Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Fabian Schmidt

Entwicklung eines Human-Computer-Interface zur Steuerung eines Telepräsenzroboters

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Fabian Schmidt

Entwicklung eines Human-Computer-Interface zur Steuerung eines Telepräsenzroboters

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. rer. nat. Thomas Lehmann
Zweitgutachter: Prof. Dr. Tim Tiedemann

Eingereicht am: 5. Oktober 2018

Fabian Schmidt

Thema der Arbeit
Entwicklung eines Human-Computer-Interface zur Steuerung eines Telepräsenzroboters

Stichworte
Robotik, Sensorik, Aktorik, Rollstuhl

Kurzzusammenfassung
Im Rahmen dieser Arbeit wird ein intuitives Human-Computer-Interface zur Steuerung eines Telepräsenzroboters in der Form eines modifizierten Rollstuhls entwickelt. Ein solches Interface ist herkömmlichen Eingabegeräten (im Idealfall) in Bezug auf den Grad der Immersion deutlich überlegen.

Fabian Schmidt

Title of the paper
Development of a Human Computer Interface to Control a Telepresence Robot

Keywords
robotics, sensor system, actuating elements, wheelchair

Abstract
Within the scope of this thesis, an intuitive human-computer interface in the form of a modified wheelchair for controlling a telepresence robot is being developed. Such an interface is (ideally) vastly superior to conventional input devices in terms of the degree of immersion.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

Im Rahmen dieser Arbeit wird ein intuitives Human-Computer-Interface zur Steuerung eines Telepräsenzroboters in der Form eines modifizierten Rollstuhls entwickelt. Ein solches Interface ist herkömmlichen Eingabegeräten (im Idealfall) in Bezug auf den Grad der Immersion deutlich überlegen.

Üblicherweise basieren Controller zur Steuerung von technischen Geräten (wie etwa Robotern) auf Knöpfen und Reglern, welche die möglichen Befehle des Benutzers in abstrakter Weise darstellen. Die Idee des Controllers, welcher in dieser Arbeit entwickelt wird, ist es dagegen, eine intuitivere und selbsterklärendere Steuerung zu ermöglichen, indem der Abstraktionsgrad gesenkt wird. Dafür wird der Controller in seiner Gestalt dem zu steuernden Objekt, einem Roboter mit zwei Rädern, durch den Einsatz eines Rollstuhls angenähert.

Der Nachteil dieser Anpassung ist die stärkere Spezialisierung auf ein Einsatzgebiet – andere Dinge können mit diesem Controller nicht mehr gut gesteuert werden. Diese Arbeit baut auf einer existierenden Mechatronik-Projektarbeit aus dem Jahr 2017 an der HAW Hamburg auf [GRIP 2018].

2 Analyse des Vorgängerprojekts

Auf Basis der kommerziellen und universellen Roboter-Plattform Pioneer 3DX hat Hendrik Wiese im Rahmen einer Mechatronik-Bachelorarbeit im Jahr 2014 an der HAW Hamburg den Telepräsenzroboter Mory-A entwickelt. Dieser Roboter wurde zunächst mit einem handelsüblichen Playstation-Controller gesteuert.

Um die Intuitivität von Mory-A zu erhöhen, haben die Mechatronik-Studenten Joschka Sondhof, Tim Brockmann, Reynaldo Gunawan, Rinaldy Sutyono und Andreas Bloch im Rahmen einer Projektarbeit namens Grip versucht, die Steuerung des Roboters mithilfe eines Rollstuhls zu realisieren. Da das Ergebnis der Grip-Projektarbeit nicht funktionstüchtig war, wird der Rollstuhl-Controller nun in dieser Arbeit überholt und funktionstüchtig gemacht.

Im Folgenden wird der Stand des Grip-Projekts bei der Übergabe an den Autor beschrieben.

2.1 Aufbau des Telepräsenzroboters

Der Roboter Mory-A besitzt zwei angetriebene Räder und ein Schwenkrad, welches nicht direkt steuerbar ist. Gelenkt wird ausschließlich durch unterschiedliche Geschwindigkeiten der einzeln angetriebenen Räder. Aufgrund dieser Eigenschaften weist der Roboter starke Ähnlichkeiten zu einem Rollstuhl auf.

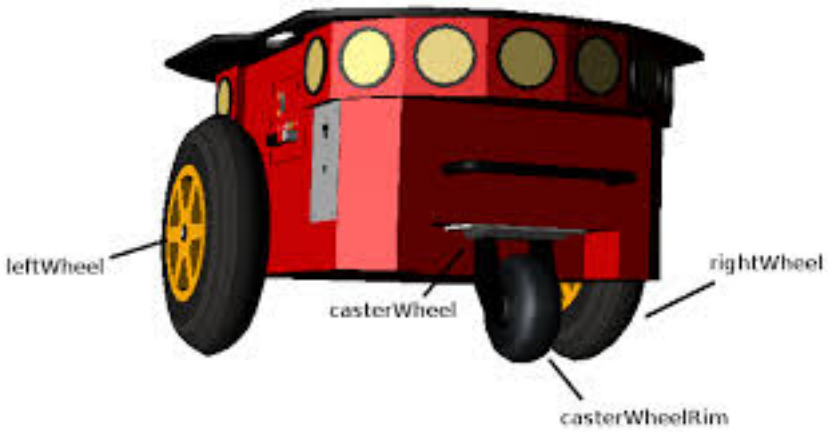

Abbildung 2.1: Modelle des Pioneer 3-DX Seshadri 2018

Mory-A verfügt außerdem über einen beweglichen Kameraaufbau: Zwei Kameras sind nebeneinander angeordnet und in ihrem Abstand den menschlichen Augen nachempfunden.

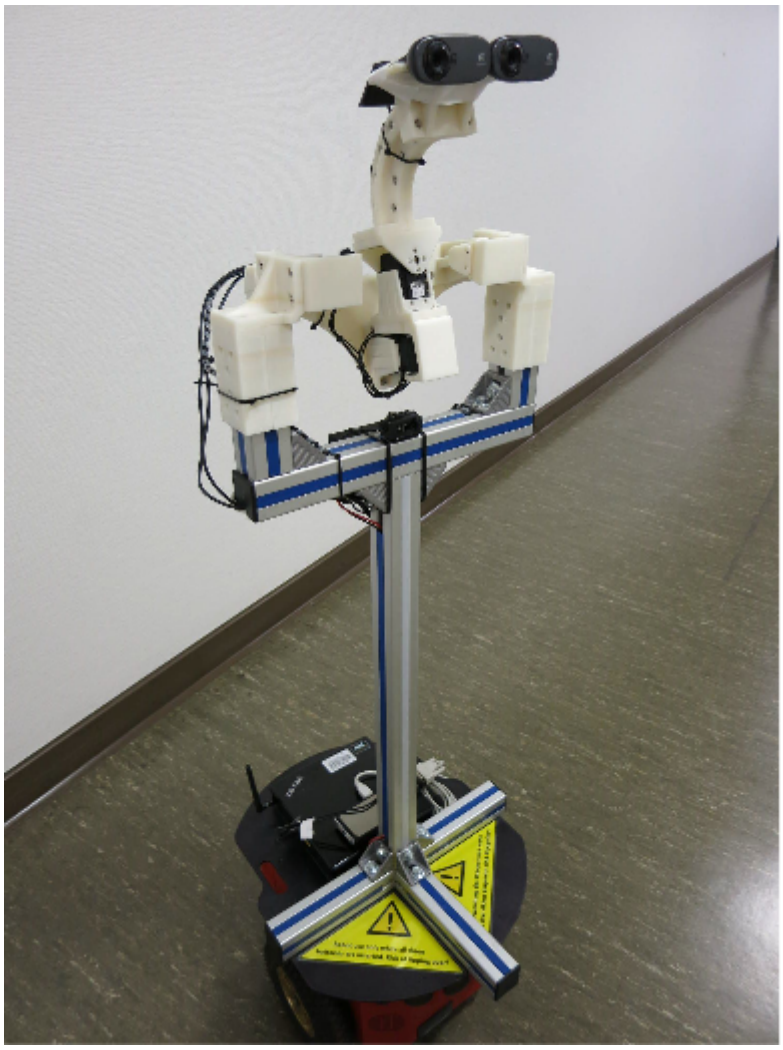

Abbildung 2.2: Mory-A Telepräsenzsystem

Der Roboter wird über drei 12-V-Batterien mit Spannung versorgt. Die Steuerung und Telemetrie des Pioneer 3DX erfolgt über eine RS232-Schnittstelle. Über diese Schnittstelle ist der 3DX mit einem Mini-Computer (Intel Nuc) verbunden, der auch von den Roboter-Batterien gespeist wird. Auf diesem Computer läuft ein Ubuntu-Betriebssystem mit den notwendigen Software-Komponenten. Auch die beiden Kameras sind über USB mit dem Rechner verbunden. Außerdem stellt der Nuc über WLAN eine Netzwerkverbindung her.

2.2 Aufbau des Rollstuhls

Der Rollstuhl, welcher zum Controller umgebaut wurde, ist ein Modell namens Basic und stammt von der Firma Berollka-aktiv Rollstuhltechnik GmbH.

Abbildung 2.3: Berollka-aktiv Basic GmbH 2018

Beim Grip-Projekt wurde für den Rollstuhl ein Podest gebaut, sodass die Räder in der Luft hängen und frei bewegbar sind. Um die Position der Räder zu ermitteln, wurde ebenfalls ein Gestell aus Alu-Profilen gebaut, an welches Drehencoder montiert wurden (siehe (Abbildung 2.4). Auf den Achsen der Drehencoder wurde jeweils eine Abtriebsrolle(2) befestigt, welche mithilfe einer Feder gegen den Reifen des Rollstuhls(1) gedrückt wurde. Diese Encoderräder wurden mit Gummi ummantelt, um ein Durchrutschen der Räder zu verhindern.

Jeder Drehencoder wurde jeweils von einem Mikrocontroller ausgewertet, welcher über eine Powerbank mit Spannung versorgt wurde. Die Mikrocontroller übermittelten die Werte per WLAN an den verbauten Mini-Computer.

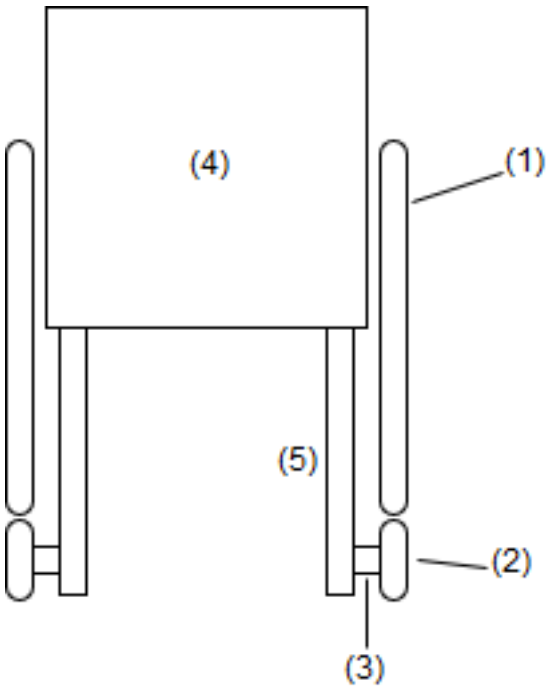

Abbildung 2.4: Skizze des Rollstuhls von oben: (1) Rollstuhlrad, (2) Encoderrad, (3) Encoder, (4) Sitzfläche, (5) Gestell

2.3 Aufbau der Software

Die bisherige Software des Grip-Projekts bestand aus verschiedenen Komponenten, welche auf dem Intel Nuc und dem Mikrocontroller ausgeführt wurden. Für die Kommunikation zwischen den Komponenten auf Softwareebene wurde das Robot Operating System (ROS) verwendet. ROS ist ein Software-Framework für Roboter, dessen Komponenten (auch Nodes genannt) mittels eines Publish-Subscribe-Patterns untereinander kommunizieren. Die Nodes sind in diesem Fall Python-Skripte, welche die Möglichkeit haben, Nachrichten aus Topics zu lesen oder in Topics zu schreiben. Topics sind in diesem Zusammenhang als Message Queues zu betrachten.

Da auf dem ESP32, der die Encoder auswertet, kein ROS installiert werden kann, werden dessen Messwerte in Geschwindigkeiten umgewandelt und dann per UDP an den Nuc gesendet.

Um die Bewegungen der Rollstuhlräder zu registrieren und daraus Steuerbefehle für den Roboter zu machen, wurden zwei ROS-Nodes *rosgripWheelieInterface* und *rosgripDrivecontrol* auf dem Intel Nuc ausgeführt:

- **rosgripWheelieInterface**: Empfängt die UDP-Nachrichten, die vom ESP32-Mikrocontroller kommen. Der Node übersetzt die Geschwindigkeiten in Nachrichten vom Typ *Geschwindigkeit.msg*, welche dann in das Topic *grip_sense* geschrieben werden. Dieser Nachrichtentyp enthält Felder für die jeweiligen Geschwindigkeiten der Roboterräder.
- **rosgripDrivecontrol**: Empfängt Nachrichten aus dem Topic *grip_sense*, welche vom Typ Geschwindigkeit.msg sind. Diese Werte werden nun in zwei Vektoren umgerechnet (einer beschreibt die Lineargeschwindigkeit, der andere die Winkelgeschwindigkeit des Roboters). Dies ist nötig, da das Interface der Pioneer-3DX-Basis nur dieses Format versteht. Nach der Umrechnung werden die Werte in Form einer Nachricht vom Typ *geometry_msgs/Twist* in das Topic *RosAria/cmd_vel* geschrieben.

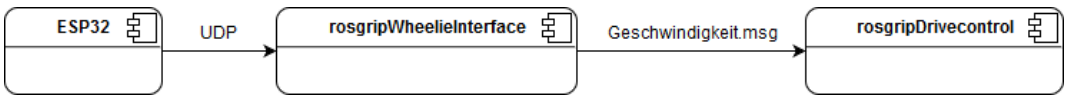

Abbildung 2.5: Verlauf einer Message auf Softwareebene

2.4 Probleme des Projekts

Da das Vorgängerprojekt Grip nicht fertiggestellt werden konnte, hatte das System Fehler, wodurch es nicht funktionsfähig war. Durch Tests der einzelnen Komponenten (Rollstuhl-Sensorik und Roboter-Software) konnten diese Fehler genauer eingegrenzt werden:

Auf mechanischer Ebene gab es Probleme mit den Abtriebsrollen zum Messen der Radgeschwindigkeit. Dabei handelte es sich zwar um Kunststoff-Rollen, welche mit Gummi ummantelt waren und mit einer Feder gegen den Reifen gedrückt wurden. Trotzdem rutschten die Abnehmer bei starker Beschleunigung oder starkem Abbremsen der Räder durch. So war es möglich, dass die Bewegung des Rades nicht auf die Abnehmer übertragen wurde und solche Manöver nicht registriert werden konnten.

Ein weiteres Problem am Rollstuhl war die Software auf den ESP32-Mikrocontrollern, welche die Geschwindigkeit der Räder messen sollten. Der ESP32 verschickte die Geschwindigkeit des jeweiligen Rades per UDP über das Netzwerk. Durch Analyse des Netzwerkverkehrs konnte festgestellt werden, dass die Implementierung fehlerhaft war. Trotz Stillstand des Rades wurden Geschwindigkeiten größer null übermittelt. Auch die Messwerte beim Beschleunigen erschienen unrealistisch hoch, weil teilweise Messwerte Über 400 m/s ausgelesen wurden.

Ebenfalls wurden Fehler bei der Roboter-Software festgestellt. Bei der Konvertierung der einzelnen Geschwindigkeitswerte in die, vom Roboter verstandene Vektorform, ergaben sich falsche Ergebnisse: Während Stillstand (beide Räder haben die Geschwindigkeit 0) und Geradeausfahrt (beide Räder haben die gleiche Geschwindigkeit ungleich Null) korrekt in Vektoren übersetzt wurden, gab es Probleme bei einigen Drehbewegungen des Roboters auf der Stelle. Wenn ein Rad die Geschwindigkeit null, das andere Rad aber eine Geschwindigkeit größer null aufwies, ergab sich ein falscher Ausgabevektor, bei dem sich beide Räder drehten, obwohl eines hätte stillstehen sollen.

Aufgrund dieser Fehler und Probleme wurde deutlich, dass sowohl die Hardware-Komponenten als auch die Software-Komponenten überarbeitet werden müssen, um das Projekt funktionsfähig zu machen.

3 Konzeptionierung eines Human-Computer-Interface auf Basis eines Rollstuhls

Zwar ist der Rollstuhl als Grundlage des Controllers durch die Aufgabenstellung schon vorgegeben worden, dennoch gilt es zu überprüfen, ob ein Rollstuhl die Anforderungen an ein Human Computer Interface überhaupt erfüllen kann.

3.1 Anforderungen an ein technisches Interface

Bei dem hier besprochenen Controller in Form eines Rollstuhls handelt es sich offensichtlich um ein Gerät, welches eine Schnittstelle zwischen Mensch und Roboter darstellt, also um ein Human-Computer-Interface. Um die Qualität eines Interfaces zu beurteilen gibt es laut Markus Schmid 2017, S. 37 f. folgende, grundlegende Kriterien:

- Selbsterklärende Bedienung
- Einfache und intuitive Bedienung
- Ohne Bedienungsanleitung
- Konsistente Bedienung
- Kompatible Bedienung

Dabei unterscheiden die Autoren zwischen unterschiedlichen Abstraktionsgraden eines Interfaces. Bei einem niedrigen Abstraktionsgrad hat der Benutzer eine relativ identische Abbildung der zu bedienenden Gestalt. Bei einem hohen Abstraktionsgrad ist kein echter Zusammenhang zwischen der Interfacegestalt und der zu bedienenden Gestalt zu erkennen. Das hat laut Markus Schmid 2017, S. 80 folgende Vor- und Nachteile:

Hoher Abstraktionsgrad	Niedriger Abstraktionsgrad
hohe Selbsterklärungsfähigkeit (+)	geringe Selbsterklärungsfähigkeit (-)
geringer Lernaufwand (+)	hoher Lernaufwand (-)
hohe Fehlerrobustheit (+)	geringe Fehlerrobustheit (-)
hohe räumliche Kompatibilität (+)	geringe räumliche Kompatibilität (-)
teilweise ungünstige Bedienbarkeit (-)	gute ergonomische Bedienbarkeit (+)
geringe Flexibilität bei Anpassung an Bediensituation (-)	hohe Flexibilität bei der Anpassung an Bediensituation (+)

Als Ideal gilt laut Markus Schmid 2017, S. 80 eine Mischung der beiden Abstraktionsgrade:

- Kombination aus ergonomischer, guter Bedienbarkeit und räumlicher Kompatibilität (+)
- geringer Lernaufwand nötig (+)
- Kriterien der räumlichen Kompatibilität und Bewegungskompatibilität werden noch berücksichtigt (+)

Ebenfalls zu beachten ist die Anwenderzielgruppe. Dabei wird gemäß Markus Schmid 2017, S. 27 ff. zwischen Investitionsgut und Konsumgut unterschieden. Bei einem Konsumgut handelt es sich um ein Produkt, welches für Otto Normalverbraucher ausgelegt ist. Da Otto Normalverbraucher nicht speziell geschult ist, muss das User Interface dementsprechend intuitiv zu bedienen sein. Ein Investitionsgut dagegen ist vor allem für Gewerbetreibende ausgelegt. Diese Investitionsgüter werden von Fachpersonal bedient, wodurch die Bedienung deutlich komplexer sein darf, als bei einem Konsumgut. Dafür werden an solche Maschinen auch höhere Anforderungen gestellt, zum Beispiel eine höhere Präzision als bei einem Konsumgut.

3.2 Steuerung von Telepräsenzrobotern in unterschiedlichen Anwendungsgebieten

Um die allgemeinen Anforderungen an ein Human-Computer-Interface zur Steuerung eines Roboters präzisieren zu können, muss das jeweilige Anwendungsgebiete betrachtet werden:

Telepräsenzroboter werden oftmals an Orten eingesetzt, die für Menschen nicht zugänglich oder gefährlich sind: Zum Beispiel bei der Entschärfung von Bomben oder bei der Aufklärung in Kriegsgebieten. Auch in einsturzgefährdeten Höhlen oder unter Wasser kommen heutzutage Roboter mit Bildübertragungstechnik zum Einsatz [*Tauchroboter RB 300* 2018].

Außerdem erhalten Roboter auch zunehmend Einzug in Privathaushalte, wie zum Beispiel der Telepräsenzroboter Double 2 des Herstellers Double Robotics [*Telepräsenroboter Double* 2018]. Dieser überträgt Kamerabilder bidirektional und lässt sich über eine Tastatur oder einen Tablet-Computer fernsteuern.

Abbildung 3.1: Double 2 Telepräsenzroboter für Videotelefonate [Double Robotics 2018]

An das Interface eines Telepräsenzroboters zur Bombenentschärfung werden natürlich andere Anforderungen gestellt, als an das Interface eines Roboters für Videokonferenzen. Das wird deutlich wenn man die Steuereinheiten solcher Geräte vergleicht: Der erwähnte Double-2-Roboter kann mit vier Pfeiltasten einer herkömmlichen PC- oder Bildschirm-

Tastatur auf einem Tablet-Computer oder Smartphone bedient werden. Industrielle Roboter etwa zur Bombenentschärfung haben dagegen spezielle, unhandliche Fernbedienungen mit vielen Knöpfen und Joysticks (siehe 3.2).

Abbildung 3.2: Steuereinheit eines Polizeiroboters [Lab 2018]

Es wird deutlich, dass es nicht einen einzigen Controller-Typ geben kann, der perfekt zu jedem Anwendungsfall passt und alle genannten Kriterien für ein gutes Interface erfüllt.

3.3 Allgemeine Controllerkonzepte auf Basis eines Rollstuhls

Im Nachfolgenden werden zwei universelle Konzepte zur Steuerung eines Telepräsenzroboters auf Basis eines Rollstuhls vorgestellt – unabhängig von der Beschaffenheit des Roboters. Die Grundidee beider Konzepte ist die Verwendung der Rollstuhlräder als sogenanntes Tangible User Interface (TUI). Ein TUI ist laut Bernhard Reim 2015, S. 635 wie folgt definiert: „Tangible User Interfaces augmentieren die reale, physische Welt, indem digitale Informationen an alltägliche materielle Objekte und Umgebungen gebunden werden.Ïn beiden folgenden Fällen werden die Räder des Roboters an die Räder des Controllers, also des Rollstuhls, gebunden. Da diese Bindung bidirektional erfolgen soll, muss, im Gegensatz zum Vorgängerprojekt, auch ein Rückkanal in Form einer sogenanntem Force-Feedback-Funktion integriert werden.

3.3.1 Statisches, rollstuhlimitierendes Konzept

In diesem Konzept bestimmen die Eigenschaften eines Rollstuhls das Verhalten eines Roboters. Die Steuerung verhält sich so, als wäre man mit einem Rollstuhl an Stelle des Roboters. Das bedeutet zum Beispiel, dass der Widerstand an den Rädern des Controllers größer wird, wenn der Roboter eine Steigung hochfährt, als wenn er sich auf einer Ebene bewegt. Daraus ergeben sich folgende Vorteile:

- Konzept muss nicht an den Anwendungsfall angepasst werden.
- Einfache und intuitive Bedienung, weil die Bedienung eines Rollstuhls den meisten Menschen vertraut ist, oder schneller erlernt werden kann.

Allerdings bringt es auch Nachteile mit sich:

- Konzept kann nicht an den Anwendungsfall angepasst werden (beim Zurücklegen langer Strecken ermüdet der Benutzer beispielsweise sehr schnell, was beim Anwendungsfall eines Aufklärungsroboters nachteilig wäre).
- Usability nimmt ab, je weniger Ähnlichkeit der Roboter mit dem Rollstuhl hat (zum Beispiel muss das Drehen auf der Stelle möglich sein und die Fahrleistungen wie Geschwindigkeit und Beschleunigung des Rollstuhls müssen erbracht werden können).

- Höherer Aufwand bei der Umsetzung, da das Verhalten eines Rollstuhls möglichst exakt reproduziert werden muss. Dieses wird aber durch viele Faktoren beeinflusst (nicht nur die Bewegung der Räder sondern auch Sitzposition und Gewichtsverteilung).

3.3.2 Konfigurierbares, anwendungsspezifisches Konzept

Beim zweiten Konzept wird der Controller durch Software-Parameter an den Roboter angepasst und dadurch auf den jeweiligen Anwendungsfall zugeschnitten. Dabei stehen die zwei Räder des Controllers als universelle Stellteile zur Verfügung, deren Funktion anwendungsspezifisch geändert werden kann. Zum Beispiel könnte ein Rollstuhlrad die Geschwindigkeit regeln, und das andere Rollstuhlrad die Lenkung realisieren. Die Vorteile sind hierbei:

- Konzept kann ideal an den Anwendungsfall angepasst werden (zum Beispiel kann eine Übersetzung der Radgeschwindigkeiten implementiert werden, um lange Strecken mit wenig Kraft zurücklegen zu können).
- Technische Limitierungen des Roboters können berücksichtigt werden (zum Beispiel kann via Force Feedback die maximale Geschwindigkeit der Rollstuhlräder künstlich begrenzt werden)
- Geringerer Aufwand bei der Umsetzung, weil nicht alle Merkmale des physikalischen Verhaltens des Rollstuhls berücksichtigt werden müssen (wie Sitzposition oder Gewichtsverteilung).

Nachteile dieses Konzepts:

- Bedienung des Controllers muss vom Benutzer geübt werden, da unter Umständen nicht intuitiv oder selbsterklärend.
- Bedienung muss immer an den jeweiligen Roboter angepasst werden. Kein klares Standard-Verhalten.

Die beiden Konzepte können den bereits genannten Begriffen Konsumgut und Investitionsgut zugeordnet werden. Für ein Konsumgut scheint das statische, rollstuhlimitierende Konzept besser geeignet zu sein. Dessen Bedienung ist für Otto Normalverbraucher intuitiv

und selbsterklärend (besonders stark zum Beispiel bei Rollstuhlfahrern). Für ein Investitionsgut hingegen würde sich das konfigurierbare, anwendungsspezifische Konzept eher eignen, denn oftmals werden an die Steuerung von Telepräsenzrobotern im professionellen Umfeld (vergleiche Kapitel 3.2) speziellere Anforderungen gestellt als an die Steuerung eines Rollstuhls, zum Beispiel in Bezug auf die Präzision der Bewegungen. Des Weiteren müssen solche Roboter ohnehin häufig von geschultem Fachpersonal bedient werden, weswegen die geringere Intuitivität nicht so stark ins Gewicht fällt.

3.4 Konkretes Controllerkonzept für den Telepräsenzroboter Mory-A

Da der Mory-A das Bewegungsverhalten eines Rollstuhls nicht vollständig abbilden kann, zum Beispiel wegen geringerer Höchstgeschwindigkeit, wird das konfigurierbare, anwendungspezifische Konzept als Grundlage für die Steuerung ausgewählt. Außerdem sind vielfältige Use Cases für diesen Telepräsenzroboter denkbar, weshalb eine konfigurierbare Steuerung von Vorteil ist.

Im Folgenden werden zwei mögliche Steuerungs-Konfigurationen beschrieben:

- **Geringe Geschwindigkeit und präzise Steuerung:** Dafür wird der Controller so angepasst, dass die absoluten Radpositionen des Rollstuhls mit den Radpositionen des Roboters synchronisiert werden. Dabei wird, für eine höhere Präzision, eine Untersetzung implementiert.

- **Hohe Geschwindigkeit und längere Strecken:** Bei dieser Konfiguration fungieren die beiden Räder des Controllers als Geschwindigkeitsregler. Jedes Rad hat eine neutrale Position, dreht man das Rad des Controllers aus der Neutralposition nach vorne, so wird das entsprechende Rad am Roboter auf eine bestimmte Geschwindigkeit beschleunigt und diese gehalten. Dreht man das Rad des Controllers aus der Neutralposition rückwärts, wird das entsprechende Rad am Roboter in umgekehrte Richtung beschleunigt und gehalten. Somit lassen sich die Geschwindigkeiten der einzelnen Räder des Roboters über die stationäre Position der Räder des Rollstuhls mit sehr geringem Kraftaufwand steuern.

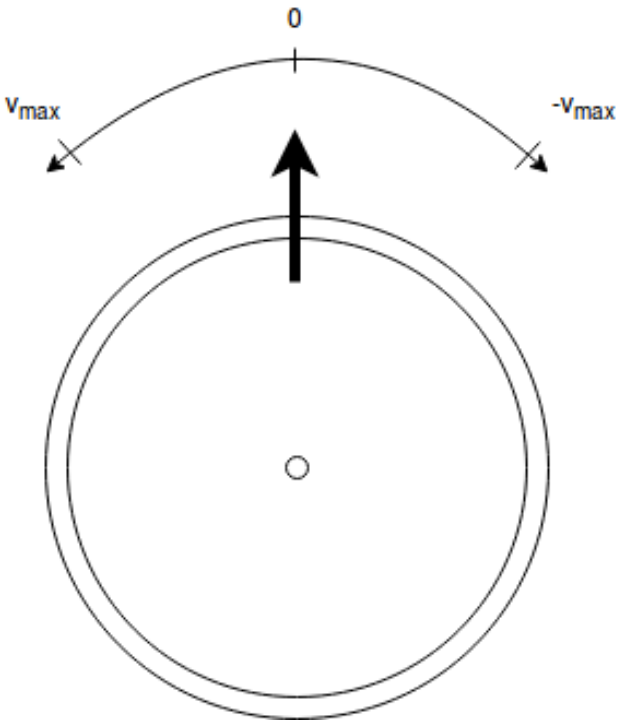

Abbildung 3.3: Rollstuhlrad als Geschwindigkeitsregler

Für beide Konfigurationen ist eine geringe Latenz erforderlich, damit bei der Bedienung auch schnelle Korrekturen und Richtungsänderungen erfolgen können. Mit steigender Größe der Eingabe-Latenz sinkt der Grad der Immersion eines Steuerungskonzeptes. Auch wenn es keine eindeutigen Werte gibt, wird in dieser Arbeit versucht, einen Richtwert von 100 ms nicht zu überschreiten.

4 Mechanische und elektrische Konstruktion

Ziel der folgenden Konstruktionsdetails ist es, Bewegungen der Rollstuhlräder präzise erfassen zu können, sowie bei Bedarf die Räder bremsen und beschleunigen zu können. Da diese Bachelorthesis im Bereich der Technischen Informatik angesiedelt ist, erfolgte die mechanische und elektrische Konstruktion pragmatisch und kurz.

4.1 Allgemeine Konstruktionsmöglichkeiten für Sensorik und Aktorik in einem Rollstuhl

4.1.1 Sensorik

Zum Erfassen der Bewegungen von Rädern gibt es grundsätzlich unterschiedliche Möglichkeiten:

- **Drehencoder:** Diese Komponenten messen die Drehzahl mit einer eigenen Welle. Dafür muss die Bewegung von der Welle des Rades auf die Welle des Drehgebers übertragen werden. Dazu wird entweder ein Zahnrad oder ein Riemen als direkter Abtrieb verwendet oder es wird, wie im Vorgängerprojekt Grip, eine gefederte Abtriebsrolle mit Kontakt zur Lauffläche des Reifens verwendet.
- **Optische oder elektromagnetische Sensoren:** Hall-Sensoren oder Photodioden registrieren die Bewegung mittels am Rad befestigten Merkmalen (zum Beispiel Lochscheiben oder Magneten). Der Vorteil solcher Sensoren liegt darin, dass die physikalischen Kräfte des Rades nicht auf den Sensor übertragen werden. Die Speichen eines Rollstuhlrads eignen sich gut als Haltevorrichtungen für die Sensor-Merkmale, da sie in konstanten Abständen angeordnet sind. Die magnetische Variante wird bei handelsüblichen, elektrischen Fahrrad-Tachometern verwendet. Allerdings ist die

Auflösung dieser Verfahren tendenziell gering und die Fehleranfälligkeit sehr hoch, da magnetische Hall-Sensoren durch fremde Magnetfelder und Photodioden durch Verschmutzungen leicht beeinflusst werden können.

4.1.2 Aktorik

Um die Rollstuhlräder abzubremsen gibt es ebenfalls verschiedene Möglichkeiten:

- **Felgenbremse:** Ähnlich wie bei einem Fahrrad könnte an der Felge des Rollstuhls eine Bremse montiert werden. Diese Bremse müsste elektrisch betätigt werden können, zum Beispiel durch einen Elektromotor oder einen Elektromagnet. Der Vorteil einer Felgenbremse ist eine sehr hohe Bremskraft, die üblicherweise sogar ausreicht um das Rad komplett zu blockieren. Nachteilig ist die Tatsache, dass die Bremskraft elektrisch nur sehr schwer regelbar ist.

- **Elektromotor:** Auch durch den Kurzschluss oder die Gegenkraft eines Elektromotors kann ein Rad gebremst werden. Vorteile dieser Methode sind die gute und einfache Regelbarkeit der Bremskraft, sowie die Tatsache dass mit dem Motor das Rad zusätzlich auch noch angetrieben werden könnte, was neue Einsatzmöglichkeiten eröffnet. Ein Nachteil ist, dass die Bremskraft abhängig von der Leistung der Stromquelle und der Leistungsaufnahme des Motors ist. Die Leistungsaufnahme des Motors wiederum hängt indirekt auch mit der Größe des Motors zusammen. Das heißt, um eine ähnlich hohe Bremskraft wie bei der Felgenbremse zu erreichen, wäre eine deutlich stabilere und aufwändigere Motorhalterung nötig. Außerdem wird die Energie des Rades beim Bremsen innerhalb der Motorspulen in Wärme umgewandelt, was bei zu großer Last zur Zerstörung des Motors führen könnte.

4.1.3 Auswahl konkreter Komponenten

Im Falle des hiesigen Projekts überwiegen die Vorteile des Elektromotors als Bremsvariante, bedingt durch die einfach regelbare Bremswirkung und die neuen Anwendungsmöglichkeiten durch einen optionalen Antrieb der Rollstuhlräder. Da es Elektromotoren auch als kombinierte Einheit mit integrierten Drehencodern gibt, fällt die Wahl der Sensorik ebenfalls leicht. Durch den Einsatz einer solchen Komponente wird der Konstruktionsaufwand reduziert, da keine separate Sensorhalterung benötigt wird. Außerdem ist die Präzision des integrierten Drehencoders einem zusätzlichen, externen Sensor überlegen.

Als Motor wird ein 12-V-Motor[1] des Herstellers CQRobot mit einem Drehmoment von 8,6 kgcm pro Rad eingesetzt, was am Reifen des Rollstuhls einer Antriebskraft von 1,146 kg (∗ 9,81 N/kg) enstprechen würde. Der Blockierstrom wird vom Hersteller mit 6,5 A angegeben. Der Motor besitzt ein Getriebe mit einem Übersetzungsverhältnis von 18,8:1, wodurch er auf eine maximale Drehzahl von 585 min^{-1} beschleunigen kann. In einem Praxistest wurden die Herstellerangaben allerdings verfehlt. Der Blockierstrom betrug maximal 4 A. Die resultierende Antriebskraft am Reifen lag bei circa 0,9 kg (∗ 9,81 N/kg).

4.2 Antriebsvarianten und Einbau des Motors

- **Antriebsrolle:** Antrieb der Reifen mittels einer zusätzlichen Antriebsrolle, ähnlich der Konstruktion bei der Sensorik des Grip-Projekts (siehe Abbildung 2.4). Einen Rahmen zu bauen, der eine ausreichende Reibung zwischen Reifen und Motor sicherstellt, scheint aber sehr aufwändig.
- **Zahnräder:** Diese Art des Antriebs ist sehr unflexibel, da die Räder gut ineinander greifen müssen und deswegen sehr präzise positioniert werden müssen.
- **Kette und Ritzel:** Die Ritzel sitzen dabei auf der Welle des Motors und auf der Rollstuhl-Achse. Dieser Antrieb ist allerdings wartungsintensiv, da die Kette geschmiert und genau gespannt werden muss.
- **Zahnriemen und Riemenscheibe:** Wie bei der Kette sitzen die Riemenscheiben auf der Welle des Motors und auf der Rollstuhl-Achse. Diese Antriebsart ist sehr präzise und wartungsarm und wurde deswegen in diesem Projekt verwendet.

Zur Befestigung des Motors am Rahmen des Rollstuhls kann zum Teil das alte Gestell der Abtriebsrollen mit Drehencoder aus dem Vorgängerprojekt wiederverwendet werden. Der Motor wird auf einer Aluschiene befestigt, wodurch die Spannung des Zahnriemens eingestellt werden kann. Da der Motor eine relativ hohe Drehzahl von 585 min^{-1} bei 12 V leisten kann, wurde eine Untersetzung eingebaut. Die Zahnriemenscheibe an der Rollstuhlachse besitzt 60 Zähne, während die Zahnriemenscheibe am Motor über 15 Zähne verfügt. Daraus ergibt sich ein Übersetzungsverhältnis von 4:1 und eine neue Maximaldrehzahl von 146 min^{-1}.

[1] [*Herstellerseite CQRobot* 2018]

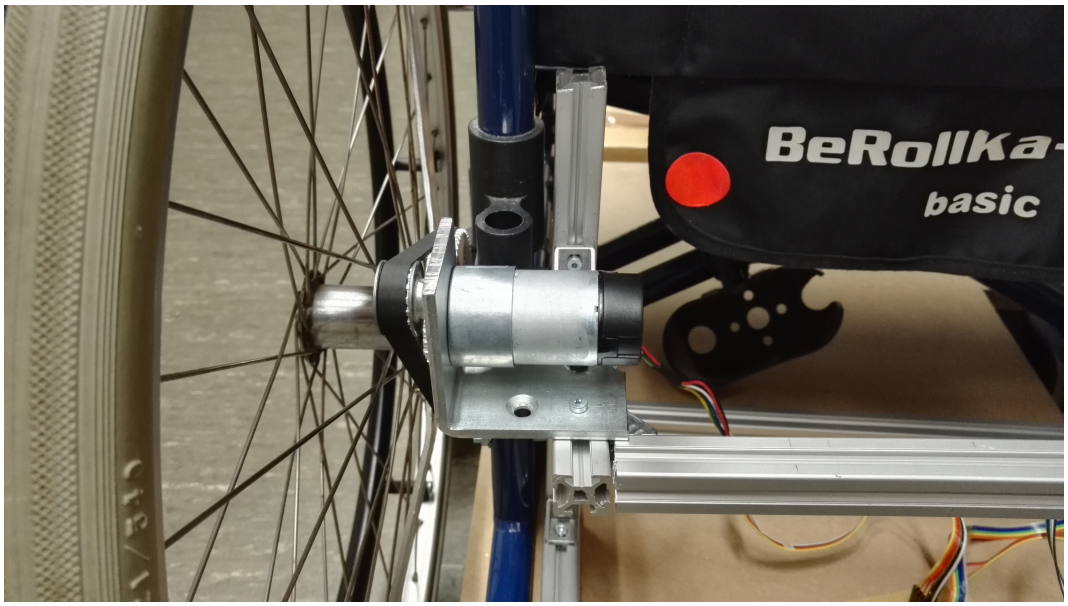

Abbildung 4.1: Halterung des Motors

4.3 Radaufbau

Um eine Zahnriemenscheibe zentrisch mit dem Rad zu verbinden, wird diese auf der Achse des Rades positioniert. Da es sich hierbei um eine feststehende Achse handelt, muss die Zahnriemenscheibe gelagert werden. Die Achse besitzt ab Werk einen besonderen Mechanismus, um die Räder des Rollstuhls ohne Werkzeug demontieren zu können.
Die Achse (1) ist nicht fest mit dem Rollstuhl verbunden, sondern wird durch eine Gewindehülse (5), welche am Rahmen des Rollstuhls festgeschraubt ist, hindurchgesteckt (siehe Abbildung 4.2). Durch zwei federgespannte Kugeln, welche aus der Achse herausragen, wird sichergestellt, dass die Achse (1) nicht aus der Gewindehülse (5) herausrutschen kann. Möchte man die Räder vom Rollstuhl demontieren, so können diese Kugeln in die Achse hineingedrückt werden und die Achse kann durch die Gewindehülse hindurch geschoben werden. Um die Zahnriemenscheibe zu montieren, wird Platz zwischen dem Rad und dem Rahmen des Rollstuhls benötigt. Es ist nicht möglich, die Achse anders zu positionieren, da sonst der Kugel-Sicherungs-Mechanismus nicht mehr funktionieren und die Achse herausfallen könnte. Deshalb muss die Gewindehülse versetzt werden, sodass mehr Platz

zwischen Rahmen und Rad entsteht, und die Zahnriemenscheibe auf der Gewindehülse positioniert werden kann. Da die Gewindehülse einen Achsdurchmesser von 16 mm besitzt und keine Kugellager in dieser Größe verfügbar waren, wird eine Adapterhülse mit einem Innendurchmesser von 16 mm und einem Außendurchmesser von 20 mm angefertigt, da Kugellager in dieser Größe erhältlich waren. Die Zahnriemenscheibe wird über ein Flachblech mit dem Rad verbunden. Dieses wurde so konstruiert, weil es am Rande der Felge bereits Löcher für einen Griff gab.

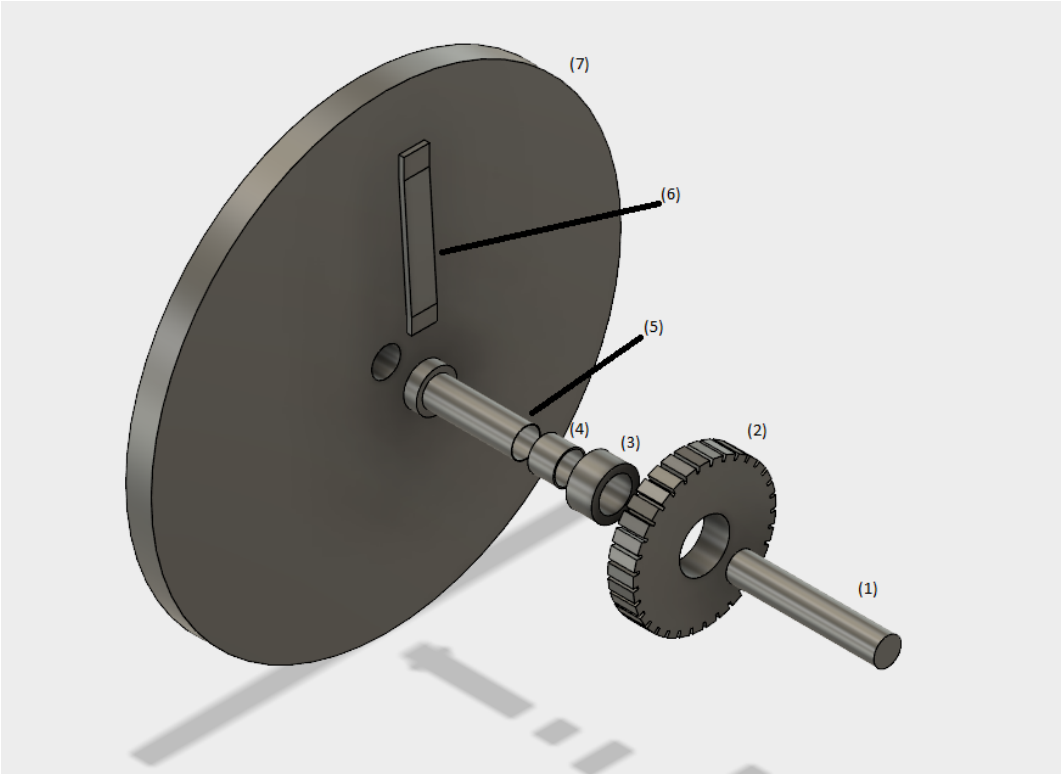

Abbildung 4.2: Explosionszeichnung der Mechanik des Rades: (1) Achse, (2) Zahnriemenscheibe, (3) Kugellager, (4) Adapterhülse, (5) Gewindehülse, (6) Flachblech, (7) Rad

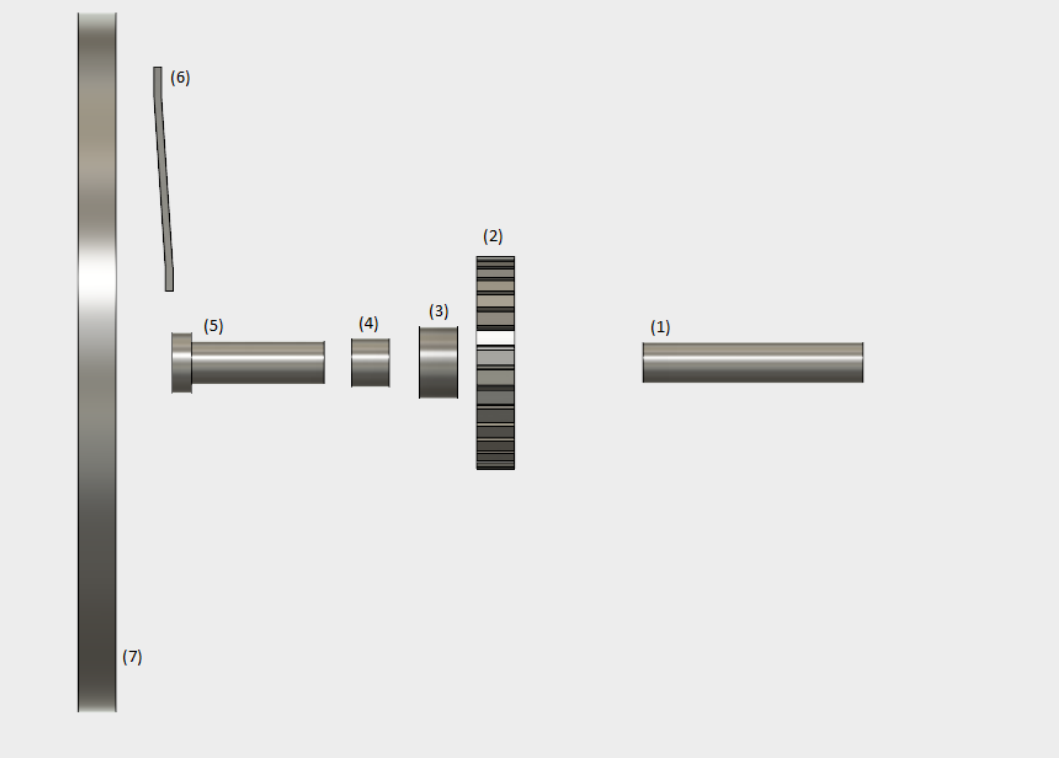

Abbildung 4.3: Explosionszeichnung der Mechanik des Rades von der Seite; (1) Achse, (2) Zahnriemenscheibe, (3) Kugellager, (4) Adapterhülse, (5) Gewindehülse, (6) Flachblech, (7) Rad

5 Elektronische Konstruktion

Wie bereits im Konzept erwähnt, werden auch mehrere elektronische Komponenten benötigt, um die Radposition des Rollstuhls auslesen und verändern zu können.

5.1 Encoder

Zur Steuerung und Messung der Radpositionen werden, wie bereits erwähnt, zwei 12-V-Motoren mit integrierten Drehencodern benutzt. Diese Encoder müssen ausgewertet und die Messwerte regelmäßig über ein Netzwerk an den Nuc-Mini-Computer auf dem Telepräsenzroboter übertragen werden. Für diese Aufgabe wird ein Mikrocontroller-Board auf Basis des weit verbreiteten ESP32-Chipsatzes[1] verwendet, welches per USB mit einer handelsüblichen Powerbank mit Strom versorgt wird. Der ESP32 verfügt über einen Dual-Core-Prozessor und ein integriertes WLAN-Modul und kann somit die Transaktionen über das Netzwerk parallel zur Auswertung der Sensoren ausführen. Außerdem bietet der zugehörige Compiler die Möglichkeit, das Programm in der auf C++ basierten Arduino-Programmiersprache zu schreiben, wodurch der Implementierungssaufwand abnimmt, da es für viele Operationen bereits kostenlose und gute Bibliotheken gibt.

Da die maximale Eingangsspannung für einen ESP32 an den GPIO-Ports 3,3 Volt beträgt, und diese Spannung auch im Bereich der zulässigen Versorgungsspannung der Drehencoder liegt, ist die Stromversorgung sehr einfach zu realisieren. Das ESP32-Board besitzt einen passenden Spannungswandler, welcher 3,3 V Ausgangsspannung zur Verfügung stellt und an den die Drehencoder direkt angeschlossen werden können. Die Drehencoder besitzen jeweils zwei 3,3-Volt-Daten-Ausgänge, auf denen die aktuelle Drehposition der Motorwelle im binären Gray-Code ausgegeben wird. Die zwei Ausgänge der Drehencoder sind direkt mit zwei GPIO-Ports des ESP32 verbinden.

Ab Werk verfügen die Drehencoder über eine Auflösung von 64 Signaländerungen pro Umdrehung. Allerdings bezieht sich diese Angabe auf die Motorwelle, die noch an ein

[1] [ZERYNTH 2018]

internes Getriebe angeschlossen ist, bevor sie aus dem Motorgehäuse heraustritt. Durch Umrechnung mit dem Übersetzungsverhältnis des Getriebes (18,8:1) ergeben sich etwa 1203 Signale des Encoders pro Umdrehung der Getriebe-Ausgangswelle. Anschließend werden die Rollstuhlräder mit dem Zahnriemen im Übersetzungsverhältnis 4:1 angetrieben, so dass sich eine Gesamt-Schrittzahl pro Rollstuhl-Raddrehung von 4812 ergibt.

5.2 Motorsteuerung

Die zwei Motoren können mit bis zu 12 V und einem Blockierstrom von 6,5 A betrieben werden. Ziel ist es, dass die Geschwindigkeit und Drehrichtung der Motoren unabhängig voneinander veränderbar ist. Zur Steuerung wird, analog zur Encoderauswertung, ein ESP32-Board verwendet, da die Steuerung der Motoren ebenfalls über das Netzwerk erfolgen soll. Um die hohe Spannung und den hohen Strom der Motoren mit einem Microcontroller Regeln zu können, wird eine separate Motortreiber-Komponente eingesetzt. Hierbei handelt es sich um ein kommerzielles Dual-Motor-Modul[2], welches Ströme von bis zu 36 V und 15 A schalten kann und auf zwei getrennten H-Brückenschaltungen basiert.

Der Motortreiber benötigt 5 V Versorgungsspannung. Diese wird über einen 5-V-Versorgungspin am ESP32 ebenfalls von der Powerbank bezogen. Zum Steuern der Motoren gibt es am Treibermodul jeweils einen digitalen Eingang für die Drehrichtung, und einen Pin für die Geschwindigkeit, die via PWM-Signal geändert wird. Auch diese Pins sind mit dem ESP32-Mikrocontroller verbunden. Außerdem besitzt das Modul zwei Ausgänge für die Motoren und einen Eingang für die Versorgungsspannung der Motoren, die von einem Labornetzteil versorgt werden. Für den portablen Einsatz wäre auch der Einsatz eines Akku als Spannungsquelle für die Motoren realisierbar.

[2] [*Dual Motortreiber Modul* 2018]

6 Software

In diesem Kapitel wird der Aufbau der Software beschrieben. Dabei wurde der grundsätzliche Aufbau der Komponenten vom Vorgängerprojekt Grip beibehalten, allerdings wurde die Implementierung der einzelnen Komponenten grundlegend erneuert.

6.1 Experimentelle Ermittlung des optimalen Kommunikationsverhaltens des ESP32-Mikrocontrollers

Um ein optimales Kommmunikationsverhalten zwischen den ESP32-Mikrocontrollern und dem Nuc zu erreichen, soll herausgefunden werden, wie sich verschiedene Abstände zwischen UDP-Pakten auf die Latenz des gesamten Controllers auswirken. Es wird hierbei von der Verwendung von UDP-Paketen ausgegangen, da diese eine geringere Netzwerklaufzeit als TCP-Pakete aufweisen. Dieses Experiment hat den Hintergrund, dass es unterschiedliche Semantiken gibt, die Kommunikation zwischen ESP32 und *rosgripWheelieInterface* zu implementieren. Man kann zum Beispiel nur dann ein UDP-Paket senden, wenn sich die Radposition verändert hat, oder wie letztlich implementiert, Pakete in einem festen Intervall verschicken.
Da die Synchronisierung des ESP32-Boards mit einem NTP-Timeserver relativ ungenau ist, ist es nicht ratsam zur Messung der Latenz einen Timestamp zu verschicken und diesen mit dem Timestamp eines Empfängers zu vergleichen. Man könnte damit lediglich eine aussagekräftige Roundtrip-Dauer messen. Deshalb wird ein erweiterter Versuchsaufbau mit zwei ESP32-Boards und einem Oszilloskop vorgenommen. Die Zeitmessung erfolgt innerhalb des Oszilloskops. Die Messung wird ausgelöst sobald ein UDP-Paket beim Sender verschickt wurde, und beendet sobald ein UDP-Paket im Empfänger angekommen ist. Signalisiert werden diese Ereignisse durch zwei Leitungen die mit jeweils einem GPIO-Port der beiden ESP32 verbunden sind.

Pseudocode beim Sender der UDP-Pakete:

```
1    while(1) {
2        digitalWrite(outputTimeMeasurePin1, 1);
3        udp.beginPacket(udpAddress, udpPort);
4        udp.printf("Test Package");
5        udp.endPacket();
6        digitalWrite(outputTimeMeasurePin1, 0);
7        wait(TESTTIME);
8    }
```

Listing 6.1: Pseudocode des Senders

Pseudocode beim Empfänger der UDP-Pakete:

```
1    while(1) {
2        if(inputTimeMeasurePin1 == 1) {
3            uint32_t startTime = micros();
4            digitalWrite(outputTimeMeasurePin2, 1);
5            Udp.parsePacket();
6            digitalWrite(outputTimeMeasurePin2, 0);
7            uint32_t stopTime = micros();
8        }
9    }
```

Listing 6.2: Pseudocode des Empfängers

Bei dieser Implementierung ist zu beachten, dass der Empfänger die Zeitmessung erst dann startet, wenn beim Sender der *outputTimeMeasurePin1* auf High gesetzt wird. Die Dauer der High-Phasen des gelben Signals vom *outputTimeMeasurePin2* bestehen also nicht nur aus der Netzwerk-Laufzeit, sondern auch aus der Dauer zum Schreiben und zum Lesen eines UDP-Pakets. Die blauen Signale stammen vom *outputTimeMeasurePin1* und geben Auskunft darüber, wie lange die CPU des Senders mit den Befehlen zum Senden eines UDP-Pakets blockiert ist. Für das Experiment wurden nun verschiedene Werte für *TESTTIME* eingesetzt, beginnend bei einer Sekunde.
Bei einem Intervall von einer Sekunde wurde festgestellt, dass sich die Verarbeitungsdauer mit jedem neuen Paket steigert. Dabei wird die Verarbeitungsdauer pro UDP-Paket länger, bis nach 4 Paketen wieder die kürzeste Zeit gemessen werden kann. Die Messwerte liegen zwischen 25 (Abbildung 6.1) und 100 ms (Abbildung 6.2).

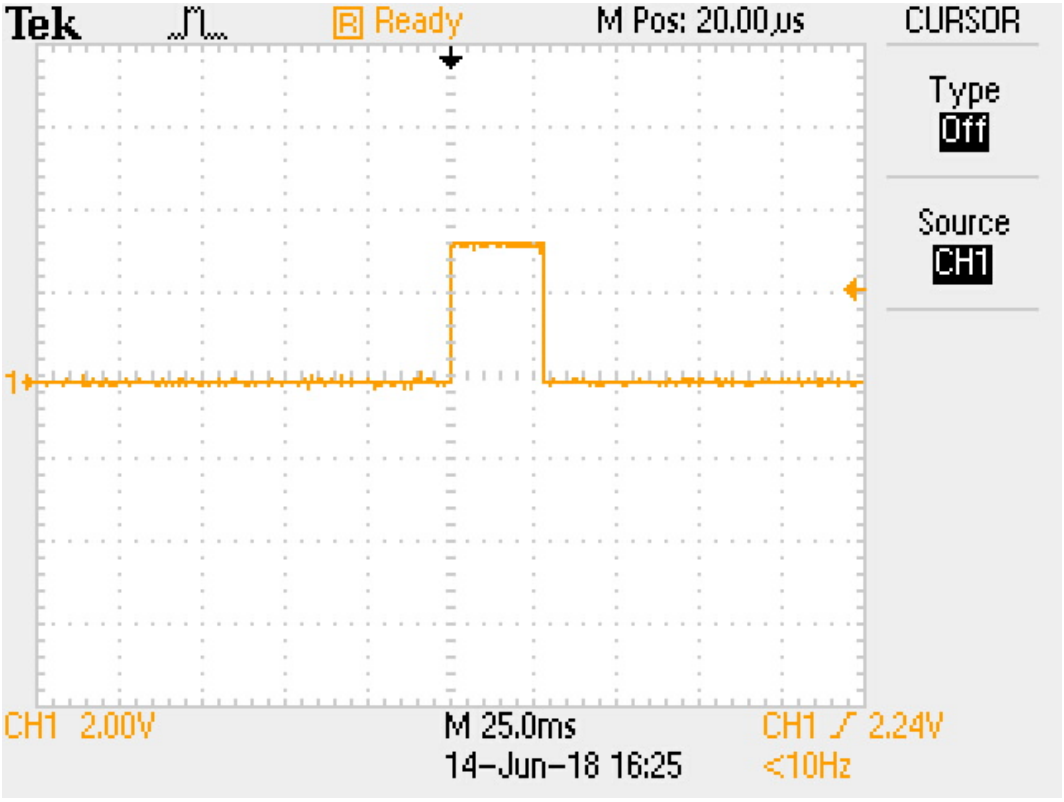

Abbildung 6.1: Kürzeste Verarbeitungsdauer beim Intervall 1 s

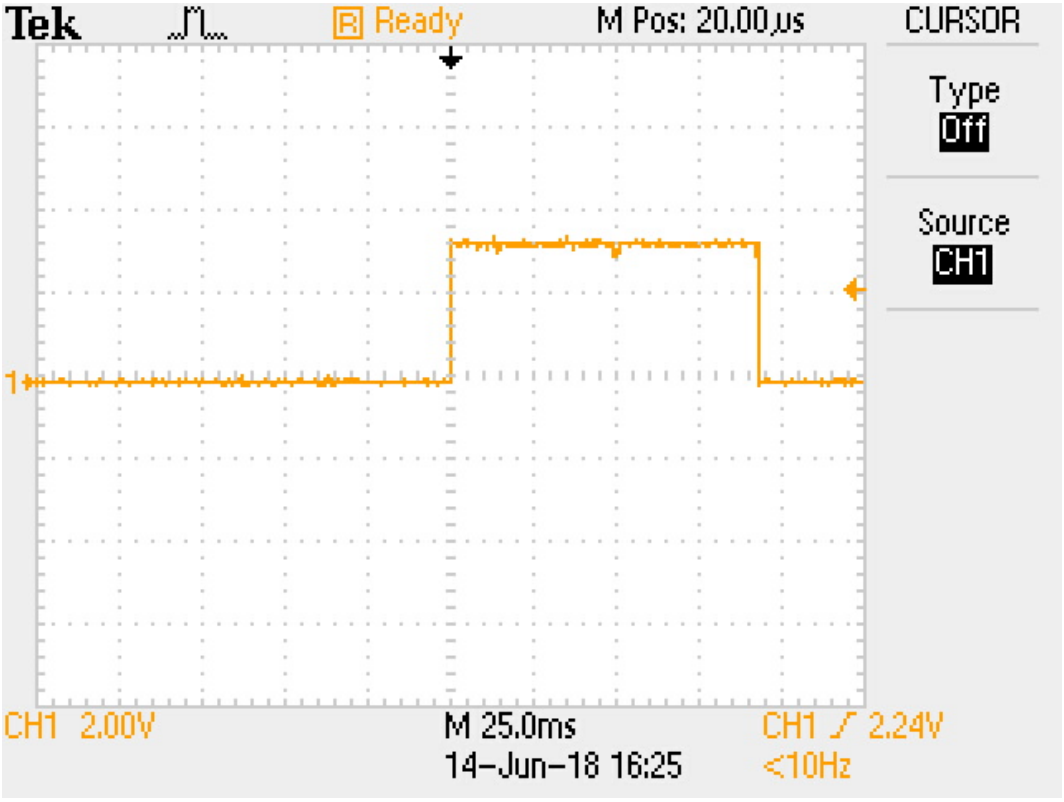

Abbildung 6.2: Längste Verarbeitungsdauer beim Intervall 1 s

Für 100 ms:

Bei einer *STARTTIME* von 100 ms ist die Verarbeitungsdauer bedeutend kürzer als bei einem Wert von 1 s. Sie liegt bei circa 120 μs (Abbildung 6.3). Die Zeit, welche der Sender zum Abarbeiten der UDP-Instruktionen benötigt (circa 430 μs bei allen Messungen), ist in diesem Fall um ein Vielfaches höher als die gesamte Verarbeitungsdauer.

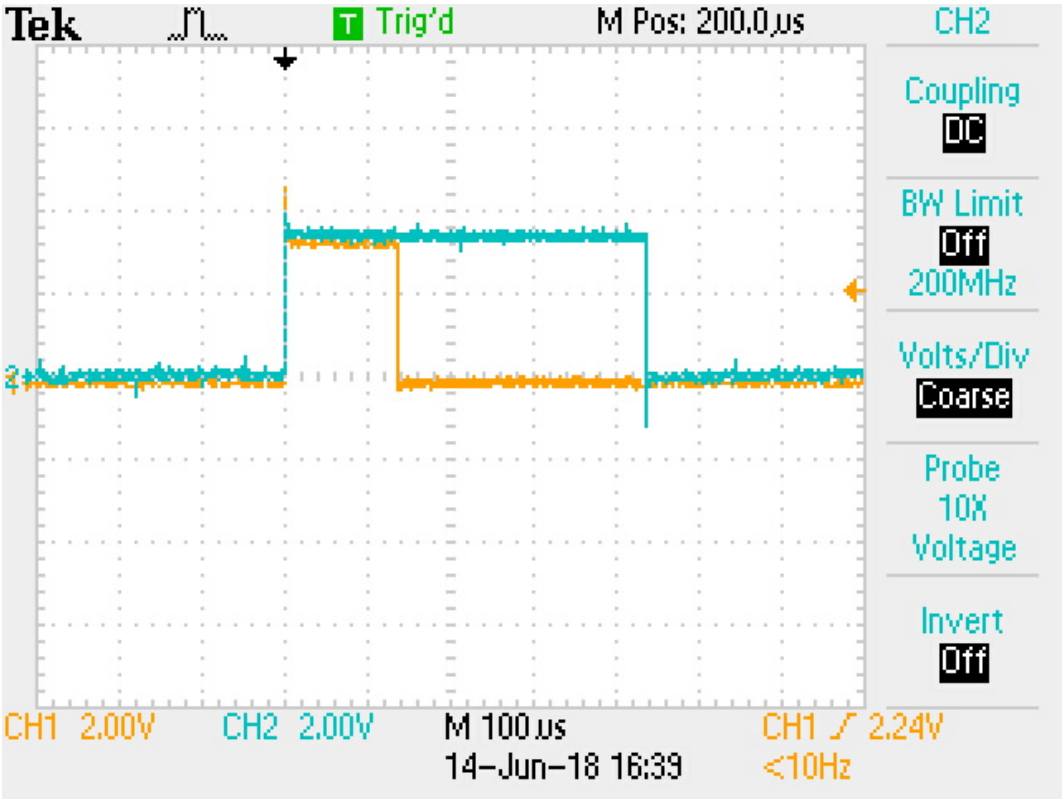

Abbildung 6.3: Verarbeitungsdauer bei einem Intervall von 100 ms

Für 10 ms:

Bei 10 ms nimmt die Verarbeitungsdauer wieder zu: Sie liegt bei circa 2,5 ms.

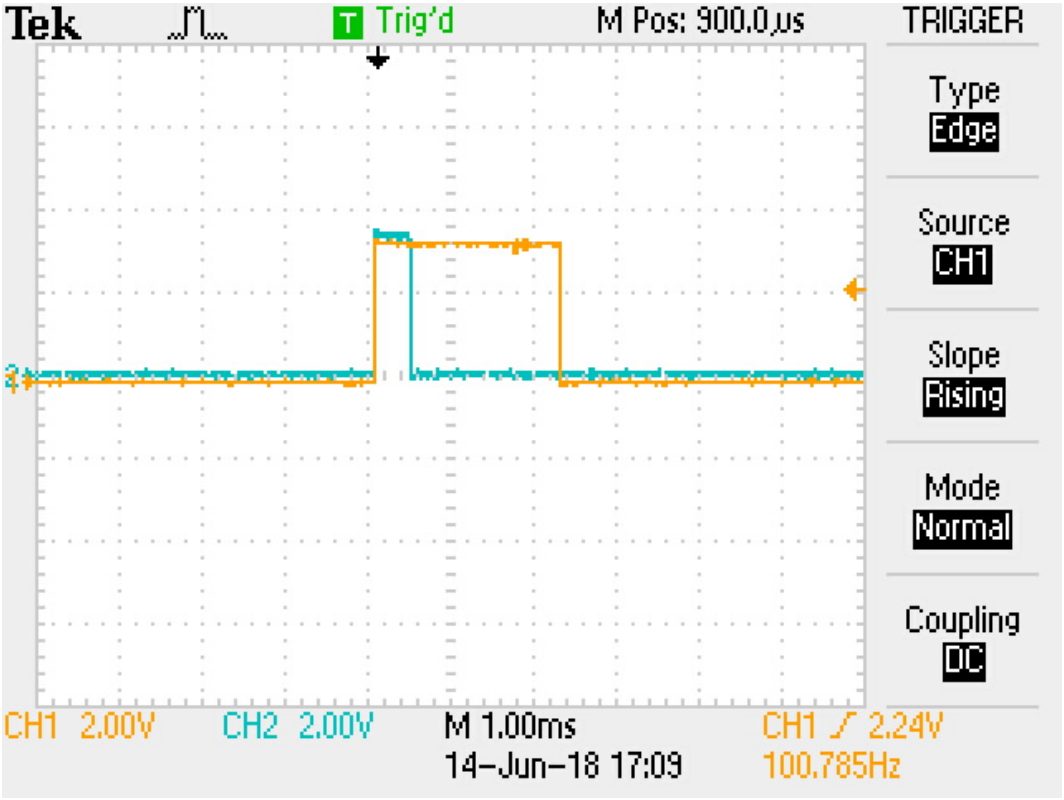

Abbildung 6.4: Verarbeitungsdauer bei einem Intervall von 10 ms

Für 1 ms:

Darüber hinaus wurde getestet, in welchem Intervall die ESP32 die Pakete verarbeiten können, indem ein sehr kleines Intervall von 1 ms ausgewählt wurde. Dabei wurde festgestellt, dass der ESP32 etwa alle 12,5 ms ein UDP-Paket der oben beschriebenen Größe verarbeiten kann.

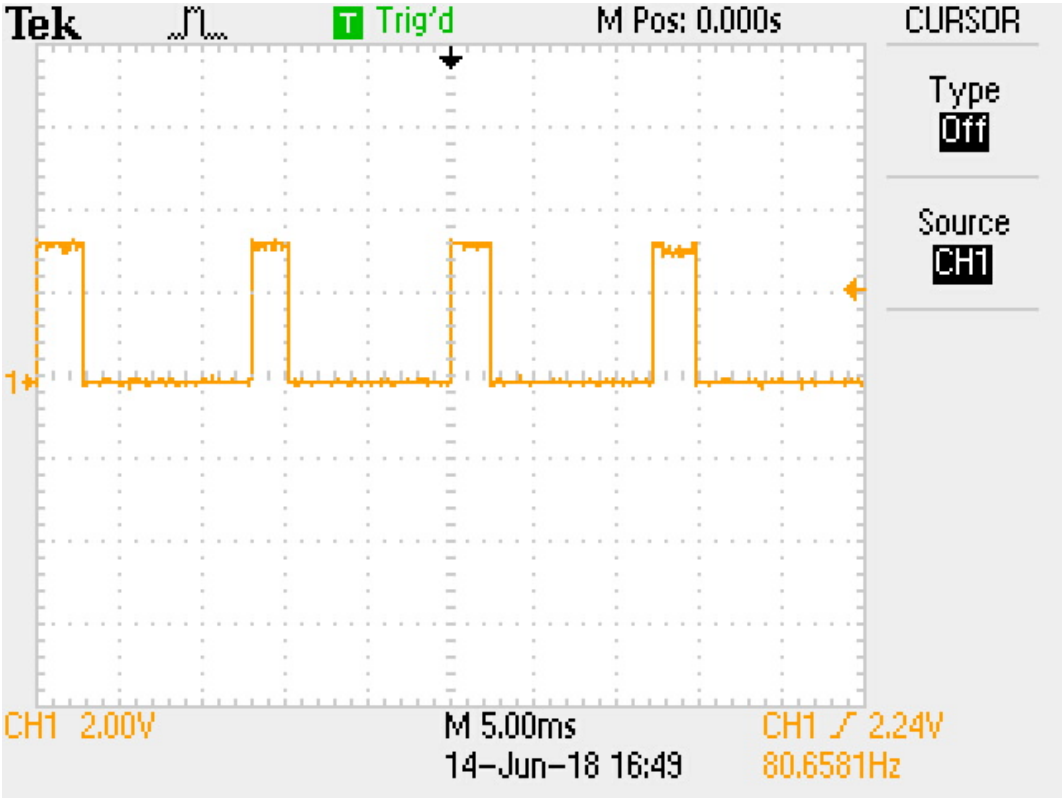

Abbildung 6.5: Verarbeitungsdauer bei einem Intervall von 1 ms

Eine Kommunikation ist in unregelmäßigen Abständen, zum Beispiel nur bei einer Zustandsänderung des Encoders, für einen ESP32 als Sender nicht ratsam, da die Verarbeitungsdauer, bei längeren Pausen zwischen den UDP-Paketen, sehr hoch ist. Die Ursache der langen Verarbeitungsdauer bei großen Intervallen konnte in diesem Versuch nicht festgestellt werden. Wird der ESP32 als Empfänger eingesetzt, so sollten die UDP-Pakete kein kleineres Sendeintervall als 12,5 ms haben, da es sonts zu einem Verarbeitungsstau kommen

kann.Eine regelmäßige Kommunikation mit kurzen Intervallen im Bereich zwischen 30 und 150 ms scheint in beiden Fällen gut geeignet zu sein um eine möglichst schnelle Reaktionszeit des Roboters oder der Motoren am Rollstuhl zu gewährleisten. Für die regelmäßige Nachrichtensemantik spricht außerdem, dass auf der Seite des Roboters ein Verbindungsabbruch schon nach dem ereignislosen Verstreichen eines einzigen Sendeintervalls erkannt werden und der Roboter angehalten werden kann.

6.2 Drehencoder-Auswertung auf einem ESP32

Der ESP32 wurde mit dem C/C++-Dialekt von Arduino programmiert. Der Mikrocontroller verbindet sich zunächst mit dem WLAN-Router und sendet im Anschluss regelmäßig ein UDP-Paket mit den aktuellen, absoluten Positionen der Räder an den Nuc. Bei der Position handelt es sich um einen Zähler, welcher bei einem Signal vom Encoder, je nach Drehrichtung, hoch oder runter gezählt wird. Das Intervall von 50 ms wurde im vorangehenden Abschnitt experimentell ermittelt. Die Signale des Drehencoders werden mit Hardware-Interrupts verarbeitet, damit auch bei schnellen Bewegungen am Rad alle Signale erfasst werden. Da der ESP32 ein speziellen Co-Prozessor für die WLAN-Schnittstelle besitzt, kann man davon ausgehen, dass bestimmte Teile der Netzwerkkommunikation parallel ausgeführt werden und von der Interrupt-Service-Routine nicht unterbrochen werden können.

6.3 ROS-Node zur Auswertung der Sensordaten und Umsetzung verschiedener Steuermodi

Beim *rosgripWheelieInterface* handelt es sich, wie im Vorgängerprojekt Grip, um einen ROS-Node, also um eine Komponente des ROS-Frameworks, die in Python implementiert wurde. Hier werde die UDP-Pakte des ESP32, der die Drehencoder auswertet, empfangen. Im Gegensatz zu Grip wird nun hier auch noch das anwendungsspezifisches Verhalten des Rollstuhl-Controllers konfiguriert. Wie in Abschnitt 3.4 festgelegt, sollen in der Umsetzung zwei verschiedene Steuermodi ermöglicht werden:

- Im ersten Modus soll die Radposition des Rollstuhls direkt auf die Radposition des Roboters übertragen werden. Da die nachgeschaltete Komponente, das *rosGripDriveInterface* als Parameter zwei Geschwindigkeiten (linkes Rad und rechtes Rad)

verarbeiten kann, werden die Positionsdaten der Encoder im *rosgripWheelieInterface* in die nötigen Radgeschwindigkeiten des Roboters umgerechnet und in das Topic *grip_sense* geschrieben. Der Nachrichtentyp konnte hierbei ebenfalls von Grip übernommen werden.

- Im zweiten Modus sollen die Räder des Rollstuhls als Geschwindigkeits-Wahlhebel, ähnlich dem eines Schiffes, benutzt werden können. Somit muss im Node *rosGripWheelieInterface* nicht mehr die Rollstuhl-Rad-Geschwindigkeit aus den einzelnen Encoder-Nachrichten berechnet werden, sondern es muss der Rollstuhl-Rad-Position eine Roboter-Geschwindigkeit zugeordnet werden (zum Beispiel 2,5 m/s bei +80° Rollstuhl-Rad-Position). Da der Node *rosGripWheelieInterface* auf dem Nuc ausgeführt wird, der ein vollwertiges Linux-Betriebssytem ausführt, kann mit geringem Aufwand per Fernzugriff ein anderes Verhalten konfiguriert werden.

6.4 ROS-Node zur Umrechnung der Interface-Parameter des Roboters

Auch bei dieser Komponente wird die Funktionalität des Vorgängerprojekts beibehalten. Der Node empfängt Nachrichten des Typs *Geschwindigkeit.msg*, in der die neuen Soll-Geschwindigkeiten der einzelnen Roboter-Räder enthalten sind und wandelt diese in eine Nachricht vom Typ *geometry_msgs/Twist*, welche Felder für die Lineargeschwindigkeit und Winkelgeschwindigkeit des gesamten Roboters besitzt. Wie bereits in Kapitel 2.4 beschrieben, war diese Umrechnung fehlerhaft und nicht nachvollziehbar, weswegen sie neu umgesetzt wurde. Im Folgenden wird das mathematische Prinzip der Umrechnung beschrieben:

$$
\begin{aligned}
v_r &: \quad Geschwindigkeit\ rechtes\ Roboterrad\ in\ m/s \\
v_l &: \quad Geschwindigkeit\ linkes\ Roboterrad\ in\ m/s \\
v_{lin} &: \quad Lineargeschwindigkeit\ in\ m/s \\
v_{ang} &: \quad Winkelgeschwindigkeit\ in\ rad/s \\
r &: \quad Radius\ der\ Achse\ des\ Roboters
\end{aligned}
$$

Als Rotations-Zentrum der Winkelgeschwindigkeit wird der Mittelpunkt der Roboterachse verwendet. Da bei der Winkelgeschwindigkeit vom mathematischem Uhzeigersinn ausgegangen wird, lässt sich folgende Gleichung aufstellen:

$$\begin{aligned} k: \quad & Umrechnungsfaktor\ von\ rad/s\ in\ m/s \\ v_r = \quad & v_{lin} + k * v_{ang} \\ v_l = \quad & v_{lin} - k * v_{ang} \end{aligned}$$

Wobei für k gilt:

$$\begin{aligned} k = \quad & \frac{U}{2\pi} \\ k = \quad & \frac{2\pi * r}{2\pi} \\ k = \quad & r \end{aligned}$$

Daraus ergibt sich für die Lineargeschwindigkeit folgender Zusammenhang:

$$\begin{aligned} v_r = \quad & v_{lin} + k * v_{ang} \\ v_{lin} = \quad & v_r - k * v_{ang} \\ v_l = \quad & v_{lin} - k * v_{ang} \\ k * v_{ang} = \quad & v_{lin} - v_l \\ v_{lin} = \quad & v_r - (v_{lin} - v_l) \\ v_{lin} = \quad & v_r - v_{lin} + v_l \\ 2v_{lin} = \quad & v_r + v_l \\ v_{lin} = \quad & \frac{v_r + v_l}{2} \end{aligned}$$

Und ähnlich dazu erfolgt die Berechnung der Winkelgeschwindigkeit wie folgt:

$$\begin{aligned} k * v_{ang} = \quad & v_{lin} - v_l \\ v_{ang} = \quad & \frac{v_{lin} - v_l}{k} \end{aligned}$$

$$v_{ang} = \frac{v_{lin} - v_l}{r}$$

In der Implementierung des *rosgripDriveControllers* sehen die Berechnungen wie folgt aus:

```
v_lin = (v_r + v_l) / 2
v_ang = (v_lin - v_linkes_rad) / r
```

Listing 6.3: Berechnung der Winkel- und Lineargeschwindigkeit

6.5 Motorsteuerung auf einem ESP32

Die Motorsteuerung, welche beim Bremsen und Beschleunigen des Rades (also beim sogenannten Force Feedback) zum Einsatz kommt, befindet sich auf dem zweiten ESP32. Sie empfängt UDP-Pakte, die Geschwindigkeit und Richtung für die einzelnen Motoren beinhalten. Die Geschwindigkeit ist dabei als 8-bit unsigned-Variable umgesetzt, da dies der Auflösung der PWM-Bibliothek entspricht. Die Nachricht ist als JSON codiert.

```
{
    [
        "id" : 1,
        "pwm_duty_cycle" : 255,
        "direction" : 0
    ],
    [
        "id" : 2,
        "pwm_duty_cycle" : 100,
        "direction" : 1
    ]
}
```

Listing 6.4: Aufbau einer JSON-Nachricht zur Motorsteuerung

Nachdem ein UDP-Paket empfangen wird, werden die entsprechenden GPIO-Ports, die mit dem Motortreiber verbunden sind, geschaltet und der ESP32 blockiert bis zum nächsten Empfang eines UDP-Pakets. Hier bestätigt sich erneut die Entscheidung, zwei ESP32 einzusetzen. Da das Empfangen von Motorsteuerinformationen und das Auswerten von Encodersignalen parallel passieren muss. Da der ESP32 keine echte Parallelität unterstützt, senkt die Verwendung von zwei Mikrocontrollern den Implementierungsaufwand.

Um ein vollständiges Force-Feedback zu implementieren, müssten, analog zum Encoder-Informationsfluss, auch auf dem Intel Nuc weitere Komponenten implementiert werden, welche Informationen (zum Beispiel ob die Roboter-Räder gerade blockiert sind) vom Roboter abfragen, auswerten und die Ergebnisse an den ESP32 der Motorsteuerung schicken. Das Implementieren dieser Komponenten wurde aus Zeitgründen nicht vorgenommen.

7 Fazit

Da es sich bei den Kriterien zur Bewertung der Qualität eines Human-Computer-Interface nur um sehr weiche Kriterien handelt, ist eine objektive Bewertung des Projektes nicht möglich. Deswegen müssen subjektive Eindrücke zur Bewertung herangezogen werden. Der Telepräsenzroboter Mory-A konnte erfolgreich mit dem Controller gesteuert werden. Dabei sind mit dem Rollstuhl-Controller alle Bewegungsmöglichkeiten des Roboters nutzbar. Die Reaktionsgeschwindigkeit des Systems ist hoch, Verzögerungen sind kaum spürbar. Es ist allerdings bemerkbar, dass die Motoren des Roboters nicht so schnell beschleunigen oder bremsen können, wie die handgetriebenen Räder des Rollstuhls. Dieses Defizit könnte durch die Implementierung eines entsprechenden Force-Feedback-Verhaltens ausgeglichen werden. Die Motoren an den Rädern des Rollstuhls sind stark genug, um Geländeinformationen zu simulieren. Die Leistung reicht allerdings nicht aus, um die Räder langfristig und komplett zu blockieren.
Ebenfalls hat es sich als sinnvoll erwiesen, mehrere Steuerungsmodi zu implementieren. Bei der direkten Übertragung der Radpositionen ist ein präzises Steuern bei langsamen Geschwindigkeiten sehr intuitiv möglich, allerdings ist das Steuern bei hoher Geschwindigkeit schwierig, da es schwer ist, die Geschwindigkeit beider Räder des Controllers synchron zu halten. Dies liegt sowohl am Benutzer, welcher bei steigendem Krauftaufwand die Räder unterschiedlich stark beschleunigt, als auch an unterschiedlichen Reibwiderständen der Räder (bedingt durch Abnutzungserscheinungen und Produktionstoleranzen des Rollstuhls und der Antriebskomponenten). Da der Benutzer beim Versuch, diese Unterschiede auszugleichen, oftmals überkompensiert, fährt der Roboter bei hohen Geschwindigkeiten oft in Schlangenlinien. Auch hier wäre es möglich, diese Schwächen über Force-Feedback auszugleichen, allerdings wäre dies aufwändiger ist, als die Implementierung eines zweiten Steuerungsmodus, der die Räder des Controllers zu Geschwindigkeitsreglern umfunktioniert. Dieses Prinzip ist gut geeignet für lange und schnelle Fahrten. Der Benutzer kann sehr präzise die Geschwindigkeit der einzelnen Räder steuern und benötigt nicht so viel

Kraft und Ausdauer wie beim ersten Konzept. Allerdings ist die Steuerung damit weniger intuitiv und selbsterklärend.

Literatur

Bernhard Reim, Raimund Dachselt (2015). *Interaktive Systeme*. Springer Vieweg. ISBN: 978-3-642-45247-5.

Double Robotics, Inc. (2018). *Double 2 Telepräsenzroboter*. https://www.doublerobotics.com/press.html, Zugriffsdatum: 03.10.2018.

Dual Motortreiber Modul (2018). https://www.amazon.de/gp/product/B06X96MNQC/ref=oh_aui_search_detailpage?ie=UTF8&psc=1, Zugriffsdatum: 21.05.2018.

GmbH, Berollka-aktiv Rollstuhltechnik (2018). http://www.berollka.de/rollstuehle/erwachsene/basic/, Zugriffsdatum: 01.10.2018.

GRIP (2018). *Git Repository*. https://bitbucket.org/JoschkaSondhof/grip/, Zugriffsdatum: 01.07.2018.

Herstellerseite CQRobot (2018). https://www.amazon.com/gp/product/B0756J95TY/ref=s9_acsd_hps_bw_cr_x__a_w?th=1, Zugriffsdatum: 01.06.2018.

Lab, The Machine (2018). *Roboter Steuereinheit*. http://www.themachinelab.com/Custom/Projects.html, Zugriffsdatum: 04.10.2018.

Markus Schmid, Thomas Maier (2017). *Technisches Interface Design*. Springer Vieweg. ISBN: 978-3-662-54948-3.

Seshadri, Swarooph (2018). https://kulkarniisushant1.wordpress.com/author/kulkarniisushant1/, Zugriffsdatum: 25.09.2018.

Tauchroboter RB 300 (2018). http://sandyair.org/blog/rb-mini-300/, Zugriffsdatum: 03.10.2018.

Telepräsenroboter Double (2018). https://www.doublerobotics.com/double2.html, Zugriffsdatum: 04.10.2018.

ZERYNTH (2018). *Datenblatt ESP32-Board*. https://docs.zerynth.com/latest/official/board.zerynth.doit_esp32/docs/index.html, Zugriffsdatum: 01.10.2018.

A. Datenträger

Ordner:

- ESP32
- ROS
- Dummy
- Test_WLAN

ESP32:

- encoderController: Beinhaltet den Code, welcher auf dem ESP32 ausgeführt wird, der für die Auswertung der Encoder zuständig ist.
- motorController: Beinhaltet den Code, welcher auf dem ESP32 ausgeführt wird, der für die Steuerung der Motoren zuständig ist.
- UDPTimeTestRecv: Beinhaltet den Code, welcher auf dem ESP32 ausgeführt wird, der beim WLAN-Test die UDP-Pakete empfangen hat
- UDPTimeTestSend: Beinhaltet den Code, welcher auf dem ESP32 ausgeführt wird, der beim WLAN-Test die UDP-Pakete gesendet hat

ROS:

- msg: Beinhaltet die Definition von Geschwindigkeit.msg
- Nodes: Beinhaltet den Code der aktuellen ROS-Nodes.
- Nodes_old: Beinhaltet den Code der ROS-Nodes bei Übergabe des Projekts

Dummy:

- motorControllerTest: Python-Skript zum Senden von UDP-Pakten zur Steuerung der Motoren
- wheelDummy: Python-Skript zum Simulieren der UDP-Pakte eines Encoders

Test_WLAN: Beinhaltet Bilder der Messungen der Netwerklatenzen

Test_Rollstuhl_old: Beinhaltet eine Wireshark-Exportdatei, welches die Geschwindigkeiten der alten Sensorik beinhaltet

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 5. Oktober 2018 Fabian Schmidt