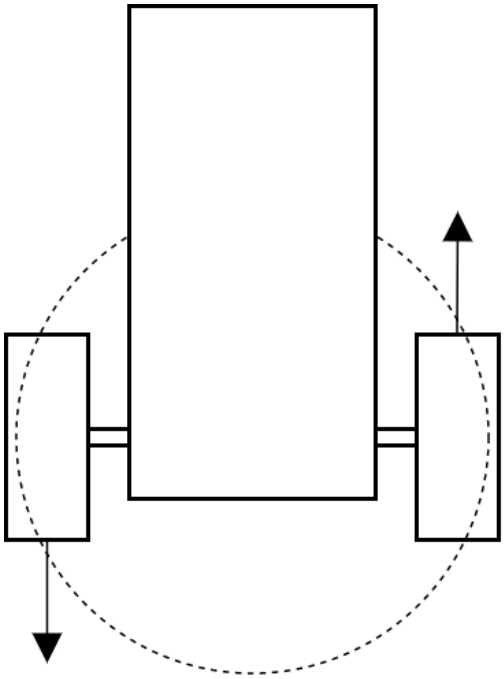

Abbildung 21: " Loomos Little Helper" Spurkreis

Mit Hilfe dieser Berechnung ist eine vollständige 360 Grad Drehung möglich. Das Experiment wird, wie in Abschnitt 4.4 beschrieben, wiederholt durchgeführt. Lediglich die Parameter haben sich geändert. Durch die geänderten Parameter sollten nun auch die Erwartungen von einer maximalen Abweichung von 5 % erfüllt werden.

4.8 Ergebnisse des optimierten Experiment Zwei

Im Folgenden werden die Ergebnisse des optimierten Experimentes dargestellt. Hierbei wurden, wie im vorangegangenen Experiment Zwei, die Parameter über das *Topic* /robot/engine/left übermittelt. Statt der 44,01 cm wurden nun 49,23 cm als Parameter übermittelt.

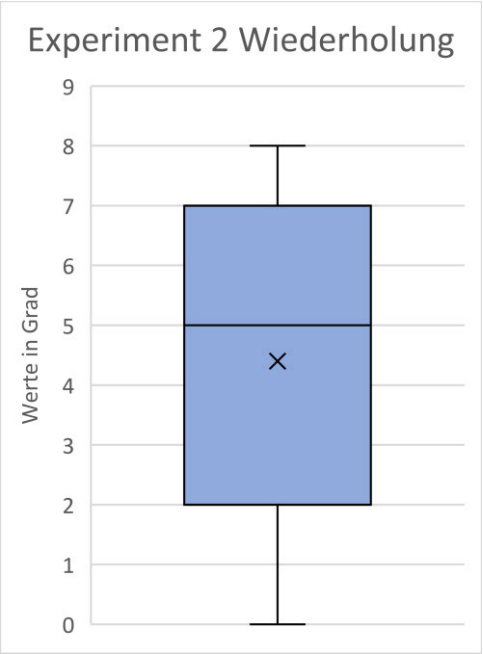

Abbildung 22: Experiment zwei mit neuen Parametern

In Abbildung 22 sind die Ergebnisse des optimierten Experimentes zu sehen. Der dargestellte Wertebereich geht von 0 cm bis 9 cm. Dabei sind schon deutlich bessere Ergebnisse erkennbar als im vorherigen Experiment, da der „Loomos Little Helper“ nun die komplette 360-Grad-Drehung absolviert. Bei diesem Versuch sind die Abweichungen mit einem maximalen Wert von 7 Grad groß. Der Median liegt bei 5 Grad und die durchschnittliche Abweichung bei 1,22%.

4.9 Auswertung des optimierten Experimentes

Bei der Optimierung des zweiten Experimentes sind im Vergleich zum ersten Versuch des zweiten Experimentes sehr viel bessere Ergebnisse erzielt worden. Die durchschnittliche Abweichung ließ sich von 10,61% auf 1,22% verringern. Somit verläuft der Spurkreis nicht, wie angenommen, mittig im Reifen, sondern weiter außen

5 Betrachtung der Ergebnisse

Im vorangegangenen Kapitel wurden die Experimente durchgeführt und die Ergebnisse dargestellt. Im Nachfolgenden wird auf diese Ergebnisse eingegangen und die möglichen Fehlerquellen betrachtet.

5.1 Zusammenfassung der Ergebnisse

In diesem Abschnitt werden alle Ergebnisse der Experimente zusammengefasst und verglichen. Bei Experiment Eins wird die Genauigkeit der Positionsbestimmung bei Geradeausfahrt anhand verschiedener Geschwindigkeiten verdeutlicht.

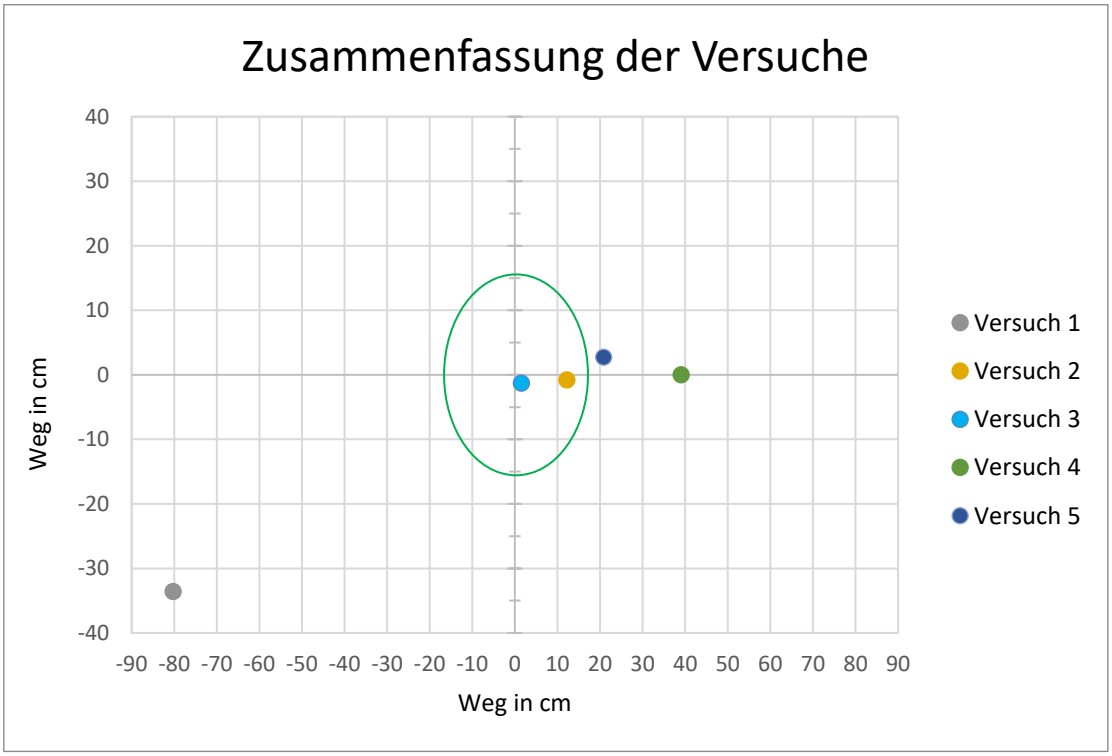

Abbildung 23: Zusammenfassung der Versuche aus Experiment Eins

In Abbildung 23 sind die Medianwerte des jeweiligen Versuches eingezeichnet. Der grüne Kreis markiert den Erwartungsbereich. Es ist zu erkennen, dass die Punkt aus Versuch Zwei und Drei sich im Erwartungsbereich befinden. Entgegen der Erwartung haben nicht alle Versuche des ersten Experimentes die Maximalabweichung von 5 % erfüllt. Im Vergleich zueinander hat sich ergeben, dass die Geschwindigkeit von 0,05 m/s im ersten Versuch sehr fehleranfällig ist. Bei Versuch Zwei wurden die Erwartungen von einer Abweichung von maximal 5% im Mittel erfüllt. Der dritte Versuch ist ähnlich dem Zweiten verlaufen. Auch hier wurden die Erwartungen im Mittelwert erfüllt, denn es gab lediglich einen Ausreißer, der die Erwartungen knapp nicht erfüllte. Ab Versuch Vier wurden die Ergebnisse wieder schlechter und die Erwartung wurde nicht erfüllt. Besonders auffällig war, dass teilweise die zu fahrenden 300 cm überschritten wurden. Bei Versuch Fünf wurden die zu fahrenden 300 cm jedes Mal überschritten. Ansonsten ist die durchschnittliche Abweichung auf der y-Achse bei allen Versuchen unter 2%, wodurch die erwarteten 5% Abweichung deutlich unterschritten wurden. Lediglich Versuch Eins bildet eine Ausnahme mit einer Abweichung von 12,87 %. Das zeigt, dass die Geschwindigkeit aus Versuch Eins nicht geeignet ist, den „Loomos Little Helper“ anzutreiben.

Die Abweichung im Experiment Zwei ist, verglichen zum ersten Experiment, deutlich höher. Die Erwartung von maximal 5 % wurde mit einer tatsächlichen Abweichung von durchschnittlich 10,61% deutlich überschritten. Die Streuung der Ergebnisse war sehr gering. Entgegen der Erwartung lag keines der Ergebnisse in der Nähe der vorab angenommen Abweichung von 5 %. Während des Experimentes war zu beobachten, dass der „Loomos Little Helper“ auf dem vorab eingezeichneten Wendekreis blieb, was bei einer Fahrwegabweichung von über 10 % nicht zu erwarten war. Durch die Optimierung des zweiten Experimentes, mit den neu berechneten Parametern, wurde die Fahrwegabweichung von 10,61% auf 1,22% verringert. Dadurch liegt der Durchmesser des Spurkreises nicht, wie zuerst angenommen bei 14,01 cm, sondern bei 15,67 cm

Bei der Durchführung der Experimente konnte außerdem beobachtet werden, dass sich die Leitungen auf der Steckplatine durch die Vibrationen der Motoren lösten. Im Verlauf der Experimente mussten die Steckverbindungen erneuert werden, was zur Folge hatte, dass einige Experimente wiederholt werden mussten.

5.2 Mögliche Fehlerquellen

In diesem Abschnitt wird auf möglichen Fehlerquellen der Ergebnisse aus Kapitel 4.2 eingegangen. Es werden verschiedene Ursachen für die Fehler betrachtet.

Um Softwarefehler auszuschließen, die eine fehlerhafte Motorenansteuerung auslösen könnten, wurde an den Reifen eine Markierung angebracht. Dadurch konnten die Reifenumdrehungen manuell und optisch gezählt werden. Außerdem wurde es dadurch möglich, die Gleichmäßigkeit der Umdrehungen zu kontrollieren

Diese manuelle Zählung und Kontrolle wurde bei jedem Versuch in Experiment Eins durchgeführt. Es stellte sich heraus, dass die Reifen bei jedem der Versuche gleichmäßig drehten und auch die im Abschnitt 4.1 berechneten 14,25 Umdrehungen ausgeführt wurden. Durch diese einfache Methode konnten die Umdrehungen überprüft und somit Softwarefehler ausgeschlossen werden.

Die Ergebnisse legen nahe, dass die Fehlerquellen nicht auf Seiten der Software liegen und sehr wahrscheinlich einen mechanischen Ursprung haben. Im Folgenden werden Vermutungen aufgestellt, welche Einflüsse diese Fehler verursacht haben könnten.

Wie im Buch „*Springer Handbook of Robotics*“ beschrieben, können die folgenden Größen Einfluss auf die Genauigkeit der odometriebasierten Positionsmessung haben. [17]

Die Radgeometrie, Unrundheiten, Verschleiß oder fehlerhafte Messungen des Durchmessers können hier Einfluss nehmen. Wenn durch einen dieser Faktoren der angenommene Durchmesser nicht mit dem realen Durchmesser übereinstimmt, ist die Berechnung, die der „Loomos Little Helper“ durchführt, fehlerhaft. [17]

Bei der Bodenbeschaffenheit können Unebenheiten und Schlupf, also das Durchrutschen der Räder, Einfluss nehmen. In Verbindung mit einer ungleichen Verteilung des Fahrzeuggewichtes kann es dazu führen, dass ein Rad stärker belastet wird, als das andere und dadurch der Schlupf bei den Rädern unterschiedlich auffällt. [17]

Eine ungleiche Verteilung des Gewichtes ist beim „Loomos Little Helper“ gegeben, da sich der Motor für den Multifunktionsarm, konstruktionsbedingt, nicht in der Mitte befindet. Dies

führt dazu, dass sich mehr Gewicht auf der rechten Seite des „Loomos Little Helper“ befindet. Dieses Ungleichgewicht könnte auch die Linkskurve im ersten Versuch des ersten Experimentes erklären. [17]

Im vierten Versuch des ersten Experiments wurde beobachtet, dass die zu fahrenden 300 cm teilweise überschritten wurden. Im fünften Versuch war dies immer der Fall. Sehr wahrscheinlich ist die hohe Geschwindigkeit und der daraus resultierende Bremsweg die Ursache.

Mit Hilfe des zweiten Experimentes wurde festgestellt, dass der Spurkreis nicht mittig im Reifen verläuft, wie zuerst angenommen. Um dies zu belegen, wurden die Parameter angepasst und das Experiment erneut durchgeführt.

Die Mechanik des „Loomos Little Helper“ liegt nicht im Untersuchungsbereich dieser Arbeit. Da die Betrachtung mechanischer Fehler den Rahmen zusätzlich überschreiten würde, wird auf diese möglichen Fehler nicht eingegangen.

6 Fazit und Ausblick

Im Zuge dieser Arbeit wurde eine API für den „Loomos Little Helpers“ entwickelt, die das Ansteuern der Aktoren und Sensoren des Roboters, sowie die implementierten Bewegungs- und Steuerungsfunktionen ermöglicht.

Die API wurde auf zwei Komponenten aufgeteilt, wodurch jeweils eine für die Steuerung der Motoren des Roboters und eine für die Ansteuerung der Sensoren zuständig ist. Die Implementierung der benötigten Schnittstellen erfolgt über das ROS-Framework. Der Schwerpunkt dieser Arbeit wurde auf die erste Komponente gelegt, die für die Steuerung der Motoren des Roboters zuständig ist.

Um die Softwarefunktionen des Roboters bewerten zu können und um zu entscheiden, ob diese für die weitere Entwicklung des „Loomos Little Helpers“ verwendet werden können, wurden mehrere Experimente durchgeführt.

Im ersten Experiment wurde das Fahrverhalten unter verschiedenen Geschwindigkeiten beobachtet. Es wurde festgestellt, dass das Fahrverhalten des „Loomos Little Helpers“ bei den Geschwindigkeiten 0,1 m/s und 0,15 m/s am präzisesten ist.

Mit Hilfe des zweiten Experimentes wurde die Drehung des „Loomos Little Helpers“ auf Genauigkeit der Odometrie untersucht. Dabei wurde festgestellt, dass die Annahmen für die Berechnung der Parameter nicht ganz korrekt waren. Leider war dies im Vorfeld nicht abzusehen. Anhand der Erkenntnisse aus dem ersten Versuch des zweiten Experimentes wurden deshalb neue Parameter berechnet und mit diesen das Experiment optimiert. Bei der Optimierung des zweiten Experimentes waren die Ergebnisse deutlich besser. Die Abweichungen sanken im Schnitt von 10,61% auf durchschnittlich 1,22%.

Die Erkenntnisse aus den Experimenten zeigen, dass allein durch Odometrie keine genaue Positionsbestimmung beim „Loomos Little Helper“ möglich ist, denn schon bei kurzen Strecken entstehen hohe Ungenauigkeiten. Zwar erfüllten die Geschwindigkeiten von 0,1 m/s und 0,15 m/s die Erwartung und eine Veränderung der Parameter im zweiten Experiment erbrachte eine Verbesserung der Ergebnisse, jedoch summierten sich die Fehler der odometriebasierten

Positionsmessung. Zusätzlich fehlende Korrekturmöglichkeiten legen nahe, eine alternative Positionsbestimmung oder eine andere Art der Korrekturmöglichkeit zu integrieren.

Als Ausblick für die Zukunft kann die bei dieser Arbeit entstandene API und die Anbindung von ROS eine gute Grundlage für weitere Projekte bilden. Des Weiteren werden im nachfolgenden Abschnitt noch einige Ideen zu möglichen Verbesserungen des Roboters aufgezeigt.

6.1 Ideen zur Verbesserung

Wie in Abschnitt 5.1 beschrieben, lösten sich die Steckverbindungen der Leitungen aufgrund der Vibrationen der Motoren. Das *Breadboard* könnte durch eine Leiterplatte ersetzt werden und die Verbindungen fest verlötet werden.

Beim Abbremsen aus höheren Geschwindigkeiten tritt das Phänomen Bremsweg auf, welches die zurückgelegte Strecke verfälscht. Diesem Phänomen kann durch die Wahl einer geringeren Geschwindigkeit oder dosierten Bremsung entgegengewirkt werden.

Ein weiteres Problem stellt die ungleiche Gewichtsverteilung dar. Dies kann durch die Anbringung eines Gegengewichts gelöst werden. Durch das Gegengewicht wird das Gewicht gleichmäßig auf die beiden Hinterräder verteilt.

Weitere Sensoren, die zum Beispiel einen aktuellen Positionsabgleich durchführen, können die sehr fehleranfällige odometriebasierte Positionsbestimmung verbessern. Ein Beispiel für einen gut geeigneten Sensor wäre ein „Gyroskop-Winkel“. [16]

Literaturverzeichnis

[1] C. Heer, „IFR International Federation of Robotics," 2021. [Online]. Available: https://ifr.org/ifr-press-releases/news/robot-density-nearly-doubled-globally. [Zugriff am 5 April 2022].

[2] J. Hertzberg, K. Lingemann und A. Nüchter, „Mobile Roboter," Berlin Heidelberg, Springer-Verlag Berlin Heidelberg, 2012, pp. 7-8.

[3] J. M. O'Kane, „A Gentle Introduction to ROS," Columbia, 2018, p. 1.

[4] „Rasppishop," [Online]. Available: https://www.rasppishop.de/Raspberry-Pi-3-Model-B-14-GHz-64Bit-Quad-Core. [Zugriff am 23 05 2022].

[5] J. M. O'Kane, „A Gentle Introduction to ROS," Columbia, 2018, pp. 2-4.

[6] J.-D. Walz, „Fraunhofer-Institut für Produktionstechnik und Automatisierung. Jahresbericht 2012 Seite 46," Fraunhofer IPA, Stuttgart, 2012.

[7] L. Upton, „https://www.raspberrypi.com/," 2021. [Online]. Available: https://www.raspberrypi.com/news/ten-years-of-the-raspberry-pi-blog/. [Zugriff am 12 04 2022].

[8] E. Rummich, „Elektrische Schrittmotoren und -antriebe," P. D. D. h. W. J. Bartz, D. H. Mesenholl und D. E. Wippler, Hrsg., Renningen, expert verlag, 2015, pp. 1-37.

[9] J. Hertzberg, A. Nüchter und K. Lingemann, „Mobile Roboter," Berlin Heidelberg, Springer-Verlag, 2012, p. 123.

[10] T. Zimmermann, „robotik-produktion," 2019. [Online]. Available: https://www.robotik-produktion.de/robotik-und-produktion-2-2019/robot-operating-system/. [Zugriff am 11 04 2022].

[11] V. Mazzari, „https://www.generationrobots.com/," 17 12 2019 . [Online]. Available: https://www.generationrobots.com/blog/de/ros2-was-andert-sich-gegenuber-ros/. [Zugriff am 30 05 2022].

[12] „STAIR: STanford Artificial Intelligence Robot," [Online]. Available: http://stair.stanford.edu. [Zugriff am 19 05 2022].

[13] „Open robotics," Open Source Robotics Foundation, [Online]. Available: https://www.openrobotics.org/. [Zugriff am 19 05 2022].

[14] „ROS Introduction," ROS, [Online]. Available: http://wiki.ros.org/ROS/Introduction. [Zugriff am 19 05 2022].

[15] J. Hertzberg, K. Lingemann und A. Nüchter, „Mobile Roboter," Berlin Heidelberg, Springer-Verlag Berlin Heidelberg, 2012, pp. 326-330.

[16] J. Hertzberg, K. Lingemann und A. Nüchter, „Mobile Roboter," S. Vieweg, Hrsg., Berlin, Heidelberg, Springer-Verlag Berlin Heidelberg, 2012, pp. 144-147.

[17] B. Siciliano und O. Khatib, „Handbook of Robotics," Springer, Hrsg., Berlin, Heidelberg , Springer-Verlag , 2008, pp. 477-481.

A ROS-Schnittstellenimplementierung

Vollständige Übersicht der implementierten ROS-Schnittstellen mit den benötigten Datentypen, den empfangenden *ROS-Nodes*, sowie ihre Implementierung.

Topic /cmd_vel Datentyp: geometry_msgs/Twist

Empfänger: robot_engine

Verarbeitende Funktion: cmdVelCallback()

- Empfang von Translations- und Rotationsgeschwindigkeiten in m/s für die Steuerung des Roboters
- Weiterleiten der Daten

Topic /robot/engine/left Datentyp: Move.srv

Empfänger: robot_engine

Verarbeitende Funktion: `leftCallback()`

- Empfang von einer Rotationsbewegung für die Steuerung des Roboters
- Ausführen der Rotation
- Feedback bei vollständiger Ausführung der Rotation

Topic /robot/engine/right Datentyp: Move.srv

Empfänger: robot_engine

Verarbeitende Funktion: `rightCallback()`

- Empfang von einer Rotationsbewegung für die Steuerung des Roboters

- Ausführen der Rotation
- Feedback bei vollständiger Ausführung der Rotation

Topic /robot/engine/backwards Datentyp: Move.srv

Empfänger: robot_engine

Verarbeitende Funktion: `backwardsCallback()`

- Empfang von einer Translationsgeschwindigkeit in m/s für die Steuerung des Roboters
- Ausführung der rückwärts Bewegung
- Feedback bei vollständiger Ausführung

Topic /robot/engine/forwards Datentyp: Move.srv

Empfänger: robot_engine

Verarbeitende Funktion: `forwardsCallback()`

- Empfang von einer Translationsgeschwindigkeit in m/s für die Steuerung des Roboters
- Ausführung der vorwärts Bewegung
- Feedback bei vollständiger Ausführung

Topic /robot/engine/up Datentyp: Move.srv

Empfänger: robot_engine

Verarbeitende Funktion: `upCallback()`

- Empfang von einer Hubbewegung in m/s für die Steuerung des Hubmotors
- Ausführung des Aufwärtshubs
- Feedback bei vollständiger Ausführung

Topic /robot/engine/down Datentyp: Move.srv

Empfänger: robot_engine

Verarbeitende Funktion: `downCallback()`

- Empfang von einer Hubbewegung in m/s für die Steuerung des Hubmotors
- Ausführung der Abwärtshubs
- Feedback bei vollständiger Ausführung

Topic /robot/sonar/front/distance Datentyp: GetDistance.srv

Empfänger: robot_sonar

Verarbeitende Funktion: `getDistanceSensorFront()`

- Messen des Abstandes zum nächsten Objekt vor dem Roboter
- Feedback mit dem Messergebnis

Topic /robot/sonar/back/distance Datentyp: GetDistance.srv

Empfänger: robot_sonar

Verarbeitende Funktion: `getDistanceSensorFront()`

- Messen des Abstandes zum nächsten Objekt hinter dem Roboter
- Feedback mit dem Messergebnis

B ROS-Installation

Im Folgenden wird die Installation von ROS für den Versuchsaufbau beschrieben. Um ROS auf dem Raspberry Pi zu installieren und brauchbar zu machen, muss zunächst ein kompatibles Betriebssystem installiert werden. Hierfür wurde Ubuntu 18.04 LTS gewählt. Es muss darauf geachtet werden, dass sowohl der Einplatinenrechner als auch die ROS Version mit dem Betriebssystem kompatibel sind.

In dieser Arbeit wird die ROS Version „*Melodic Morenia*“ verwendet, für welches das Betriebssystem Ubuntu 18.04 empfohlen wird. Um nachfolgende nötige Schritte auszuführen, müssen die gezeigten Befehle in der Konsole eingegeben werden.

1. Computer so einrichten, dass er Software von packages.ros.org akzeptiert.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc
) main" > /etc/apt/sources.list.d/ros-latest.list'
```

2. Schlüssel einrichten

```
sudo apt install curl

curl -s https://raw.githubusercontent.com/ros/rosdistro/master
/ros.asc | sudo apt-key add -
```

3. Ubuntu aktualisieren

```
sudo apt update
```

4. ROS Installation

```
sudo apt install ros-melodic-desktop-full
```

5. Umgebungsvariablen einrichten

```
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

6. Installieren von *Dependencies* zum Bauen von *ROS packages*

```
sudo apt install python-rosdep python-rosinstall python-rosins
tall-generator python-wstool build-essential
```

7. Initialisieren von ROS

```
sudo rosdep init
rosdep update
```

Für eine aktuelle Installationsanleitung empfiehlt sich der Besuch der offiziellen Webseite[7].

[7] ROS: Installation - https://wiki.ros.org/ROS/Installation

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

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

_____	_____	_____
Ort	Datum	Unterschrift im Original