Roboter Orchestrierungssoftware

Exposé für die Lehrveranstaltung Systemadministration

Stefan Geiring, 30658
Marvin Müller, 32850
Nicolas Baumgärtner, 32849

Wintersemester 22/23
05.November 2022

Betreuer: M.Sc. Aykan D. Inan
Ravensburg-Weingarten University of Applied Sciences

Inhaltsverzeichnis

1 Einleitung

1.1 Thema

Roboter werden immer häufiger eingesetzt egal ob in Spezial Industrial bereichen oder auch einfach Zuhause. Die einfache und übersichtliche Orchestrierung vieler Roboter ist deshalb besonders wichtig. Roboterschwärme spielen außerdem, in der heutigen Zeit immer öfters eine wichtige Rolle. Ein Beispiel für große Roboterschwärme die bereits eingesetzt werden sind Lieferdrohen, Drohnenshows als ersatz für Feuerwerk oder die simplere Variante davon, eine Lagerverwaltung.

1.2 Motivation

Die Hauptmotivation für dieses Projekt besteht darin weitere Teile des ROS-Oekosystems neben dem standardmäßigen ROS1 kennenzulernen.
Das bietet die Möglichkeit der Aneignung neuen Wissens in einem fächerübergreifenden Projekt, das sich über 4 Sektoren: der reinen Softwareentwicklung, Webentwicklung, Hardware/Elektronik-Entwicklung als auch der Robotik Entwicklung/Forschung erstreckt.
Neben der reinen Aneignung von Wissen soll es uns aber auch die Möglichkeit geben weitere Projekte in Zukunft auf diesem Wissen aufzubauen und gemeinsam mit dem Robotiklabor der Hochschule neue Projekte zu erstellen.

1.3 Ziel

Ziel des Projektes soll es sein eine Orchestrierungssoftware für Roboter auf ROS2 Basis zu entwicklen. Mit der Hilfe der Orchestrierungssoftware soll es möglich sein, die Roboter zu überwachen und zu steuern.
Desweitern soll eine Hardware Platform für einen kleinen Beispiel Roboter entwickelt werden, welche im nachhinein durch diverse Komponenten erweitert werden können. Wie zum Beispiel Kamera, Bumper, Laser etc.

1.4 Eigene Leistung

Ein bedeutender Teil unserer eigenen Leistung ist die Entwickung einer einfachen Weboberfläche auf ROS Basis welche als Orchestrierungssoftware für Roboter dienen soll. Die Weboberfläche der Orchestrierungssoftware soll in erster Linie als einfache Kontroll- und Debugging-Schnittstele dienen.
Um unsere Orchestrierungssoftware sinngemäß demonstrieren zu können sollen außerdem kleine Roboter auf ESP32 Basis gebaut werden. Diese Roboter sollen aus einem 3D-Gedruckten Gehäuse bestehen. Auf den ESP32 soll Micro-ROS auf Basis von FreeRTOS ausgeführt werden.
Unsere kleinen Beispiel-Roboter sollen außerdem im Idealfall mit modular austauschbaren Erweiterungen ausstattbar sein. Diese Erweiterungen sollen Simple Sensorik und Aktorik zur Verfügung stellen.

1.5 Aufbau der Arbeit

- ROS in Docker Container lauffähig bekommen.
- Micro-ROS auf ESP32 oder anderer Hardware lauffähig bekommen. Hardware ist noch nicht final definiert, deshalb muss erst noch evaluiert werden ob der ESP32 unseren Anforderungen und Wünschen gerecht wird.
- Frontend für ROS auf Web-Basis programmieren.
- Evaluation des Frameworks für Web-Frontend (evtl. Software zwischenlayer nötig).
- Design des Roboter Gehäuses entwerfen, welches 3D-Gedruckt wird.
- Antriebsstrang des Roboters entwerfen.
- Evaluation und Testing welcher Antrieb unseren Anforderungen entspricht.

2 Grundbegriffe

Docker:

Software für die Container Verwaltung.

ROS:

Das Acronym ROS steht für Robot Operating System, es bietet eine Sammlung von Software-Bibliotheken, Werkzeugen und ein Framework um die Softwareentwicklung und Ausführung von Andwendugen für Roboter zu erleichtern. [ROS, 2022]

React:

Web Frontend Framework was ursprünglich von Facebook ins leben gerufen wurde um deklarativ und Komponentenbasiert User-Interfaces zu entwickeln. [Facebook, 2022]

Arduino:

Entwicklungsplatform auf Basis von Atmel AtMega Prozessoren. Wurde entworfen um Leien den Einstieg in die Microcontroller Welt stark zu vereinfachen. Findet heutzutage weltweit Anwendung in der Maker Szene. [Arduino, 2018]

ESP32:

ESP32 bezeichnet eine Microntroller Familie auf Basis der ARM Architektur. Die ESP32-Familie wurde von der Firma Espressif entwickelt. [Espressif, 2022a]

RaspberryPi:

Single-board Computer der Raspberry Foundation. Der Computer wurde entwickelt um vor allem Kindern das Programmieren und Arbeiten mit Computern näher zu bringen. Raspberry pis werden außerdem häufig für Hobby-IOT und Robotikprojekte auf Grund der geringen Kosten im Verhätnis zur Konnektivität und Rechenleistung.[Raspberry-Pi-Foundation, 2022]

ESP-IDF:

ESP-IDF steht für Espressif IOT Development Framework. Das ist ein Framework mit dessen Hilfe man ESP-Socs programmieren kann. [Espressif, 2022b]

RTOS

Abkürzung für Real Time Operating System. In diesem Projekt wurde in erster Linie das RTOS 'FreeRTOS' verwendet, das für die Verwendung mit Mikrocontrollern optimiert ist. [freeRTOS, 2022]

Micro-ROS:

Micro-ROS wird ein Framework genannt welches es ermöglicht Mikrocontroller in das ROS Oekosystem einzubinden. Mehr dazu im Kapitel Stand der Technik und Forschung. [eProsima, 2022a]

Toolchain:

Eine Toolchain ist eine Kombination mehrerer Softwaretools um komplexe Aufgaben in der Software Entwicklung durchzuführen. In speziellen Fall dieser Arbeit ist es eine

Kombination von Micro-ROS und ESP-IDF. Mit Hilfe dieser Tools wird die Software für den Roboter cross-compiliert, geflasht und konfiguriert.

3 Zielsetzung und Anforderungen

Das Ziel dieses Projektes ist die Entwicklung einer Weboberfläche mit der Roboter auf ROS Basis gesteuert und überwacht werden können. Das Hauptziel dabei ist, dass die Weboberfläche alle Roboter grafisch darstellen kann und simple Werkzeuge zur Verfügung stellt um mit diesen zu kommunizieren und diese zu steuern. Außerdem soll damit eine Schnittstelle zwischen Browsern und ROS geschaffen werden.

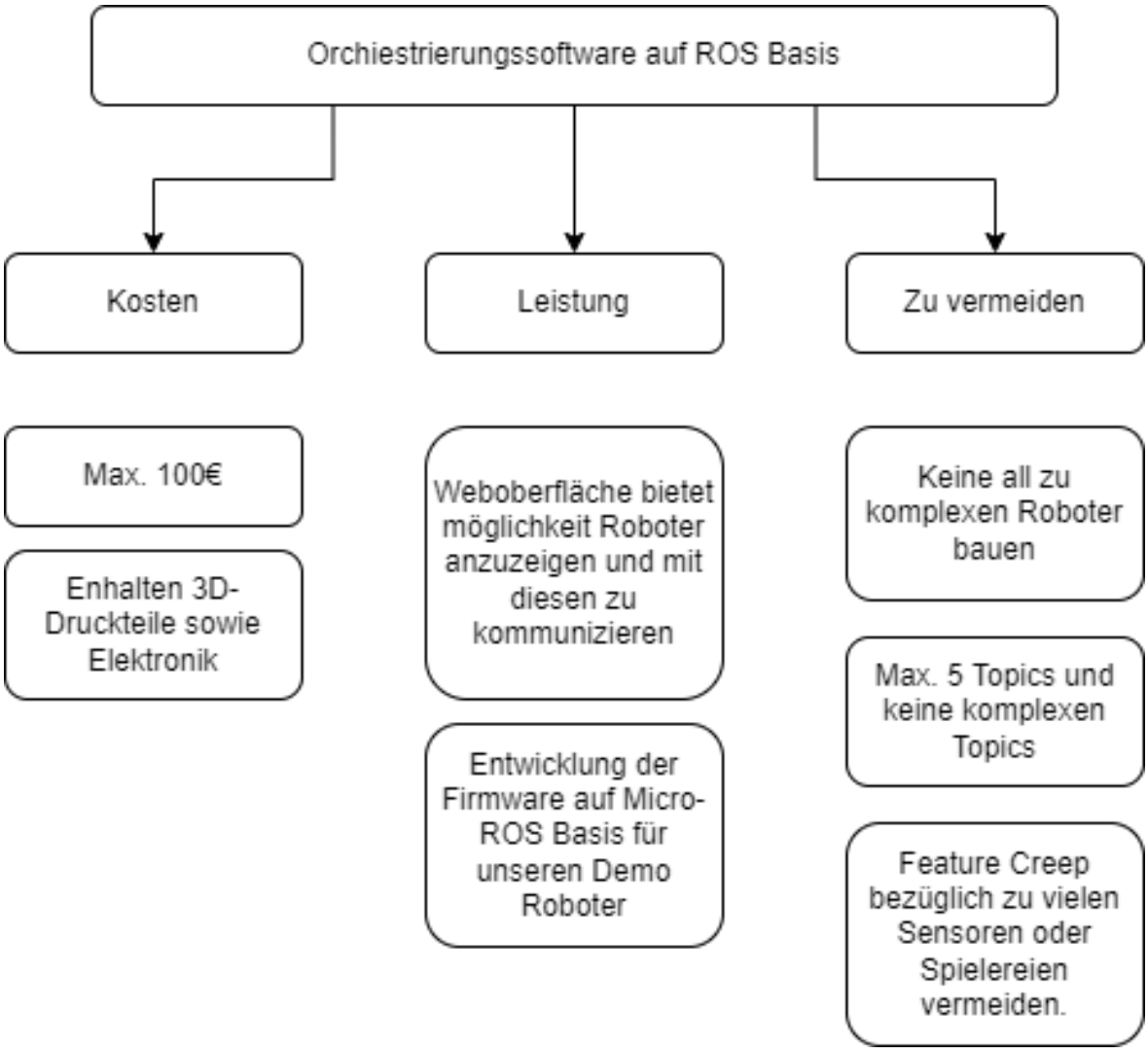


Abbildung 1: Anforderung Diagram

Kosten:

- Endkosten belaufen sich auf weniger als 100€.
- In diesen Kosten soll der Microcontroller inkl. 3D-Druck und gesamter Elektronik inkludiert sein.

Leistung:

- Weboberfläche ist lauffähig.
- Roboter kann mit ROS-Messages gesteuert werden.

- In Eigenleistung kleine fahrbare Roboterplatform auf ESP32-basis kreieren.

Zu vermeiden:

- Keinen zu komplexen Roboter designen. (Vollständig autonom fahrender Roboter)
- Feature creep mit all zu vielen Sensoren gilt es zu vermeiden.
- Roboter Antrieb mit zu viel Technik ausstatten.
- Vorerst sollte nur ein Testroboter entwicklet werden.
- Webinterface nicht zu stark mit ROS Datentypen befüllen.

4 Stand der Technik und Forschung

Technologischer Standpunkt Software:
ROS bietet bereits ein starkes und recht einfach nutzbares Framework, allerdings vermisst man eine schöne und einfach bedienbare grafische Oberfläche. Es existieren kleinere Projekte welche sich dieser Frontend entwicklung annehmen. Allerdings hat uns keines dieser Projekte zufrieden gestellt. Vor allem was die Orchestrierungsmöglichkeit mehrerer Roboter angeht.

Robotik:

Die Robotik beschäftigt sich mit der Entwicklung, Herstellung, Steuerung und dem Betrieb von Robotern. Ein Roboter ist ein universell einsetzbarer Bewegungsautomat, der programmierbar ist und gegebenenfalls über Sensoren verfügt, mit welchen er mit der phsyischen Welt interagieren kann. Bei der Robotik handelt es sich um eine interdisziplinäre Wissenschaft, da sie Gebiete der Informatik, Maschinenbau, Elektronik und der Mathematik vereint. [Konradin, 2022]
Als eines des bekanntesten und meist verbreitetsten Frameworks zur Steuerung von Robotern ist ROS (Robot Operating System), auf das im Folgenden genauer eingegangen wird.

ROS:

ROS steht für Robot Operating System. Das “Operating System” ist kein richtiges Betriebssystem sondern eine Sammlung von Softwarebibliotheken, die es ermöglichen einfach und schnell Robotiksysteme zu bauen. Der Kern von ROS besteht aus Interfaces, genannt ROS-Graph, die eine anonymisierte und standardisierte Interprozesskommunikation ermöglichen. Dieser Graph ist ein Netzwerk aus ’nodes’, welche über ’topics’ miteinander kommunizieren. ’Nodes’ werden als Prozesse auf einem oder mehreren verschiedenen Computern ausgeführt. Auf einem ’topic’ wird immer dieselbe ’message’ mit einem definierten Datentyp von ’nodes’ verbreitet. Für das Verbreiten und Empfangen von Messages müssen die Nodes ’publisher’ und ’subscriber’-Interfaces implementieren. In Abbildung 2 kann man einen ROS-Graph sehen, der aus zwei Nodes besteht, die über ein Topic miteinander kommunizieren. In

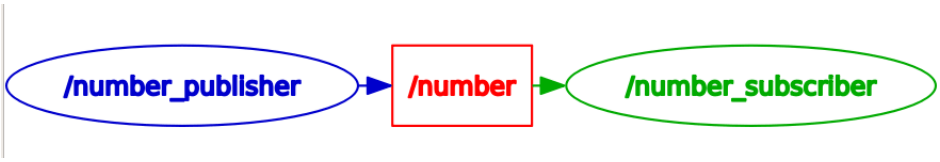


Abbildung 2: Beispiel ROS Graph

diesem Fall gibt es einen publisher der Nachrichten sendet und einen subscriber der Nachrichten empfangen kann.

Des weiteren gibt es noch viele Tools, die wie zum Beispiel in Abbildung 2 den ROS-Graph visualisieren können und eine große Menge an Bibliotheken welche Standard-Algorithmen der Robotik implementieren. Diese Tools und Bibliotheken werden ebenfalls zu ROS gezählt weshalb ROS als weitaus mehr als ein Framework angesehen wird.

Es gibt eine ältere Version von ROS die einfach nur ROS genannt wird und eine neuere Version namens ROS2. Der Hauptunterschied zwischen ROS und ROS2 ist, dass ROS einen zentralen Server, genannt 'ROS-Master' verwendet und ROS2 einen dezentralen Ansatz verfolgt. Der 'ROS-Master' besteht im Kern aus einer Datenbank in der die Teilnehmer der verschiedenen Topics aufgelistet werden und in der die Teilnehmer des Netzwerks ihre entsprechenden Partner finden können, mit denen sie eine TCP-Verbindung aufbauen.

ROS2 baut auf dem Data-Distribution Service Standard von OMG auf. Das ist eine Spezifikation für eine Middleware, welche ein 'Data-Centric Subscriber-Publisher (DCPS)'-Modell beschreibt und somit sehr ähnlich zu dem ROS-Graphen ist. Die Middleware stellt die Peer-to-Peer Funktionalität auf Basis von UDP zur Verfügung.

Micro-ROS

Micro-ROS ist eine Kombination von einem RTOS, einer Client-Bibliothek, einer Middleware und einem sogenannten ROS2 Agent, die Mikrocontroller möglichst effizient in das ROS-Oekosystem einbinden soll.

Mit Micro-ROS ist es nicht möglich ROS-messages direkt im Netzwerk zu versenden und mit anderen Teilnehmern direkt in Verbindung zu stehen. Die dafür benötigte Middleware würde zu viele Ressourcen auf einem Mikrocontroller verbrauchen. Stattdessen gibt es einen sogenannten ROS-Agent, mit dem sich der Mikrocontroller mit Hilfe einer für Mikrocontroller optimierten Middleware verbindet. Der ROS-Agent ist dann im Prinzip eine Brücke zwischen den verschiedenen Middlewares, wie man in Abbildung 3 sehen kann. Entsprechend muss der Agent auch auf leistungsfähigerer Hardware ausgeführt werden. [eProsima, 2022b]

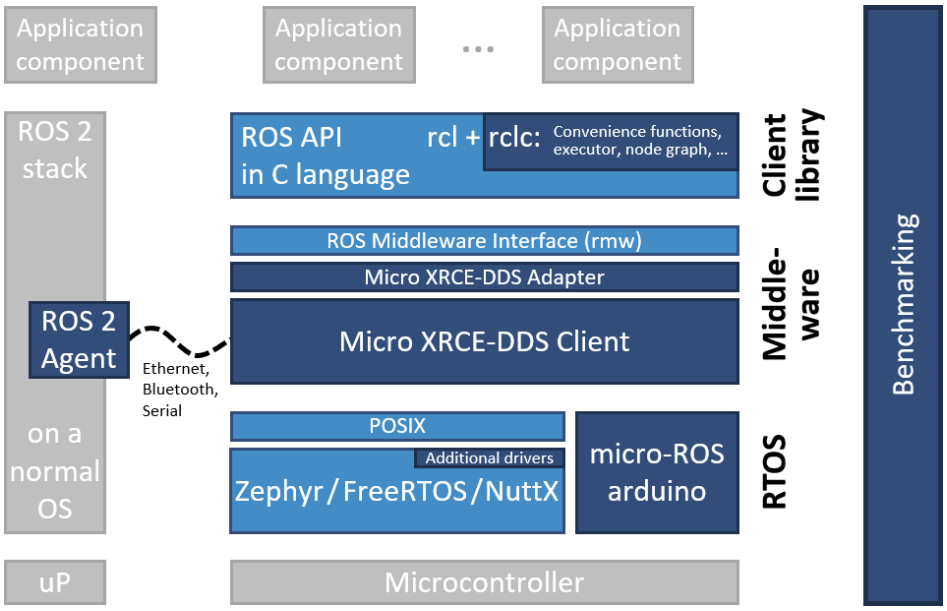


Abbildung 3: Micro-ROS Architektur

Technologischer Standpunkt Hardware:
Es existieren bereits viele Forschungen und Beispiele für diverse Roboter und deren verschiedensten Antriebskinematiken. Für unser Projekt werden wir allerdings eine eigene Roboter Platform designen und 3D-Drucken. Viele der bereits vorhandenen Roboter sind entweder zu Groß und teuer oder viel zu klein und deshalb ebenfalls nicht gut für eine demonstration geeignet. Unter anderem wollen wir das unser Roboter den Vorteil bietet weitestgehend 3D-gedruckt zu sein. Die Universität Brüssel [Goris, 2005] hat bereits eine kleine Zusammenfassung über die wichtigsten Antriebsarten mobiler Roboter zur Verfügung gestellt. Für unser Projekt haben wir uns vorerst für einen simplen Differentialantrieb entschieden.

5 Lösungsideen

5.1 Frontend

Das Frontend könnte durch normales plain HTML, CSS und JS realisiert werden. Da dies aber nicht mehr dem heutigen technischen Standard entspricht kämen viele verschiedene Frontend Frameworks in Frage. Zur Auswahl stehen React, Vue oder Angular.

5.2 Roboter

Zur Demonstration unserer Orchestrierungssoftware soll ein kleiner Roboter gebaut werden. Der Roboter sollte, wie in unseren Anforderungen bereits aufgelistet, einfach und schnell gebaut werden können. Und damit der Roboter auch problemlos in der Welt navigieren kann muss dieser auch mit einer Art Lenkung ausgestattet werden.
Für solch einen Steuermechanismus kommen viele verschiedene Lösungen infrage:
Ein Kettenantrieb ähnlich wie in Baggern wäre denkbar. Er benötigt nur 2 Motoren und bietet viel Bewegungsfreiheit ohne komplizierte Mechanik.

Die Ackermann Lenkung wie Sie heute in allen PKW und LKW vorkommt wäre ebenfalls denkbar. Hierbei müsste man nur einen Motor einbauen für den allgemeinen Vortrieb und einen kleinen Servo um die tatsächliche Lenkbewegung der Vorder- oder Hinterache zu realisieren.
Der Differentialantrieb wäre eine weitere Methode unseren Antrieb zu realisieren, hierbei werden wie beim Kettenantrieb ebenfalls zwei Motoren benötigt. Allerdings entfällt hierbei die Notwendigkeit einer Laufkette.
Unsere letze Idee wäre eine Knicklenkung wie sie oft in Baustellenfahrzeugen eingesetzt wird. Es bräuchte nur einen Motor als Antrieb und die etwas kompliziertere Mechanik von der Ackermann Lenkung könnte durch eine simplere ersetzt werden.

Der Roboter kommt natürlich nicht nur mit einem einfachen Antrieb aus sondern muss diesen auch entsprechend ansteuern können. Ein Arduino könnte sehr gut dafür geeignet sein.
Ein ESP32 käme auch in Frage. Er bietet weitaus mehr Rechenpower als ein Arduino bei minimaler Preiserhöhung. Außerdem unterstützen die meisten ESP32 eine Verbindung über WLAN.
Als teuerste aber auch Leistungsstärke Kontrolleinheit könnte auch ein Raspebrry Pi dienen. Da wir unsere Roboter mit Hilfe von ROS Framework ansteuern wollen, sollte unsere gewählte Steuereinheit Micro-ROS unterstützen.

6 Evaluation der Lösungsideen anhand der Anforderungen

6.1 Evaluierung der Frontend-Frameworks/Bibliotheken

Zur Auswahl standen uns mehrere Frontend-Frameworks, zur Umsetzung unserer Roboter-Orchestrierungssoftware, zur Verfügung. Das erste und wichtigste Kriterium war es hier eine Schnittstelle zu ROS2 zu bekommen, um verschiedenste Topics senden (publish) und empfangen (subscriben) zu können.
Außerdem wollten wir die Webanwendung in einer der bekannteren Webframeworks wie React oder Vue.js umsetzten. Hier stand kein Favorit der beiden im Vorfeld fest, sowie auch wenig bis keine Vorerfahrung. Wir entschieden uns hier für das React Framework, da wir es für sinnvoll hielten, unser Wissens-Repertoire diesbezüglich auch zu erweitern. Ebenso gibt es im Netz mehr Hilfestellungen zu React als zu Vue.js, was bei so einer speziellen Aufgabe wie einer Schnittstelle mit ROS2, schlussendlich von größerem Vorteil ist.
Nach etwas Recherche fanden wir die [Flynn, 2021, React-ROS] Schnittstelle die im npm-Paketmanager zur Verfügung gestellt wurde.
Problem bei diesem Paket war es, das es nicht mit der aktuellen React Version (18) kompatibel war und die Entwicklung hierzu wohl auch schon eingestellt wurde. Man findet auch kaum Erklärungen hierzu, weshalb wir uns nach anderen Möglichkeiten umgesehen haben.
Doch wenn man etwas weiter recherchiert stößt man sehr schnell auf die [RobotWebTools, 2022, roslibjs] Bibliothek, auf die auch das vorher erwähnte Paket aufbaute. roslibjs basiert auf Javascript und verwendet Websockets um sich mit der

[RobotWebTools, 2013, rosbridge] zu verbinden und somit Funktionen wie publishen, subscriben, service calls und vieles mehr zur Verfügung stellt. Die *rosbridge* ist ein ROS Paket, welches eine JSON-API zu ROS-Funktionalitäten für nicht ROS-Programme zur Verfügung stellt.
Auch diese Bibliothek ist mittlerweile schon etwas älter, wird aber trotzdem noch in sehr vielen Projekten verwendet, weshalb wir uns auch für diese Option entschieden. Auch haben wir zu diesem Zeitpunkt keine andere akzeptable Alternative gefunden (hierzu später im Punk zu rosboard mehr), weshalb wir uns für einen Prototypen mit der *roslibjs* Bibliothek entschieden.
Um unserer Anwendung ebenso ein modernes UI zu geben, beschlossen wir eine UI-Komponentenbibliothek zu verwenden. Auch hier standen uns mehrere Optionen zur Auswahl. Zu den Beliebtesten und Bekanntesten zählen hier Material-UI (MUI), Ant Design (AntD) und React Bootstrap. Die Entscheidung fiel uns hier sehr leicht, da schon Erfahrungen im Standard-HTML Bootstrap Framework bestanden, weshalb wir hier auf eine schnellere Einarbeitung in das React Bootstrap Framework hofften.
Nun zusammengefasst verwenden wir:

- React (als Frontend-Framework) [Facebook, 2022]
- roslibjs (als Javascript Bibliothek im Frontend) [RobotWebTools, 2022]
- rosbridge (ein ROS-Paket mit JSON-API) [RobotWebTools, 2013]
- React-Bootstrap (UI-Komponentenbibliothek) [ReactBootstrap, 2022]

6.2 Gründe für die Auswahl von ROS

Da wir alle bereits mit ROS gearbeitet haben, ist die Softwareentwicklung für einen Microcontroller deutlich einfacher, da ROS viele Standard-Aufgaben wie die Interprozesskomunikation erledigt. Außerdem ist es für uns leichter bekannte Hürden zu umgehen. ROS stellt auch eine große Auswahl an Bibliotheken und Tools bereit, die die Entwicklung ebenfalls erleichtern und beschleunigen.
Ein weiterer Grund dafür, dass wir uns für die Entwicklung mit ROS entschieden haben ist, dass wir im Robotiklabor viel Hilfe bekommen können, da dort alle Roboter mit ROS entwickelt werden. Weitere Vorteile von ROS sind, dass es open-source ist und kostenlos.

6.3 Micro-ROS und RTOS

Die Entwicklung mit dem Mikrocontroller findet mit Micro-ROS statt, da das die Standard-Bibliothek für die Entwicklung mit Microcontrollern mit ROS2 ist.
Das Besondere an ROS2 und Micro-ROS im Vergleich zu ROS1 ist, dass es ermöglicht ein Real Time Operating System auf dem Microcontroller auszuführen. Ein RTOS wird standardmäßig mitinstalliert, ist aber für uns auch interessant, da wir bisher noch nicht mit RTOS gearbeitet haben und es eine gute Lernmöglichkeit mit beschränkten Aufwand ist. Wie ROS selbst, ist Micro-ROS open-source und kostenlos.

Außerdem gibt es eine bereits bestehende Toolchain namens ESP-IDF, die das Cross-Compilieren, Flashen, Monitoring und Debugging erleichtert.

6.4 Roboter Platform

Die Anforderungen an die Roboter Demo Platform für dieses Projekt waren recht klein und einfach gehalten.
Das Hauptaugenmerk bei der Neuentwicklung und dem Design unserer Roboter Platform sollte zum einen die 3D-Druckbarkeit des Roboters sein aber auch das der Roboter vor allem in kurzer Zeit gedruckt und betriebsfertig gemacht werden kann.
In unserem Fall zeigten sich für die Antriebsart beim Differentialantreib die meisten Vorteile. Wir benötigen nur 2 Motoren die unabhängig voneinander angesteuert werden müssen. Es können normale Reifen verwendet werden und keine aufwendig zu bauende Ketten. Und beim Differentialantreib wird lediglich eine um 360° frei rotierbare Rolle als Stütze benötigt. Somit entfällt auch ein großer Aufwand bei der Lenkmechanik.
Im allgemeinen sollte die 3D-Druckbarkeit der Teile somit kein Problem darstellen.
Für die Elektronik bzw. die Steuerung unseres Roboter fiel die Entscheidung auf einen ESP32. Dieser ist kostengünstig und recht einfach zu beschaffen. Zusätzlich bietet der ESP32 weitaus mehr Rechenleistung als ein Arduino und garantiert somit auch eine gewisse Skalierbarkeit. Die restlichen elektronischen Bauteile wie z.B. Lineare Spannungswandler können in diversen Elektronik Versandshops gekauft werden.

7 Implementierung

7.1 React Front-End

7.1.1 Erstellung des Prototypes

Der wichtigste Punkt war es nun eine lauffähige Schnittstelle zwischen ROS2 und einer Webanwendung herstellen zu können. Für schnelles prototyping bietet die Open Source Robotic Foundation (OSRF) vorkonfigurierte Docker Container, in jeder beliebigen ROS Version, an. Hierzu muss nur Docker Compose auf dem Rechner installiert sein, und ein fertiges Image kann sogleich erstellt werden. Für ein fertiges ROS Image mit der Foxy Fitzroy Version kann mit dem *docker pull* [10.1] Befehl installiert werden.
Da ROS Nodes über TCP/UDP Sockets kommunizieren, muss nun eine Brücke geschaffen werden, um Daten mit dem Web-Browser austauschen zu können. Hier schafft das [RobotWebTools, 2013, Rosbridge] Paket Abhilfe, in dem es einen Websocket erstellt, welcher in allen gängigen Web-Browsern unterstützt wird. Mit diesem Paket ist es nun möglich die bekannten Publish- und Subscribe-Funktionalitäten der ROS-Umgebung zu nutzen. In der Abbildung 4 wird die eben beschriebene Kommunikation vereinfacht dargestellt, sodass nun mehrere ROS Nodes mit dem Browser interagieren können. Die *Rosbridge* kann ganz einfach über den apt-Paketmanager installiert werden.
Für eine genaue Anleitung um die *Rosbridge* lauffähig zu bekommen, wird auf den Anhang verwiesen [10.1].

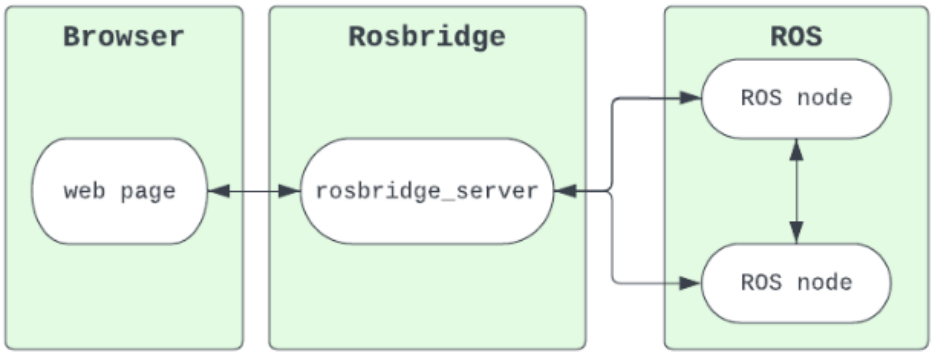


Abbildung 4: Rosbridge Kommunikation [Weon, 2022]

Im nächsten Schritt wird eine einfache Webanwendung gebaut, die mit Hilfe der *roslibjs* Bibliothek nun mit der *Rosbridge* kommuniziert und Daten austauscht. Hierzu wird eine simple HTML-Seite erstellt und *roslibjs* inkludiert.
Wenn nun die *Rosbridge* erfolgreich in der ROS-Umgebung, in diesem Fall innerhalb des Docker Containers unseres Prototypens, gelauncht wurde, kann man sich mit einem Javascript-Objekt eine Verbindung zu dem Websocket aufbauen. Hierzu wird die IP-Adresse des Docker-Containers mit dem Port 9090 benötigt. Mit der URL *ws://171.17.0.3:9090* haben wir uns beispielsweise in unserer Anwendung verbunden. Ist die Verbindung erfolgreich hergestellt, können Publisher und Subscriber an das Objekt angebunden werden. Nachdem eine simple Schnittstelle zwischen ROS2 und einem Web-Browser erfolgreich hergestellt wurde, wird es im Folgenden um die Umsetzung der React-App gehen.

7.1.2 Aufbau des Frontends (Architektur)

Da zuvor noch keine Vorerfahrungen zum Erstellen von React-Apps bestand, erfolgte hier erst eine Einarbeitung in diverse Grundlagen und Funktionalitäten in React. React ist ein Javascript Framework zum Entwickeln von Webseiten und Webanwendungen. Statt einfachen statischen HTML-Seiten wird hier mit sogenannten Komponenten gearbeitet, die mehrfach verwendet werden können. Ebenso können mit States und Hooks reaktive Single-Page-Applications erstellt werden, die ein re-rendern der jeweiligen Komponenten erlauben, ohne ein Neuladen der ganzen Seite. Weitere Pakete und Bibliotheken können mit dem Paketmanager npm ebenfalls jederzeit nachinstalliert werden.
So gibt es auch die *roslibjs* Bibliothek im npm-Paketstore. Installation und Importierung in das aktuelle Projekt wird in [10.1] ausführlicher Beschrieben.
Bezüglich der Architektur, bzw. dem Aufbau der Website, haben wir uns an die standardmäßige Vorgehensweise in React gehalten, in dem man die Seite in Komponenten aufteilt und diese so an mehreren Stellen wieder verwenden kann. Dafür haben wir ein extra Verzeichnis */components* im */src* Verzeichnis angelegt, sowie eines mit */pages* für die jeweiligen Seiten. Diese werden in der *App.js* Datei mit dem *react-router-dom* Paket geroutet.
Wir haben uns für eine linksbündige Navigationsleiste entschieden, da diese mehr zu einem Dashboard und einer Konfigurationsseite passt. Auch hier wurde im ersten Moment nicht

viel Wert auf ein besonders ausgefallenes Design gelegt. Eine schwarze Navigationsleiste mit einem aufklappbarem Burgermenü war hier für uns ausreichend.
Für den generellen Aufbau und dem Design der Website hatten wir uns im Vorfeld schon Gedanken gemacht, so war es uns auf jeden Fall wichtig eine Übersichtseite mit allen Topics zu haben und von diesen die Inhalte auslesen zu können. Im laufe der Wochen hatten wir ein Gespräch mit Benjamin Stähle aus dem RoboLab an der RWU, wir erzählten ihm von unserem Vorhaben und er zeigte uns ein ROS-Webdashboard namens *rosboard*. Diese Plattform hatte quasi die Funktionen, die wir für unsere Webanwendung auch geplant hatten. Zu diesem Zeitpunkt überlegten wir uns, ob wir von nun an diese verwenden, oder unsere eigene Webanwendung programmierten. Natürlich hätte es hier schon alle unsere gewünschten Funktionen zur Verfügung gehabt, allerdings entschieden wir uns dafür, einmal diesen Prozess von Grund auf selber zu entwickeln und unsere eigene ROS-Webanwendung zu erstellen. Wir wollten uns aber vom Design und der Vorgehensweise trotzdem an der vorgestellten Anwendung von *rosboard* orientieren. Ebenso unterscheidet diese sich auch von der Implementierung, da diese auf ganze andere Bibliotheken basiert.

Um nun die Vorteile von React in unserer Anwendung sinnvoll zu nutzen, unterteilten wir das Connection-Handling, Publishen, Subscriben oder auch beispielsweise die Auflistung der aktuellen Topics, in eigene Komponenten auf. So konnten wir mit Hilfe von useState-Hooks und useContext-Hooks, eine Singlepage-Application bauen, die mehrere Verbindungen zu ROS-Instanzen verwalten kann. Für jede offene Verbindung wird ein neues ROSLIB-Objekt angelegt und in ein Array gespeichert. So kann für jedes dieser Objekte eine neue eigene Unterseite erstellt werden, um mit den ROS-Topics zu interagieren. Diese werden ebenso im Dashboard in einer Tabelle ausgegeben (siehe Abbildung 5).

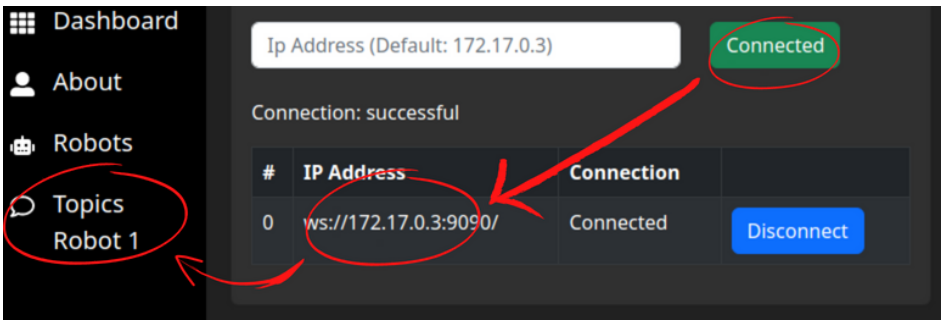


Abbildung 5: Connection Handler mit Liste und automatischem Navigationspunkt

Auf diesen nun generierten Topics-Unterseiten des jeweiligen Roboters, können nun Funktionen darauf angewendet werden. Für das Publishen auf Topics haben wir uns nur auf das */cmd_vel* Topic beschränkt, was einen recht simplem Grund hatte. Denn zum Publishen wird natürlich auch der Message-Type der Nachricht benötigt. Um dann dynamisch auf jedes beliebige Topic senden zu können, müsste der User ebenso, auch den entsprechenden Aufbau der Nachricht haben und diese richtig in ein Textfeld eingeben. Dies wäre sowohl aus User Sicht sehr umständlich und fehleranfällig, da die einzelnen Werte nicht mehr in simplen vordefinierten Textfeldern, sondern in freien Textareas, hätten abgefragt werden müssen. Da das */cmd_vel* Topic, welches für die Fortbewegung und Lenkung des Roboters

zuständig ist, unser definitiv wichtigstes Topic war, beließen wir es dabei. Dieser Nachrichtentyp enthält drei linear- und drei angular-Werte:

```
twist = {
    linear: {
        x: 0,
        y: 0,
        z: 0
    },
    angular: {
        x: 0,
        y: 0,
        z: 0
    }
}
```

Wobei für die Fortbewegung bei Robotern, die sich auf dem Boden mit Rädern bewegen, nur die x-Werte in *linear* und *angular* interessant sind. Hierfür haben wir im Frontend zwei Optionen entwickelt. Zum einen war es uns wichtig fest definierte Werte senden zu können, wofür wir uns für einzelne Textfelder pro Wert entschieden. Diese wurden dann im Code auf das entsprechende Message Format (Twist) gebracht und an die *rosbridge* gesendet. Links in Abbildung 6 sind die Textfelder, mit den jeweiligen Buttons zum Senden und Rücksetzen der Nachricht. Bei Reset wird einfach eine neue Nachricht geschickt, die alle Werte wieder auf Null setzt.

Abbildung 6: Zwei Möglichkeiten um auf */cmd_vel* zu publishen

Ebenso dachten wir uns auch, dass es eine gute Funktionalität wäre, wenn man den Roboter über Tastatur-Eingabe steuern könnte. So erstellten wir noch ein Textfeld, welches beim Fokussieren die Tastenfelder einließt. Dies konnte mit dem Event-Handler *onKeyPress* in React umgesetzt werden. Durch Gedrückthalten der Tasten W, A, S, D, wird der entsprechende linear- oder angular- x-Wert inkrementiert, bzw. dekrementiert. Mit der Taste R wird auch diese Nachricht wieder zurückgesetzt. In Abbildung 6 ist dies auf der rechten Seite zu sehen. Im Anhang in [10.2] wird einmal kurz der Vorgang des Publishen und Subscribens an einem Codebeispiel gezeigt.

Als letztes soll noch auf das Subscriben zu offenen Topics über die Webanwendung beschrieben werden. Über den Button "List Topics", werden alle momentan zur Verfügung stehenden Topics angeordnet. Wie in Abbildung 7 zu sehen ist, öffnet sich unterhalt der Topic-Liste ein Fenster für das jeweilige Topics. Bei mehreren werden diese nebenbei oder darunter automatisch aufgelistet. Innerhalb des Festers werden alle neu ankommenden Nachrichten im JSON-Format gelistet.

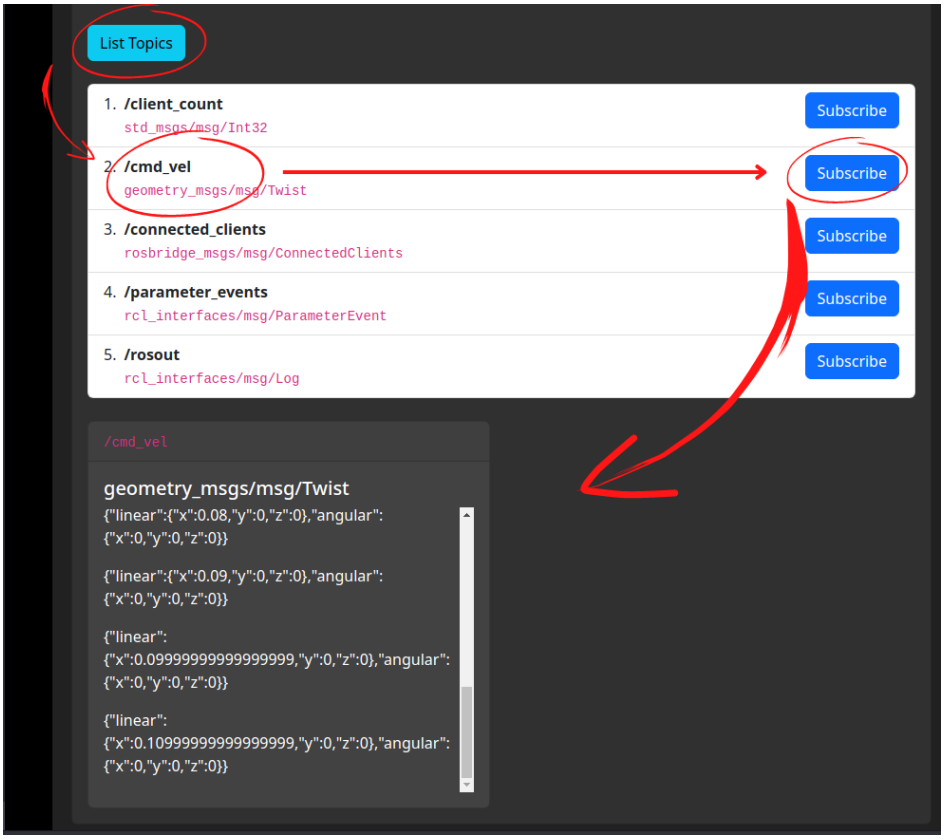


Abbildung 7: Topics Liste und Subscription zu */cmd_vel* Topic

7.2 ROS Back-End

7.2.1 Firmware Kompilierungs Toolchain

Um Programme auf dem ESP32 ausführen zu können müssen sie zuvor crosscompilt und auf den Speicher des Mikrocontrollers geflasht werden. Für das Crosscompilen von Programmen, die das Micro-ROS Framework verwenden wird eine Pipeline empfohlen, die aus 4 Teilen besteht.

1. ROS
2. Micro-ROS
3. RTOS
4. ESP-IDF

Zu Grunde liegt ersteinmal die ROS2 Compilierungspipeline welche colcon verwendet. Darauf aufbauend folgt die Micro-ROS Pipeline. In dieser Pipeline wird die Crosscompilierung für das entsprechende Realtime Operating System übernommen. Des weiteren wird noch die ESP-IDF Pipeline verwendet, die für die Crosscompilierung für den ESP32 zuständig ist und mit deren Hilfe die Programme auf den Mikrocontroller geflasht werden können.
Die Pipeline welche gerade beschrieben wurde, wurde mit Hilfe von Docker implementiert. Als erstes wird ein neuer User inklusive Home-Verzeichnis und entsprechenden Cgroup Berechtigungen angelegt. Für die Berechtigungen ist vor allem die dialout-Gruppe wichtig, da diese Zugriff auf seriellen Ports gibt.
Die Verwendung eines eigenen Users ohne root-Rechte wird verwendet, da es von Docker als Best-Practice empfohlen wird um ungewollte Veränderungen zu verhindern.
Nach dem Aufsetzen des Users wird die Micro-ROS Pipeline heruntergeladen und mit Hilfe von mitgelieferten Skripten installiert.
Nach der Installation müssen noch verschiedene Einstellungen vorgenommen werden. So ist es in eine ersten Schritt notwendig das passende RTOS, in diesem Fall *freertos*, und den Mikrocontroller zu spezifizieren. Anschließend werden Einstellungen wie beispielsweise die Zugangsdaten für das WLAN-Netzwerk eingetragen.
Es ist außerdem noch notwendig den host-user zu den cgroups docker und dialout hinzuzufügen, da es ansonsten nicht möglich ist den Container zu starten oder Programme auf den Mikrocontroller zu übertragen.
Für das erleichterte Ausführen des Containers wird eine docker-compose-Datei verwendet. Der Container wird mit zwei bind-mounts eingerichtet. Einer für das Verzeichnis, in dem der Code des Roboters ist und ein anderer für das Verzeichnis */dev* in dem die Ports für das flashen des Mikrocontrollers sind.
Es werden bind-mounts verwendet, um die Daten und Ports während der Entwicklung aktuell zu halten, damit der Container nicht ständig neu gestartet werden muss, wenn sich eine Datei ändert. Das vereinfacht die Entwicklung deutlich, da der Container beim Ab- und Abstecken des Mikrocontrollers nicht neu gestartet werden muss.

7.3 Roboter Platform

7.3.1 3D-Druckteile

Die Wichtigste Anforderung an die Platform war, dass diese Modular aufgebaut werden kann. Denn dann ist es möglich, auch im Nachhinein neue Teile, Module oder komplett andere Sensoren einfach anzuschrauben, bzw. auch kaputte oder alte Teile problemlos zu erneuern.
Da es sich beim Differentialantrieb nur um eine mechanische Anordnung der Motoren selbst handelt, musste noch ein geeignetes Bauteil als Motor selbst gefunden werden. Die Motoren müssen ein recht hohes Drehmoment, aber eine kleine Umdrehungszahl pro Minute aufweisen. Initial war es deshalb die Idee für normale 5V Motoren, wie sie auch in DVD Laufwerken verwendet werden, ein kleines Getriebe zu konzeptionieren und zu bauen, wie es in Abbildung 8 zu sehen ist.

Abbildung 8: Prototyp Eigenbau-Motorgetriebe

Die Idee bewies sich aber recht schnell als zu schwierig und wurde deshalb fallen gelassen. Als Ersatz für unseren misslungenen Versuch fiel die Entscheidung auf den Modellbau von Getriebemotoren (siehe Abbildung 9).

Abbildung 9: Modelbau Getriebemotor

Ein Anforderung an unsere Demo Roboter Platform war die 3D-Druckbarkeit, sowie deren Modularer Aufbau. Das Fahrgestell unseres Roboters setzt sich aus insgesamt 3 Modulen zusammen. Die Motorenhalterung woran die Motoren selbst und auch die H-Brücke befestigt sind. Die Zentrale Platform für unseren Mikrocontroller. Und die beiden Freirollen an der Front des Roboters.
Die gesamte Roboter Platform wurde mit Hilfe eines iterativen Design-Prozesses gestaltet. Zu Begin wurde nur das Fahrgestell des Roboters designed und gedruckt (siehe Abbildung

10).

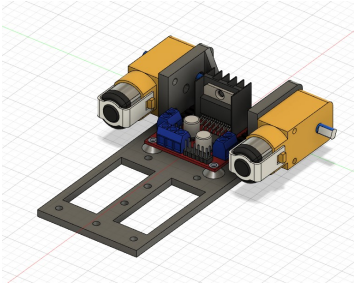
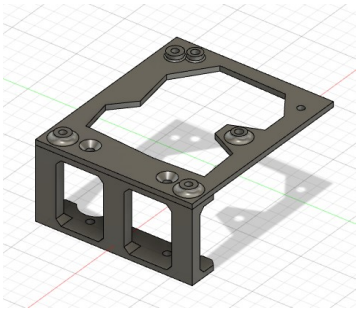
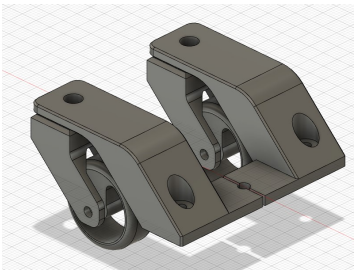


Abbildung 10: CAD Render vom finalen Fahrgestell

Sobald das Ergebnis zufriedenstellend war wurde das nächste Modul designed und gedruckt. Dadurch erreichten wir eine Art Designkette und konnten sicherstellen das alle Teile bzw. Module zusammen passen und allen Anforderungen gerecht werden. Die letzten beiden Module für unseren Demo Roboter waren die Platform für die Elektronik (Abbildung 11 links) sowie die Freilaufrollen (Abbildung 11 rechts) an der Front des Roboters.



(a) Elektronik Platform (finales Design)



(b) Freilaufrollen (2. Version)

Abbildung 11: CAD Render der Elektronik Platform und den Freilaufrollen

7.3.2 Elektronik

Wie in unserer Evaluation der Lösungsideen bereits beschrieben übernimmt der ESP32 Mikrocontroller, siehe Abbildung 12, die Steuerung unseres Roboters.

Abbildung 12: ESP32 Microcontroller

Dieser muss natürlich auch mit Spannung versogt werden. Urprünglich waren sogenannte

18650 LiIon Akkus geplant. Allerdings wurde diese Idee verworfen da Sie doch mit viel Aufwand verbunden war. Inzwischen nutzen wir wiederaufladbare 9V Block Batterien. Die Akkus haben die perfekte Größe für unseren kleinen Roboter.
Die Versorgungsspannung für die Motoren kann direkt von unserem Akkupack abgegriffen werden und zur H-Brücke geführt werden. Die H-Brücke benötigt genau so wie der ESP32 eine Versorgungsspannung für die Logik. Diese sollte laut Datenblatt bei 5V liegen, erlaubt sind aber auch 3V. Was in unserem Fall ideal ist da unser ESP32 ebenfalls nur mit 3,3V maxmimal versorgt werden darf.
Die 3,3V Logikspannung werden auf unserer Platine von einem LM317T, einem Lineareren Spannungswandler zur Verfügung gestellt.

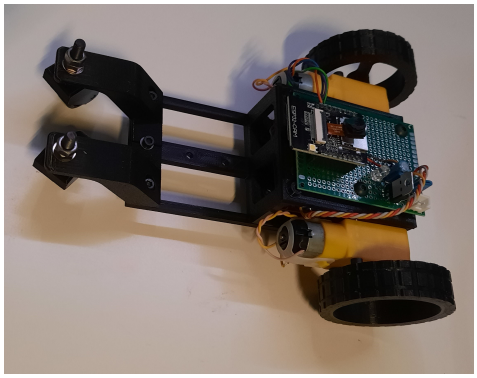


Abbildung 13: Roboter Demo Platform

Da durch die Verwendung der 9V Blockbatterie die Versorgungsspannung der Motoren nun die maximal zulässigen 6V überschreitet, ist ein zweiter Spannungsregler vonnöten, der sich um die Reduzierung der Motorenspannung kümmert. Das ganze hat aber auch einen Vorteil, da wir nun sehr große Freiheit haben was die Akkuspannung angeht. Theoretisch möglich wären nun bis zu 37V.

8 Evaluation der Implementierung

8.1 Fehler und Verbesserungen Frontend

Am Anfang des Projektes planten wir neben dem Publishen und Subscriben der Topics, sowie einer Auflistung der aktuell offenen Topics, noch eine interaktive Konsole. Dies umzusetzen stellte sich aber schwieriger als gedacht heraus. Es gab zwar gewisse Tools wie *Xterm.js*, die so etwas ermöglichen sollten, was aber vor allem in Verbindung mit React nicht zum laufen gebracht werden konnte. Auch gäbe es einige Node.js Funktionalitäten, um Terminal-Befehle ausführen zu können, doch auch diese stehen unter der Verwendung von React nicht zur Verfügung, da es sich hier um eine isolierte Umgebung handelt und somit ein Zugriff auf OS System Prozesse nicht möglich ist.
Aus diesem Grund beschlossen wir die interaktive Konsole in der Webanwendung zu streichen.
Des weiteren hätten wir noch viele spannende Ideen für unsere Webanwendung gehabt. Beispielsweise das Speichern von Konfigurations-Einstellungen für einen Roboter. Hier wäre

auch eine Datenbankanbindung denkbar. Ebenso das Publishen auf beliebige Topics, dessen Nachrichten-Typ der Benutzer nicht kennt. Sowie das Publishen auf benutzerdefinierte Topics, die in der Webanwendung ebenfalls hätten angelegt werden können.
Allerdings hat uns dafür deutlich die Zeit für gefehlt, da allein das Einlernen in React anfangs schon viel Zeit in Anspruch genommen hat.

8.2 Fehler und Verbesserungen Backend

Beim Aufsetzen der Pipeline ist es mehrfach zu Problemen gekommen, die es notwendig gemacht haben, die Implementierung anzupassen.
Zu Beginn wurde erst einmal mit Hilfe einer virtuellen Maschine ausprobiert, ob die Micro-ROS-Pipeline überhaupt funktioniert mit dem ESP32. Das hat gut funktioniert, es hat sich aber herausgestellt, dass die Pipeline sehr viel Speicherplatz benötigt und sie relativ tief ins System eingreift. (TODO Beispiele)
Als nächstes sollte dann die Pipeline mittels Docker umgesetzt werden, um zum einen die Verwendung einer virtuellen Maschine zu vermeiden, aber trotzdem die Virtualisierung beizubehalten.
Für die Virtualisierung mit Micro-ROS mit Docker gab es 3 Möglichkeiten:

1. Die Verwendung der ESP-IDF Toolchain als Extension für VSCode.
2. Die Verwendung von micro-Ros docker Extension
3. Die Erstellung eines eigenen Docker-Images

Zuerst wurde die ESP-IDF Pipeline mit Hilfe einer Anleitung aufgesetzt. Während dem Aufsetzen ist es zu einem Problem mit Docker-Desktop gekommen. Die ESP-IDF-Extension mit Docker funktionieren, allerdings funktioniert sie nur mit dem standard-Context von Docker, der aber von Docker-Desktop verändert wird. Somit war es notwendig Docker-Desktop wieder zu entfernen und zu Docker-Engine zu wechseln.
Mit Docker-Engine war es dann möglich die ESP-IDF-Extension mit VSCode auszuführen und Cdoe auf den ESP32 zu flashen. Die ESP-IDF Extension hat aber den Nachteil, dass es für uns notwendig wird, die ganze restliche Micro-ROS Pipeline selbst aufzusetzen.
Also wurde als nächstes die Micro-Ros docker Extension verwendet, die TODO. Hierbei wurde einmal versucht den debugger zu testen. Grundsätzlich ist der openOCD Server hochgefahren. Allerdings war es danach nicht mehr möglich mit der ESP-IDF-Extension aus dem ersten Schritt auszuführen. Anscheinend hatte die Ausführung dieses OpenOCD-Servers Einstellungen in der Konfiguration von VS-Code docker gemacht, die zur Folge hatten, dass die ESP-IDF-Extension nicht mehr richtig ausgeführt werden kann.
Da sich herausgestellt hat, dass die ersten beiden Methoden nicht zuverlässig waren, wurde schließlich die letzte Methode gewählt.

8.3 Fehler und Verbesserungen Roboter Platform

- Initial sollten kleine 5V Motoren aus alten DVD-Laufwerken mit selbstgebauten Getriebe als Antrieb dienen. Jedoch musste man recht schnell festellen dass das

Design von einem Getriebe doch nicht so einfach ist. Es muss auf den perfekten Abstand der Zähne der Zahnräder geachtet werden. Dass z.B. die Zahnräder konzentrisch und sich leichtgängig drehen und noch vieles mehr. Letztendlich war unser erster Getriebeversuch auch der Letzte. Der Motor hatte einfach viel zu wenig Drehmoment um alle auftretenden Reibungsverluste zu überkommen. Ein netter versuch um das ein oder andere zu lernen war er aber dennoch.

- Der erste Versuch eine Freilaufrolle zu designen und zu drucken funktionierte nur zu 50%. Die Rolle konnte zwar ohne Probleme gedruckt werden, allerdings konnte sie sich nicht frei und leichtgängig genug drehen. Das lag zum einen daran, dass der Rollendurchmesser zu klein gewählt war und der Versatz von der Rolle selbst und dem Drehpunkt zu groß war. Das hatte ein zu hohes axiales Drehmoment am Freilaufdrehpunkt zur Folge. Genau das sorgte für große Reibung und schlussendlich dafür, dass sich die Rolle nicht in Fahrtrichtung ausrichten konnte.

- Wie auch schon oben erwähnt war geplant, den Akku für den Roboter selbst aus 18650 Zellen zu fertigten. Allerdings hätte man dafür noch eine Lade- & Entladeelektronik benötigt und zusätzlich ein Punktschweißgerät. Es war einfach zu viel Aufwand für zu wenig Gewinn. Deshalb entschieden wir uns einfach für die 9V Blockbatterie. (Großen Dank an Dipl.-Ing. Joachim Feßler für die Bereitstellung der Akkus)

9 Fazit und Ausblick

Zusammenfassend kann man sagen, dass wir in diesem Projekt viel Neues lernen konnten und dies auch noch in unterschiedlichen Teilgebieten. So konnten wir einmal den kompletten Aufbau eines Roboters von Grund auf umsetzen und uns in die einzelnen Schritte hineinarbeiten. Neben dem Hardware-Aufbau konnten wir unser Wissen auch deutlich in der ROS-Umgebung vertiefen, was uns für zukünftige Projekte und Arbeiten definitiv zugutekommen lässt, da wir uns auch vorstellen können später einmal im Robotik-Bereich tätig zu sein.
Die Einarbeitung in React und die damit entstandene Webanwendung war ebenfalls sehr spannend, da es echt ein tolles Framework zum Erstellen von Single-Page-Applications ist. Um die Anwendung aber als kompletten Ersatz für die reine Terminal-Benutzung von ROS zu ersetzen, würde dies noch einiges an Aufwand kosten.
Als wichtigsten Punkt zum erwähnen ist jedoch die Verwendung von microROS auf dem ESP32, da dies viele eventuelle spätere Optionen offen hält, Roboter mit einem sehr günstigen Mikrocontroller zu betreiben. Daher freuen wir uns auf weitere Projekte innerhalb dieses Bereichs.

10 Anhang

10.1 Frontend Installationsanleitung

ROS2 Docker Image mit Foxy Fitzroy Version.

```
docker pull osrf/ros:foxy-desktop
```

foxy-desktop kann hier mit jeder anderen beliebigen Version ausgetauscht werden. Mehr Informationen hierzu findet man im entsprechenden Docker Hub:

```
https://hub.docker.com/r/osrf/ros/
```

Rosbridge mit apt-Paketmanager installieren:

```
sudo apt install ros-<ROS_DISTRO>-rosbridge-server
```

Als nächstes muss die Entwicklungsumgebung gesourced werden und anschließend das Launch-File für die *rosbridge* gestartet:

```
source /opt/ros/foxy/setup.bash

ros2 launch rosbridge_server rosbridge_websocket_launch.xml
```

Roslibjs Inkludierung in HTML ohne React:

```
<script
    type="text/javascript"
    src="http://static.robotwebtools.org/roslibjs/current/roslib.min.js">
</script>
```

Verbindung mit rosbridge über roslibjs-Bibliothek:

```
const ros = new ROSLIB.Ros({ url: "ws://"+ipAddress+":9090" });
ros.on("connection", () => {
    // successful connection
});
ros.on("error", (error) => {
    // error in connection
});
```

Nachdem eine neue React Installation durchgeführt wurde, kann mit diesem Befehl die roslibjs Bibliothek in ein bestehendes Projekt installiert..:

```
npm install roslib
```

..und inkludiert werden:

```
import ROSLIB from 'roslib';
```

10.2 ROS-Web Kommunikation Code Beispiel

Um in der Webanwendung mit ROS kommunizieren zu können sind unsere wichtigsten Funktionen das Subscriben und Publishen von Topics. Im folgenden Codebeispiel wird gezeigt, wie mittels Javascript und der *roslibjs* Bibliothek ein Subscriber erstellt wird und mit einem Listener in einer Callback-Funktion Änderungen aus einem Topic ausgibt.

```
const my_topic_listener = new ROSLIB.Topic({
    ros,
    name: topicName,
    messageType: msgType,
});

my_topic_listener.subscribe((message) => {
    const newTopics = [...alltopicslist,
            {name: topicName, content: msgType}
        ];
    setTopics(newTopics);
});
```

In *my_topic_listener* müssen zur Initialisierung sowohl das ROS-Objekt, der Topic-Name, sowie der Nachrichtentyp des angeforderten Topics enthalten sein.
Als Gegenbeispiel soll hier noch das Publishen von Topics gezeigt werden. Hier haben wir uns erst auf das */cmd_vel* Topic beschränkt, welches für die Bewegungsteuerung des Roboters zuständig ist. Hier können lineare Geschwindigkeiten, sowie Einschlagwinkel der Räder übermittelt werden.

```
const cmd_vel_listener = new ROSLIB.Topic({
    ros : ros,
    name : '/cmd_vel',
    messageType : 'geometry_msgs/Twist'
});

var twist = new ROSLIB.Message({
  linear: {
    x: linear,
    y: 0,
    z: 0
  },
  angular: {
    x: 0,
    y: 0,
    z: angular
  }
});
cmd_vel_listener.publish(twist);
```

Abbildungsverzeichnis

Literatur

[Arduino, 2018] Arduino (2018). Arduino. https://www.arduino.cc/. 4

[eProsima, 2022a] eProsima (2022a). Micro-ros. https://micro.ros.org. 4

[eProsima, 2022b] eProsima (2022b). Micro-ros concepts. https://micro.ros.org/docs/concepts/client_library/introduction/. 8

[Espressif, 2022a] Espressif (2022a). Esp-32. https://www.espressif.com/en/products/socs/esp32. 4

[Espressif, 2022b] Espressif (2022b). Esp-idf. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html. 4

[Facebook, 2022] Facebook (2022). React. https://reactjs.org. 4, 11

[Flynn, 2021] Flynn, E. (2021). react-ros. https://www.npmjs.com/package/react-ros. 10

[freeRTOS, 2022] freeRTOS (2022). freertos. https://www.freertos.org/. 4

[Goris, 2005] Goris, K. (2005). Autonomous mobile robot mechanical design. http://mech.vub.ac.be/multibody/final_works/ThesisKristofGoris.pdf. 9

[Konradin, 2022] Konradin (2022). Robotik: Was man rund um roboter wissen muss. https://automationspraxis.industrie.de/robotik/vom-roboter-bis-robotik-grundlagen-und-trends/. 7

[Raspberry-Pi-Foundation, 2022] Raspberry-Pi-Foundation (2022). Raspberry pi. https://www.raspberrypi.org/. 4

[ReactBootstrap, 2022] ReactBootstrap (2022). React bootstrap. https://react-bootstrap.github.io. 11

[RobotWebTools, 2013] RobotWebTools (2013). rosbridge. http://wiki.ros.org/rosbridge_server. 11, 12

[RobotWebTools, 2022] RobotWebTools (2022). roslibjs. https://github.com/RobotWebTools/roslibjs. 10, 11

[ROS, 2022] ROS (2022). Robot operating system. www.ros.org. 4

[Weon, 2022] Weon, E. (2022). Using rosbridge with ros 2. https://foxglove.dev/blog/using-rosbridge-with-ros2. 13, 25