



UNIVERSITÄT AUGSBURG

Fakultät für Angewandte Informatik

## Praktikum für Produktionstechnik

### Entwicklungsprozess eines lenkbaren Fahrzeuges TEGGLA

---

vorgelegt von: Jonas Wilfert: 1541778

Niklas Paprotta: 1543350

Johannes Evertz: 1463672

Marcel Khodabakhsh: 1333430

eingereicht am: 30. 07. 2019

Studiengang: B.Sc. Ingenieurinformatik

Anfertigung am Lehrstuhl: Produktionsinformatik

Fakultät für Angewandte Informatik

Verantwortlicher Professor: Prof. Dr. Ing. Johannes Schilp

Wissenschaftliche Betreuer: M.Sc. Michael Aumüller

M.Sc. Fabian Herzer

M.Sc. Shuang Lu

M.Sc. Paul Haase

## **Kurzfassung**

Dieser Bericht soll als Dokumentation des Entwicklungsprozesses und der technischen Komponenten des TEGGLA dienen. Hierbei handelt es sich um ein Fahrzeugs, welches durch die besonderen Mecanum-Räder mehr Freiheitsgrade hat als ein konventionelles Automobil. Dieses wurde im Rahmen des Praktikum für Produktionstechnik entwickelt und gebaut.

# Inhaltsverzeichnis

<b>1 Einleitung &amp; Motivation</b>	<b>1</b>
<b>2 Anforderungen</b>	<b>2</b>
2.1 Anforderungsliste . . . . .	2
2.2 Lastenheft . . . . .	3
<b>3 Planung</b>	<b>6</b>
3.1 Entwürfe . . . . .	6
3.1.1 Seggway . . . . .	6
3.1.2 Weggchair . . . . .	8
3.1.3 TEGGLA . . . . .	10
3.2 Morphologischer Kasten . . . . .	11
3.3 Online Bestellungen . . . . .	12
3.4 Verwendete Technologien . . . . .	13
3.4.1 Git . . . . .	13
3.4.2 Creo . . . . .	14
3.4.3 PlatformIO . . . . .	15
<b>4 Entwicklung der Schlüsselemente</b>	<b>16</b>
4.1 Übersicht des gesamten Modells . . . . .	16
4.2 Schaltplan . . . . .	17
4.3 BMS und Laden . . . . .	18
4.4 Mecanum . . . . .	19
4.5 Planetengetriebe . . . . .	20
4.6 ESP-32 vs. ESP-8266 . . . . .	25
4.7 User-Interface . . . . .	26
4.7.1 Java (obsolete) . . . . .	26
4.7.2 HTML5 und Controller-Anbindung . . . . .	28
4.7.3 Protokoll . . . . .	30
4.8 Steuerung per (XBox) Controller . . . . .	31

---

4.9 PLA vs. TPU . . . . .	31
4.10 Eierhalter . . . . .	32
4.11 Leichtbau . . . . .	32
<b>5 Zukünftige Entwicklungsmöglichkeiten</b>	<b>33</b>
5.1 Regler . . . . .	33
5.2 Verbesserungen . . . . .	34
5.2.1 Hardware . . . . .	34
5.2.2 Software . . . . .	35
<b>6 Zusammenfassung</b>	<b>36</b>
6.1 Fazit . . . . .	36
6.2 Errungene Erfahrung . . . . .	37
<b>A Quellcode</b>	<b>39</b>
A.1 ESP32 . . . . .	39
A.2 Webpage . . . . .	52
A.3 Java . . . . .	63
<b>B Technische Zeichnungen</b>	<b>89</b>
<b>Literaturverzeichnis</b>	<b>104</b>
<b>Listings</b>	<b>105</b>
<b>Abbildungsverzeichnis</b>	<b>106</b>
<b>Tabellenverzeichnis</b>	<b>108</b>

# 1 Einleitung & Motivation

Im Rahmen des Praktikums für Produktionstechnik an der Universität Augsburg im Sommersemester 2019 wurden lenkbare Fahrzeuge entwickelt und durch die Verwendung von 3D Druckern realisiert. Die Aufgabe des Praktikums bestand darin, ein herkömmliches Ei mit Hilfe des selbst entwickelten Autos unbeschädigt durch einen vom Lehrstuhl definierten Parcours zu transportieren.

In diesem Bericht wird detailliert auf die Planung, Anforderungsermittlung, Systementwicklung und Systemvalidierung eingegangen.

Zusätzlich werden die Schlüsseltechnologien des TEGGLA beschrieben und weitere interessante Teilaspekte wie mögliche Erweiterungs- und Verbesserungsmöglichkeiten diskutiert. Des Weiteren befinden sich im Anhang des Berichts noch alle relevanten technischen Daten, der gesamte Quellcode sowie alle technischen Zeichnungen der verschiedenen Baugruppen.

## 2 Anforderungen

Für den ersten Meilenstein, dessen Abgabe am 12.05.2019 war, musste eine Anforderungsliste und ein Lastenheft abgegeben werden. Eine Liste mit 18 Anforderungen wurde noch am gleichen Tag erstellt und im Laufe der nächsten Woche ausformuliert.

Die Entscheidung, die Abgabefristen und Termine in das Lastenheft aufzunehmen, wurde getroffen, da diese elementar für das Bestehen des Projektes sind, auch wenn sie schlussendlich nicht relevant für das Fahrzeug sind.

### 2.1 Anforderungsliste

w4 > w3 > w2 > w1		Anforderungsliste	Erstellt am 7.5.	
Lfd.	F/W	Anforderung	Änderung	Verantwortlich
1	F	Fahren		Marcel
2	F	Lenken können		Jonas
3	F	Mindestdistanz 4 Meter		Johannes
4	F	Ei transportieren können		Niklas
5	F	Vorpräsentation: 28.5.		Marcel
6	F	Finales Konzept und CAD Modell: 12.6.		Jonas
7	F	Abgabetermin: 23.7.		Johannes
8	F	Abgabe Bericht: 30.7.		Niklas
9	W4	Ei unbeschädigt transportieren		Marcel
10	W3	Leichtbauweise		Jonas
11	W3	Ressourceneffizienz		Johannes
12	W3	Kosteneffizienz		Niklas
13	W3	innovatives Design		Marcel
14	W2	Wartbarkeit		Jonas
15	W2	externe Steuerung oder autonomes Fahren		Johannes
16	W1	schnelles Fahren		Niklas
17	W1	fahren durch unwegsames Gelände		Marcel
19	W1	Modularisierung/Erweiterbarkeit		Johannes

Tabelle 2.1: Anforderungsliste

## 2.2 Lastenheft

### Fahren (F)

Das finale Fahrzeug muss in der Lage sein, von Startpunkt 1 zu einem definierten Endpunkt 2 fahren zu können. Die Art und Weise der Fortbewegung steht dabei nicht im Mittelpunkt, lediglich die fehlerfreie Funktionalität muss gegeben sein.

### Lenken können (F)

Da Fahren allein nicht ausreicht, um die S-förmige Strecke zu absolvieren, muss das Fahrzeug ebenfalls lenkbar sein. Hierfür ist es nötig mit ausreichender Genauigkeit lenken zu können, um nicht den Bereich der Strecke zu verlassen.

### Mindestdistanz 4 Meter (F)

Das Fahrzeug muss in der Lage sein, beladen mit einem Ei, eine Distanz von mindestens 4 Metern zu überwinden, damit das Ei es auch bis zum Zielpunkt schafft und nicht auf halber Strecke stehen bleibt.

### Ei transportieren können (F)

Es wird gefordert, dass das finale Fahrzeug in der Lage ist, ein Ei über eine gewisse Distanz transportieren zu können.

### Ei unbeschädigt transportieren (W4)

Bei dieser Anforderung handelt es sich um eine optionale Anforderung, welche jedoch eine hohe Priorität erhalten hat (W4). Das Ei sollte also am Ende des Projekts unbeschädigt von Startpunkt 1 zu Endpunkt 2 transportiert werden können, um diese Anforderung zu erfüllen.

### Kosteneffizienz (W3)

Da das Budget dieses Projektes begrenzt ist, ist darauf zu achten, dass das Fahrzeug sehr kosteneffizient konstruiert und produziert wird.

## **Ressourceneffizienz (W3)**

Aufgrund der Vermeidung zusätzlicher Kosten, der limitierten Baumaterialien und der Vermeidung eines größeren Zeitaufwandes soll das Projekt möglichst ressourceneffizient geplant und umgesetzt werden. Bei dieser Anforderung handelt es sich um eine optionale Anforderung, welche jedoch eine hohe Priorität erhalten hat (W3).

## **Leichtbauweise (W3)**

Diese Anforderung ist ebenfalls optional mit recht hoher Wichtigkeit (W3). Bei der Entwicklung und Konzipierung des Fahrzeugs sollte auf ein möglichst geringes Gewicht geachtet werden.

## **Innovatives Design (W3)**

Bei dieser Anforderung handelt es sich um eine optionale Anforderung, welche eine relativ hohe Priorität erhalten hat (W3). Zusätzlich zur grundlegenden Funktionalität des Fahrzeugs wird auch noch Wert auf das optische Design des Fahrzeugs gelegt sowie die Art und Weise der Fortbewegung bewertet. Hierbei werden insbesondere kreative und einzigartige Denkansätze wertgeschätzt.

## **Externe Steuerung oder autonomes Fahren (W2)**

Um das Fahrzeug sicher durch den Parcours zu navigieren, benötigt es entweder eine externe Steuerung (z.B. Fernbedienung + IR Empfänger, Pfeiltasten der Tastatur + WLAN-Verbindung etc.) oder es muss in der Lage sein, völlig autonom (durch Sensorik und „künstliche Intelligenz“) den Parcours zu meistern.

## **Wartbarkeit (W2)**

Bei dieser Anforderung handelt es sich um eine optionale Anforderung, welche jedoch eine relativ niedrige Priorität erhalten hat (W2). Durch Achtung auf Wartbarkeit und Reparaturfreundlichkeit des Fahrzeugs kann besser auf unvorhergesehene Fehlfunktionen reagiert werden.

## **Modularisierung/Erweiterbarkeit (W1)**

Für die Aspekte „Kosteneffizienz“, „Ressourceneffizienz“ und „Wartbarkeit“ spielt die modulare Entwicklung und Konstruktion eine große Rolle. Durch die Modularisierung kann das Fahrzeug

leichter überarbeitet, erweitert und gewartet werden. Des Weiteren ist eine zukünftige Erweiterung des Projekts dadurch leichter umsetzbar. Bei dieser Anforderung handelt es sich um eine optionale Anforderung, welche eine niedrige Priorität erhalten hat (W1).

## **Schnelles Fahren (W1)**

Da in der Logistikbranche Zeitdruck ein wesentlicher Faktor ist, besteht der Wunsch, dass das Fahrzeug die zurückzulegende Strecke in möglichst kurzer Zeit bewältigt und sich deshalb mit hohem Tempo fortbewegt.

## **Fahren durch unwegsames Gelände (W1)**

Das finale Fahrzeug kann sich im Optimalfall nach dem Abschluss des Projekts auch auf unwegsamen Geländen effizient und zielsicher fortbewegen. Diese Anforderung ist nicht explizit durch den Auftraggeber vorgegeben, würde jedoch einen Mehrwert des Produkts erzielen. Daher handelt es sich um eine optionale Anforderung, welche eine niedrige Priorität erhalten hat (W1).

# 3 Planung

Nachdem das Lastenheft abgegeben wurde war der nächste Meilenstein der Abschluss der Konzeptphase, zu dem eine kurze Präsentation über Gesamt- und Teilfunktionen, Lösungsprinzipien und Morphologischer Kasten gehalten werden sollte. Außerdem sollte die Präsentation pro Gruppenmitglied ein vorläufiges Konzept mit Freihandskizze enthalten.

## 3.1 Entwürfe

Da die Entscheidung für das finale Konzept sehr früh gefallen ist, wurde sich darauf geeinigt, dass zwei Gruppenmitglieder dieses Konzept bearbeiten, und dabei viel tiefer ins Detail gehen. Die übrigen zwei Gruppenmitglieder haben jeweils eine eigene Idee behandelt, blieben dabei aber deutlich oberflächlicher.

Insgesamt sind dabei die drei im Folgenden vorgestellte Konzepte entwickelt worden.

### 3.1.1 Seggway

Das Konzept des (S)Eggways (Abb. 3.1) ist, wie der Name schon verrät, inspiriert durch die immer populärer werdende Fortbewegungsart des Segways. Dieses Konzept würde viele Vorteile mit sich bringen: Das Gesamtgewicht des Seggways ist relativ gering, da es keine massiven Bauteile gibt und technische Bauteile wie das Breadboard gleichzeitig mehrere Zwecke erfüllen (Stabilisierung und Verbindung).

Zusätzlich könnte der Seggway einen sicheren Ei-Transport durch eine passende Regelung sicherstellen, da diese ermöglicht, dass auch bei starken Steigungen und unebenen Strecken immer das Gleichgewicht beibehalten beziehungsweise wiederhergestellt werden kann. Das gelingt durch das Prinzip des inversen Pendels.

Die Steuerung des Fahrzeugs erfolgt über das eingezeichnete Bluetooth-Modul, welches die erhaltenen Daten an den Arduino weitergibt. Der Arduino verarbeitet die Daten und steuert anschließend (falls nötig) die zwei Stepper-Motoren an.

Die Stepper-Motoren sind für diese Anwendung ideal geeignet und können auch direkt ohne Getriebe mit den zwei Reifen verbunden werden. Das Gyroskop dient dem Messen von Geschwindigkeit, Beschleunigung und aktuellem Winkel.

Der Eierhalter ist genau so konzipiert wie in dem realisierten TEGGLA Modell, nämlich werden mehrere einzelne Segmente verbunden um möglichst wenig Gewicht mit möglichst viel Stabilität zu verbinden.

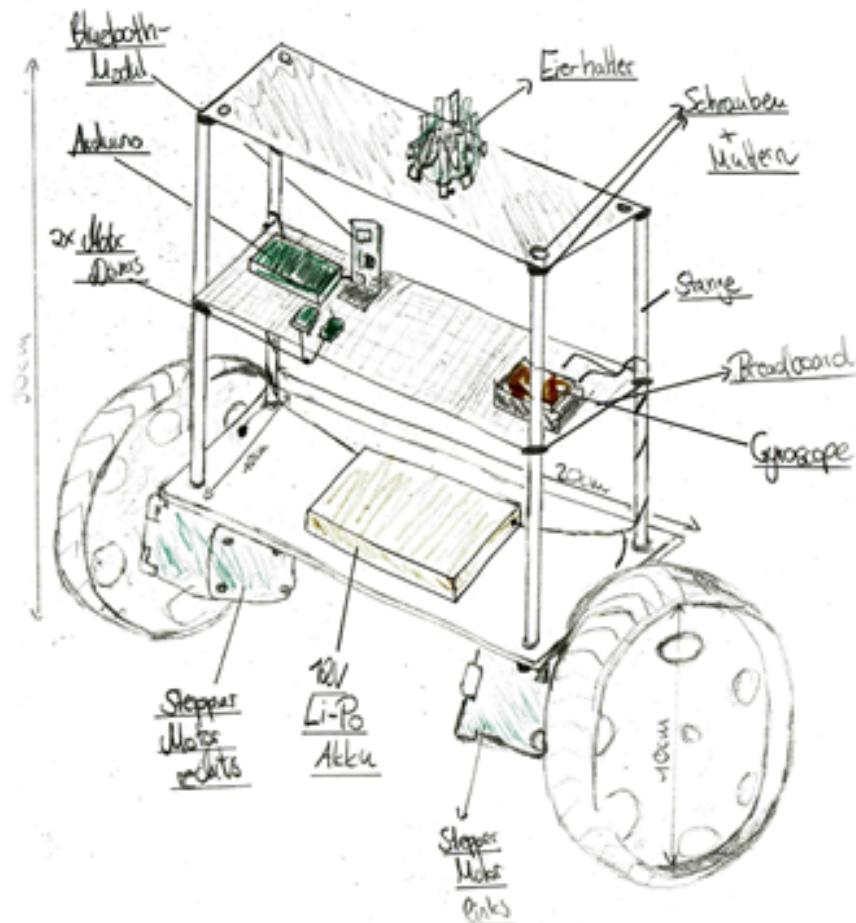


Abbildung 3.1: Skizze Seggway

Siehe Tabelle 3.1 für eine genaue Aufzählung der benötigten Komponenten. Alle restlichen Bauteile wie die Stangen, Ebenen und Reifen könnten problemlos via 3D-Druck mit PLA hergestellt werden.

Bezeichnung	Anzahl
Bluetooth-Modul	1
Arduino	1
Motor Driver	2
Stepper Motor	2
12V Li-Po Akku	1
Gyroscope	1
Breadboard	1
Schraube + Mutter	8

Tabelle 3.1: Stückliste Seggway

### 3.1.2 Weggchair

Beim Weggchair ist die Namensgebung leider ein wenig misslungen. Der Rest vom Konzept wäre mit den vom Lehrstuhl bereitgestellten Materialien recht leicht zu realisieren gewesen. Deswegen wäre es der Plan B gewesen, falls das präferierte Konzept nicht realisiert werden kann.

Das Konzept orientiert sich, wie der Name andeuten soll, recht nahe an einem Rollstuhl. Die gesamte Elektronik befindet sich in einer Zwischenebene unter der "Sitzfläche". Gelenkt wird der Weggchair indem sich beide Motoren verschieden schnell drehen.

Wie in der Skizze (Abb. 3.2) zu sehen ist, ist das "Stützrad" eher eine "Stützkugel", die in alle Richtungen über den Boden gleiten kann. Statt einer Kugel wären auch eine oder zwei drehbar gelagerte Rollen denkbar gewesen, ähnlich wie bei einem Einkaufswagen oder natürlich dem originalen Rollstuhl.

Da die verwendeten DC-Motoren unter Volllast über 7000 Umdrehungen schaffen, dafür aber weniger Drehmoment haben, wäre – wie in der finalen Idee – ein Planetengetriebe innerhalb der großen Räder zum Einsatz gekommen.

Das Ei wäre beim Weggchair an der Stelle befestigt, wo normalerweise der Rollstuhlfahrer sitzt. Dafür wäre eine Federung und Halterung an dieser Stelle gewesen. Vermutlich wäre ein mit Watte oder einem ähnlichen elastischen, polsternden Material ausgestatteter Eierbecher zum Einsatz gekommen.

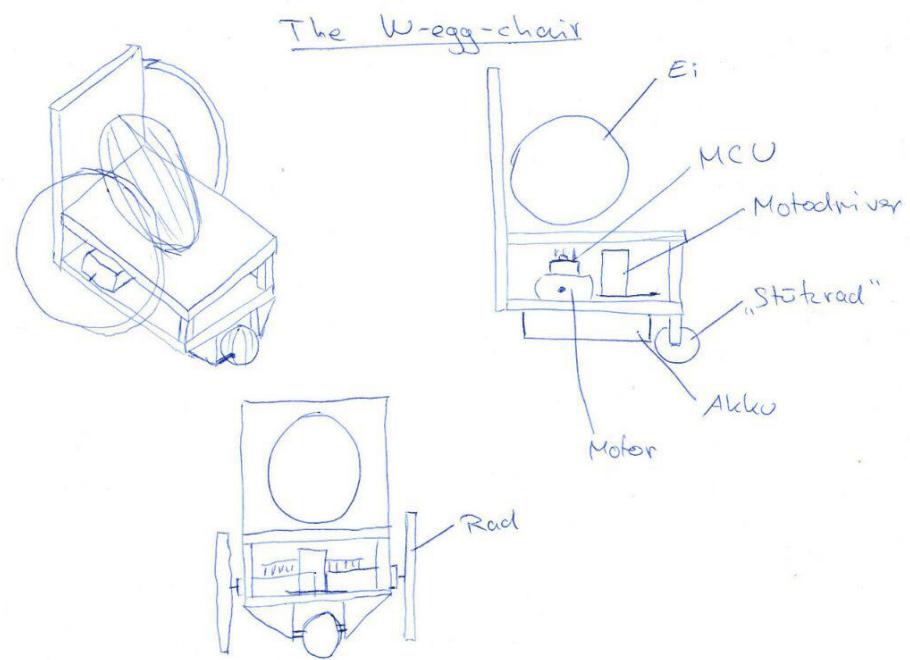


Abbildung 3.2: Skizze Weggchair

### 3.1.3 TEGGLA

Da sich die nächsten drei Kapitel nur mit dem finalen Konzept auseinandersetzen, wird an dieser Stelle nur ganz kurz auf das Konzept des TEGGLA (Abb. 3.3) eingegangen, damit die nachfolgenden Kapitel nachvollziehbar sind.

Das besondere Feature beim TEGGLA sind die omnidirektionale Räder, auch als Mecanum bezeichnet, die neben dem normalen Vorwärtsfahren auch Seitwärtsbewegungen und Rotation zulassen. Diese drei Freiheitsgrade lassen sich auch beliebig kombinieren. Dadurch ist in der Ebene jede denkbare Richtung befahrbar. Dazu sind allerdings vier Motoren notwendig.

Daher wächst die Elektronikteileliste wie folgt: Für die zwei Extramotoren wird eine zusätzliche H-Brücke benötigt. Leider hat das bereitgestellte ESP-8266 nicht genug Pins für die zusätzliche Elektronik, weswegen zum ESP-32 upgegraded werden musste.

Es folgen noch viel mehr Details, aber um die folgenden Kapitel zu verstehen, muss noch gesagt werden, dass der TEGGLA in der Entwicklungsphase noch Omni-Move hieß. Beide Namen sind im Folgenden also synonym.

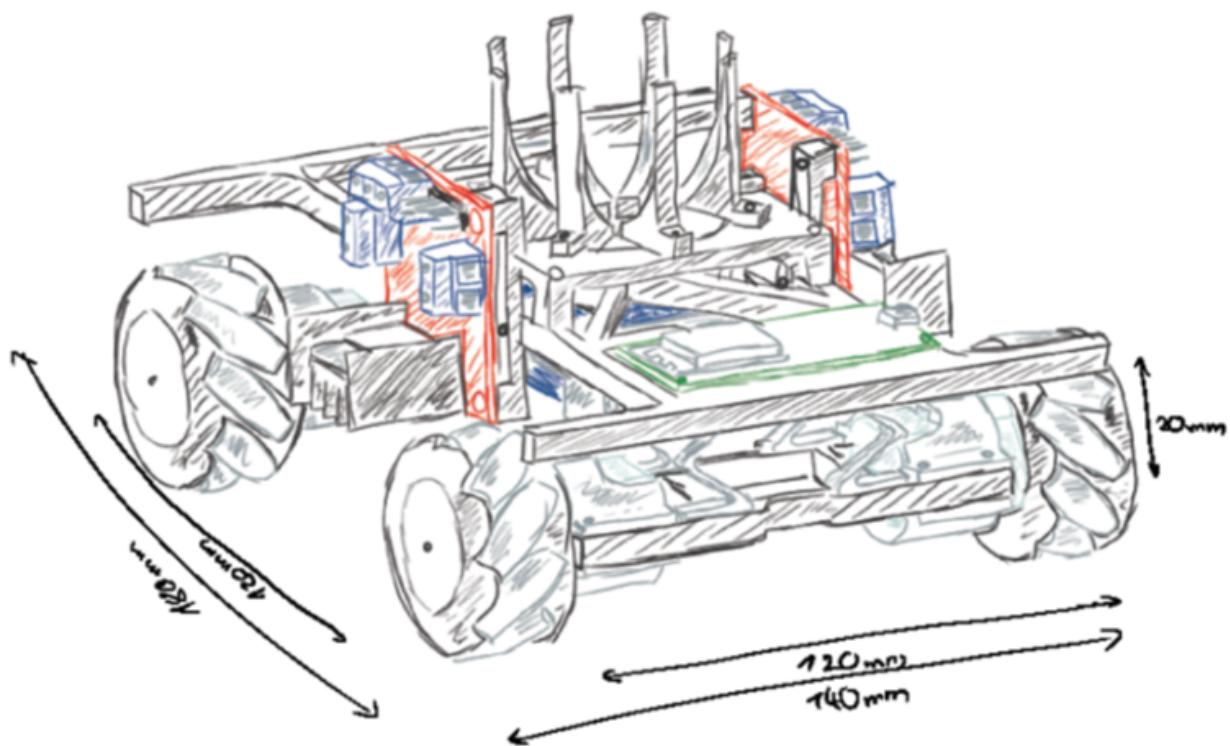


Abbildung 3.3: Skizze TEGGLA

## 3.2 Morphologischer Kasten

Mithilfe des Morphologischen Kastens (Abb. 3.4) lassen sich die benötigten Komponenten auf eine einfach ersichtliche Weise vergleichen.

	OmniMove		(S)eg(g)way	
	Variante A	Variante B	Variante C	Variante D
Antrieb	Servo	DC-Motor		Step-Motor
Lenkung	Lenkachse	Einzelne gesteuerte Motoren		
Räder	Mecanum	Rad		Spinnenbeine
Getriebe	Keins	Planetengetriebe		Gearbox
Controller	ESP32	ESP8266		Arduino
Akku	LiPo 2S	Battery		LiPo 3S
Ladetechnik	Keiner	BMS + 9V Netzteil		BMS + 5V USB
Steuerung	Laptop	Spiele-Controller		Handy
Federung	Keine	3D-Druck		Feder-Dämpfer
Lage-Sensorik	Keine	Gyroskop		Pendel
Material	PLA	PLA+TPU		Holz
Verbindung	Schrauben	Pins		ABS

Abbildung 3.4: Morphologischer Kasten

Um sich aufrecht zu halten benötigt der Seggway Stepper-Motoren statt der DC-Motoren, da diese genauer kontrolliert werden können. Dadurch könnte das Getriebe gespart werden. Allerdings braucht der Seggway auf jeden Fall ein Gyroskop (und eine gute Regelung) um aufrecht stehen zu bleiben und zu fahren.

Bei der Steuerung setzen alle Konzepte auf einen Spiele-Controller, da es keine leichtere und intuitivere Steuerung gibt.

Der Weggchair würde, wie oben angemerkt, mit den vom Lehrstuhl bereitgestellten Materialien auskommen. Für die Übersetzung wäre ein Planetengetriebe im Rad zuständig.

Der TEGGLA braucht ein ESP-32, da das ESP-8266 zu wenig Anschlüsse hat. Das Highlight, die Räder, sind bei diesem Konzept die Mecanum-Wheels. Als Energiespeicher kommt ein zweizelliger LiPo in Kombination mit einem BMS (Battery Management System) zum Einsatz. Hier ist der Einsatz des Spiele Controllers besonders wichtig, da damit alle drei Achsen (X-Achse, Y-Achse und Rotation) analog gesteuert werden können. Für die Sicherheit des Eies sorgt eine Federung aus TPU. Zum TEGGLA werden alle Teile und Entscheidungen in diesem Bericht noch näher beleuchtet.

### 3.3 Online Bestellungen

Um das Fahrzeug nach dem Praktikum behalten zu können, war eine Voraussetzung, dass nur Teile verbaut werden, die nicht Eigentum des Lehrstuhls sind. Aus diesem Grund wurden die nötigen Bauteile bei unterschiedlichen Onlineshops herausgesucht und bestellt. Hierbei fiel die Entscheidung auf Pollin, einem deutschen Elektronik-Händler und AliExpress, einem chinesischen Großhändler. In der ursprünglichen Planung wurden die Kosten pro Fahrzeug auf circa 20 € – 25 € überschlagen. In der finalen Bestellung beliefen sich die Kosten auf insgesamt etwa 32 €.

Da die Bauteile in Sammelbestellungen für insgesamt 4 Fahrzeuge getätigt wurden, verteilten sich die Versandkosten auf die Fahrzeuge, wodurch sich der Preis nach unten hin anpasste. Siehe Tabelle 3.2 für eine genaue Aufteilung der Kosten.

Artikel	Stk	€/Stk	€	Laden
Netzteil 9V 1A	1	0,95	0,95	Pollin
DC Motor	4	0,95	3,80	Pollin
2S LiPo	1	9,95	9,95	Pollin
XT60 5er Satz	0,5	1,8	0,90	AliExpress
ESP32	1	3,77	3,77	AliExpress
2s BMS	1	0,89	0,89	AliExpress
Kabelset 20cm	0,5	3,30	1,65	AliExpress
Kabelset F – F 10cm	0,5	0,68	0,34	AliExpress
Gyroskop	1	0,93	0,93	AliExpress
H-Brücken	2	1,17	2,34	Bestand
Filament /kg/	0,05	20,00	1,00	Bestand
Schrauben + Muttern [Set]	1	1,00	1,00	Bestand
Motorkabel	1	0,50	0,50	Bestand
Versandkosten AliExpress	0,25	9,00	2,25	
Versandkosten Pollin	0,25	5,00	1,25	
Total				31,52 €

Tabelle 3.2: Kostenübersicht

## 3.4 Verwendete Technologien

Damit alle Teammitglieder an dem Projekt arbeiten können, musste sich vor dem Beginn auf die Software geeinigt werden, die verwendet wird. Diese Entscheidungsfindung wird in diesem Kapitel genauer betrachtet.

### 3.4.1 Git

Zu Beginn des Projekts war bereits klar, dass eine Datenversionskontrolle für alle Projektdateien benötigt wird. Durch verschiedene Vorerfahrungen fiel die Entscheidung sehr leicht. Git löst genau diese Aufgaben perfekt. Das Projekt wurde in einem privaten Repository auf Github gehostet. Die Wahl der graphischen Oberfläche für Git war jedem Gruppenmitglied selbst überlassen. Vereinzelt wurde hier auf “Git Extensions” oder meist eher “GitKraken” (Abb. 3.5) gesetzt, aufgrund der übersichtlichen graphischen Oberfläche.

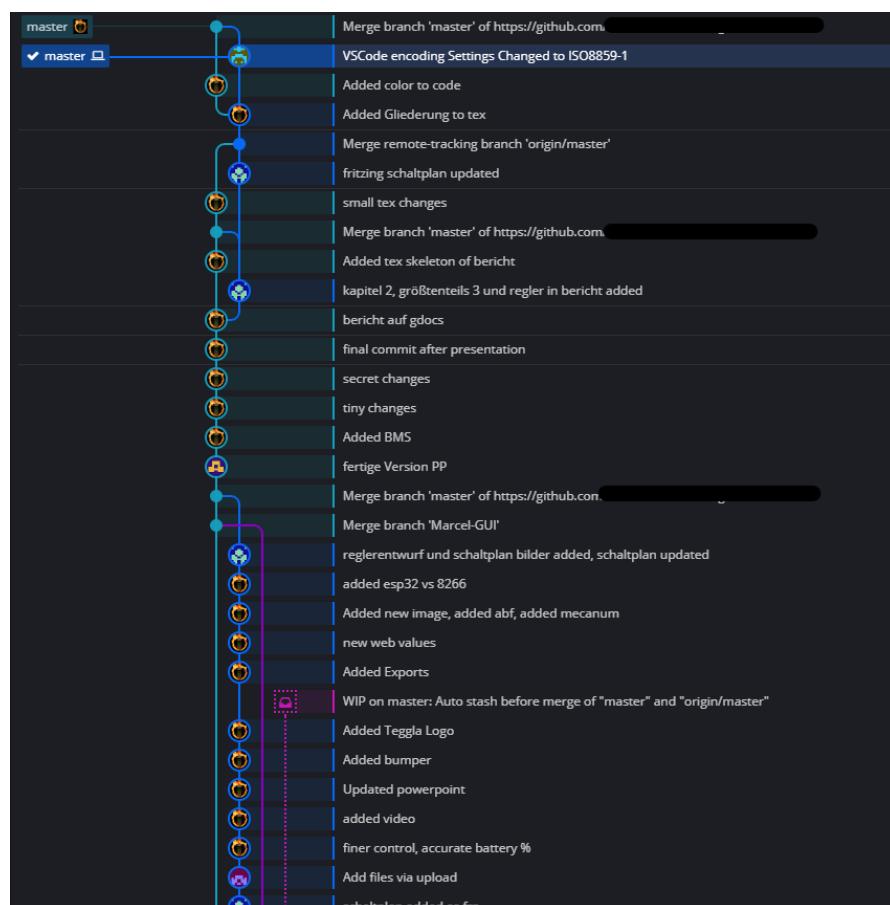


Abbildung 3.5: Graphische Oberfläche für Git (GitKraken)

### 3.4.2 Creo

Bei der CAD-Software standen mehrere unterschiedliche Programme unterschiedlicher Hersteller zur Auswahl. Diese waren “Catia” von Dassault Systemes, “Sketchup” von Trimble Inc., “Creo Parametrics” von PTC Inc., “Blender” von Blender Foundation, “OpenSCAD” (Opensource Programm) und “FreeCAD” (ebenfalls Opensource). Nach längerem Abwägen fiel die Entscheidung auf das Programm Creo Parametrics (Abb. 3.5), da dieses vergleichbar einfach zu bedienen ist, aber trotzdem einen sehr großen Funktionsumfang bieten kann. Des Weiteren bietet Creo eine gute 3D-Maus Unterstützung, was das Modellieren deutlich vereinfacht. Trotz des großen Funktionsumfangs reichten die Möglichkeiten, die Creo bietet, nicht immer aus, weshalb in Einzelfällen auf andere CAD-Programme zurückgegriffen wurde. Diese werden allerdings an den betroffenen Stellen extra erwähnt.

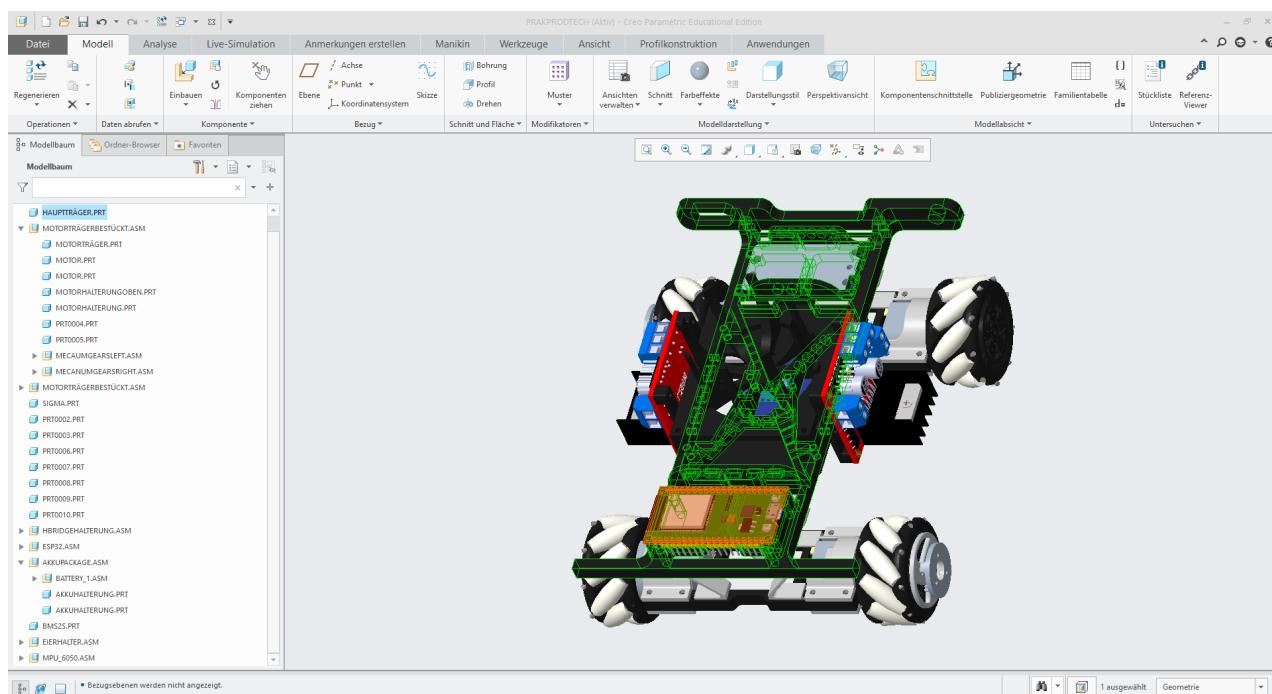


Abbildung 3.6: CAD Programm “Creo Parametrics”

### 3.4.3 PlatformIO

Die Wahl der Entwicklungsumgebung fiel auf Visual Studio Code mit PlatformIO, anstelle der in der Vorlesung vorgestellten Arduino IDE.

Mit der umfangreichen Erweiterbarkeit von VSCode ist hier die Programmierung einfacher und fehlerfreier durchzuführen, da in Arduino IDE nur bedingtes Syntax-checking vorhanden ist. Durch PlatformIO stellt sich das Einbinden von externen Bibliotheken ebenfalls als Leichtigkeit heraus. Des Weiteren ist hier die Unterstützung von einer Vielzahl von MicroControllern bereits integriert.

# 4 Entwicklung der Schlüsselemente

## 4.1 Übersicht des gesamten Modells

Wie in Abbildung 4.1 zu erkennen, besteht das Fahrzeug aus vielen Einzelteilen. Auf die wichtigsten Komponenten, die in der Stückliste (Tabelle 4.1) genannt werden, wird in den folgenden Kapiteln genauer eingegangen.

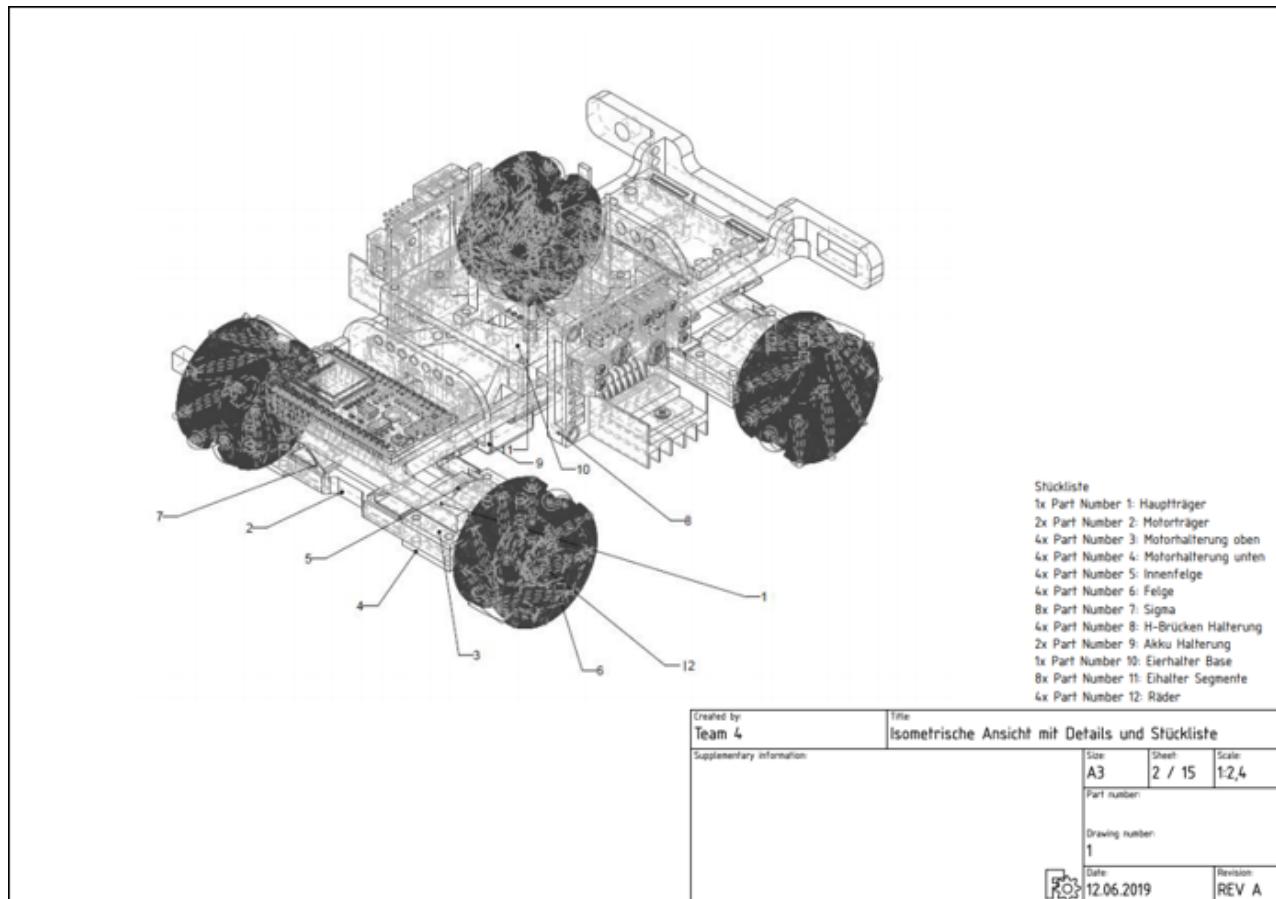


Abbildung 4.1: Übersicht der TEGGLA-Komponenten

Bezeichnung	Anzahl
Hauptträger	1
Motorträger	2
Motorhalterung oben	4
Motorhalterung unten	4
Innenfelge und Felge	4
Sigma	8
H-Brücken Halterung	4
Akku Halterung	2
Eihalterung Base	1
Eihalter Segmente	8
Mecanum-Räder	4

Tabelle 4.1: Stückliste der TEGGLA-Komponenten

## 4.2 Schaltplan

Der Schaltplan (Abb 4.2) zeigt eine grobe Skizze der Verkabelung des TEGGLA Fahrzeugs. Der 2S Lipo Akku wird hierbei durch die beiden Einzellenakkus dargestellt. Der Hauptschalter, ein Dreipositionenschalter, dient zum Ein- und Ausschalten, sowie um das Fahrzeug in den Lademodus zu versetzen. Die linke der beiden H-Brücken dient neben der Motoransteuerung auch zur Spannungsregulierung für das ESP.

Das ESP32 ist die zentrale Steuereinheit und steuert mit je sechs digitalen Pins die H-Brücken an und kann so die Motordrehrichtung sowie die Geschwindigkeit der Motoren steuern. Das ESP32 kommuniziert auch via dem I2C Bus mit dem Gyroskop und versorgt dieses mit der 3.3V Spannungsversorgung.

Die Widerstände am Ausgang des BMS, auf welches im folgenden Abschnitt eingegangen wird, bilden einen 3:1 Spannungsteiler. Dieser wird benötigt, da das ESP32 einen maximalen Spannungseingang von 3.3V verträgt, am BMS jedoch bis zu 9V anliegen können. Durch diesen Spannungsteiler ist es über den Analog-Digital-Wandler in dem ESP32 möglich, den Akkustand in 4095 Schritten auszulesen. Hierdurch wird dem User ein Feedback gegeben, wie lange die Akkuladung noch ausreicht.

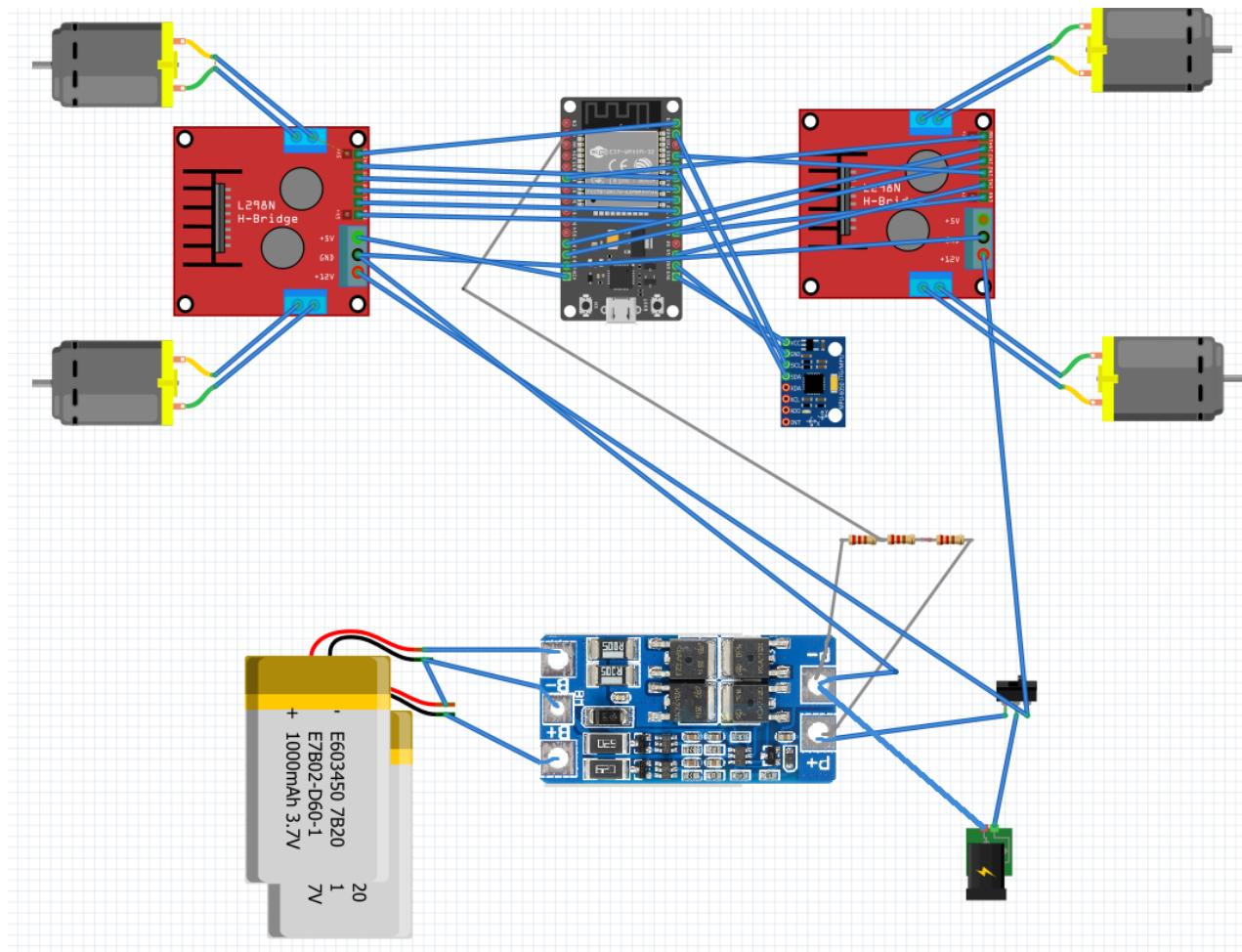


Abbildung 4.2: Schaltplan (gezeichnet mit Fritzing)

### 4.3 BMS und Laden

Um einem Akkuschaden vorzubeugen, sollte ein User doch einmal übersehen, dass der Akku des Fahrzeugs leer ist, wurde ein BMS (Battery Management System) verbaut. Dieses verhindert durch rechtzeitiges Abschalten ein Über- und Tiefenentladen des Akkus. Um das Fahrzeug wieder aufladen zu können, wurde ein externes Netzteil gekauft, welches über einen XT60 Stecker an das Fahrzeug angeschlossen werden kann. Sobald der Hauptschalter in die Ladeposition gebracht wird, beginnt das Fahrzeug nun den internen Akku zu laden, bis das BMS bei geladenem Akku den Ladevorgang vollautomatisch beendet.

## 4.4 Mecanum

Als besonderes Merkmal des TEGGLA fallen sofort die besonderen Räder auf, die sogenannten Mecanum-Räder. Hierbei handelt es sich um von Bengt Ilon 1972 erfundene Räder, die es dem Fahrzeug erlauben sich in drei Freiheitsgraden zu bewegen.

Dies wird ermöglicht durch die um  $45^\circ$  gedrehten Rollen auf den Rädern, sodass diese wie ein X aussehen. Durch unterschiedliche Drehrichtung und Drehgeschwindigkeit der Motoren kann das Fahrzeug in jegliche Richtungen innerhalb der Ebene bewegt werden. (vgl. Abb 4.3)

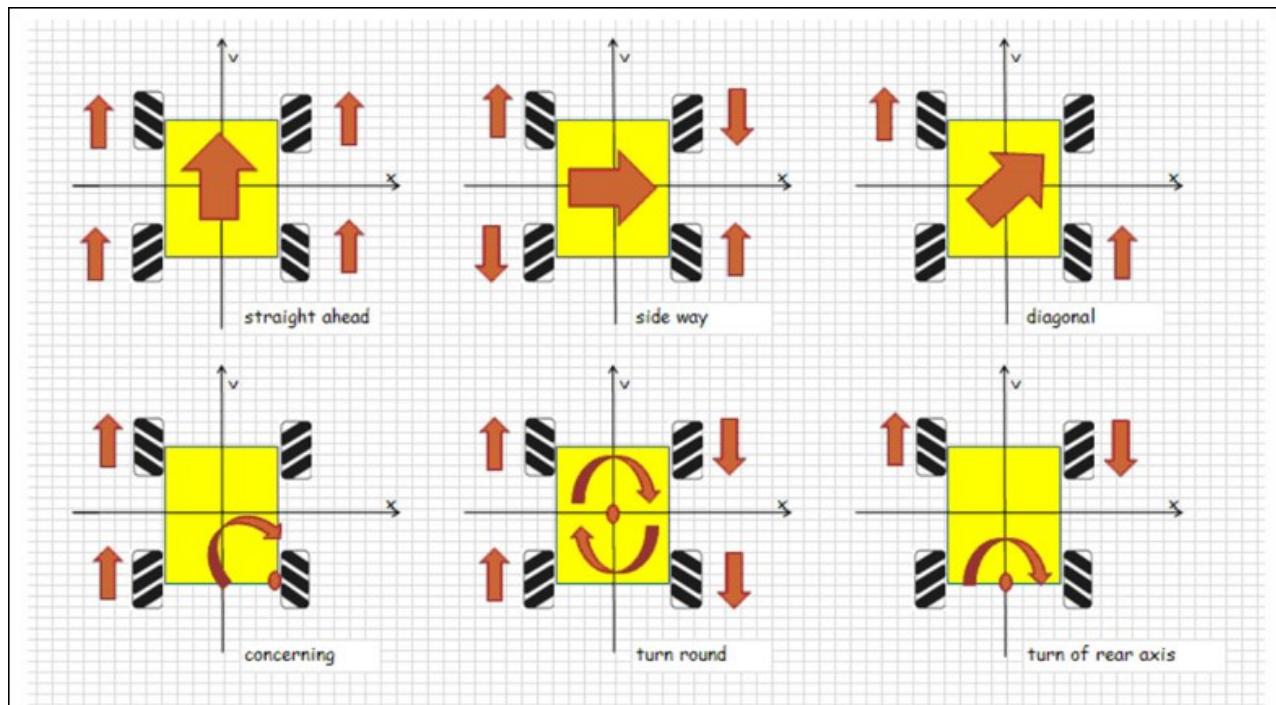


Abbildung 4.3: Freiheitsgrade von Mecanum [5]

Die jeweiligen Motorgeschwindigkeiten lassen sich hierbei durch Formeln 4.1–4.4 berechnen.

$V_x$  = Motorgeschwindigkeit des x-ten Rads im Uhrzeigersinn (beginnend vorderes linkes Rad)

$V_d$  = gewünschte Roboter Geschwindigkeit  $[-1, 1]$

$\theta_d$  = gewünschter Winkel  $[0, 2\pi]$

$V_\theta$  = gewünschte Rotationsgeschwindigkeit  $[-1, 1]$

$$V_1 = V_d \sin \left( \theta_d + \frac{\pi}{4} \right) + V_\theta \quad (4.1)$$

$$V_2 = V_d \cos \left( \theta_d + \frac{\pi}{4} \right) - V_\theta \quad (4.2)$$

$$V_3 = V_d \cos \left( \theta_d + \frac{\pi}{4} \right) + V_\theta \quad (4.3)$$

$$V_4 = V_d \sin \left( \theta_d + \frac{\pi}{4} \right) - V_\theta \quad (4.4)$$

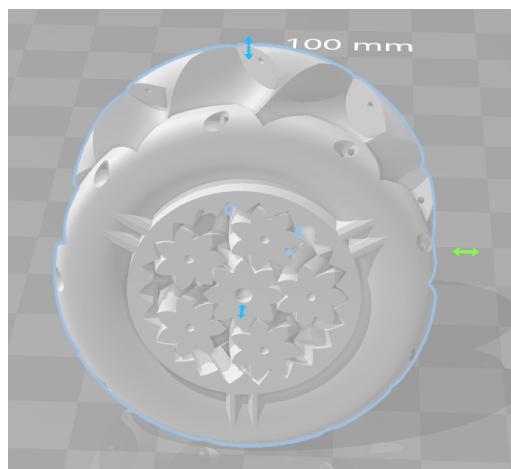
Abbildung 4.4: Formeln für die individuellen Motoren [5]

Bei den in diesem Praktikum verwendeten Rädern handelt es sich um eine Modifizierung der STL Dateien von Jonah Innoart von dem Internetportal Thingiverse [4]. In das Zentrum des Rades wurde mithilfe einer Boolean Operation von Blender3D eine Führung für die Getriebe geschnitten, sodass diese in das Rad eingeschoben werden können.

## 4.5 Planetengetriebe

Bereits in der 2. Woche des Projekts sind die ersten Entwürfe für die Planetengetriebe entstanden. Dieser setzte sich aus zwei 3D-Modellen zusammen, dem Getriebe (Abb 4.5b) und dem Rad, die beide von der Onlineplattform “Thingiverse” bezogen wurden. Diese wurden in Blender zu einem Bauteil zusammenfügt (Abb 4.5a).

Nach einigen erfolgreichen Testdrucken stellte sich allerdings heraus, dass die Getriebe zu viel Reibung hatten, sodass sie nicht anlaufen konnten.



(a) Version 1 des Planetengetriebes



(b) Original des verwendeten Planetengetriebes [1]

Um der Reibung entgegen wirken zu können, wurden im nächsten Schritt mit dem CAD Programm “OpenSCAD” (Abb 4.6) eigene Planetengetriebe erstellt. Hierbei lassen sich vielerlei Parameter einstellen, beispielsweise Größe, Toleranzen, Zahenzahl, Planetenzahl, etc. Dies ermöglicht frei mit der Übersetzung sowie den druckbaren Toleranzen zu testen um das bestmögliche Getriebe zu bekommen.

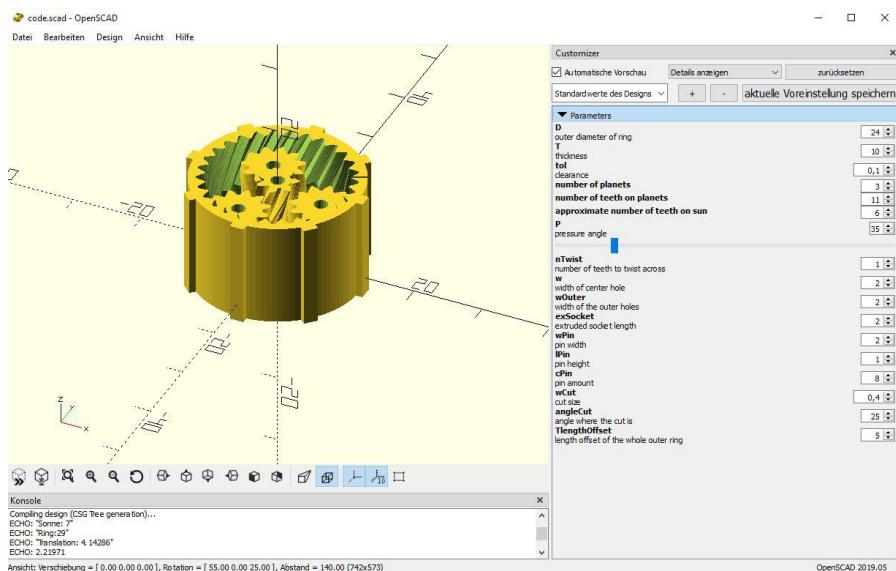
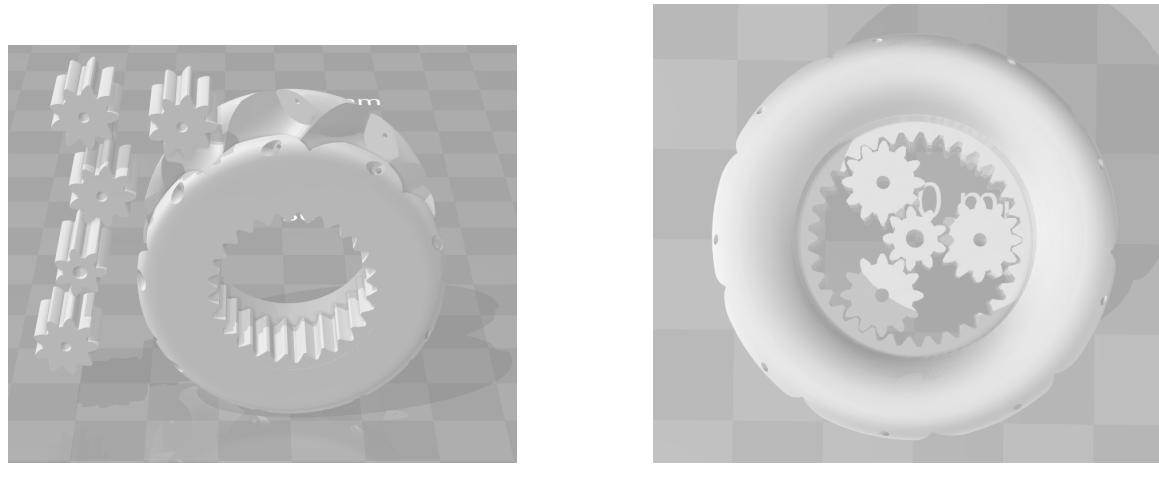


Abbildung 4.6: OpenSCAD zum Erstellen von Planetengetrieben

Durch die Funktionsweise von Additiver Fertigung lassen sich V-förmig verzahnte Getriebe als eine Einheit drucken. Dies wäre mit konventionellen Fertigungsprozessen nicht möglich. Da jedoch aufgrund von mangelnder Genauigkeit des 3D Druckers die Toleranzen so hoch gewählt werden müssen, sodass sich die Zahne nicht verschmelzen, führt dies zu viel Spiel in dem Getriebe.

Um jedoch das Getriebe weiterhin platzsparend in das Rad zu senken, wurde hier ebenfalls mit geradverzahnten Getrieben (Abb. 4.7) getestet. Hierbei entsteht jedoch das Problem der Fixierung der Planeten. Es entstand somit der Plan eine Gegenfalte zu Benutzten um alle Teile zu halten.



(a) Version 2 des Planetengetriebes

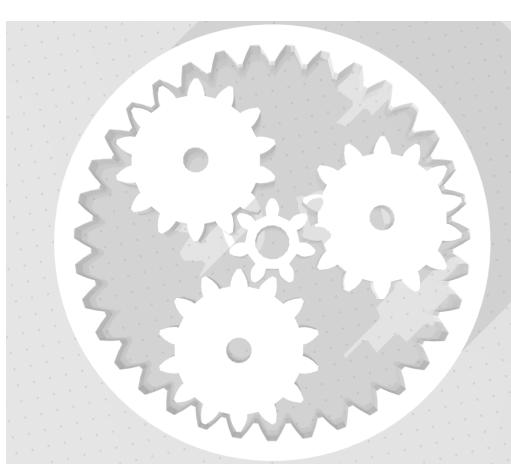
(b) Version 3 des Planetengetriebes

Abbildung 4.7: Im Rad integrierte Getriebe

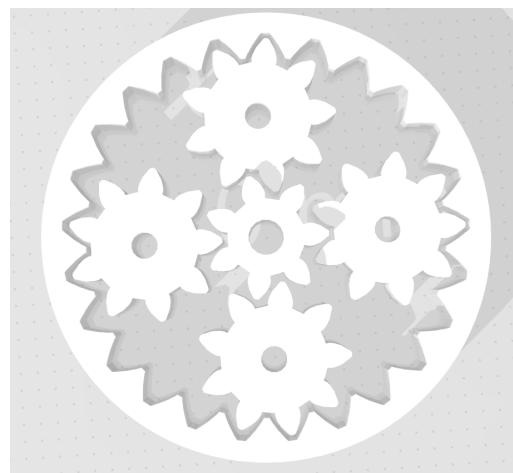
Weiterhin gab es noch das Problematik der zu niedrigen Übersetzungen. Hier hat es einige Iterationen gedauert bis eine Übersetzung von über 1:4 erreicht werden konnte.

Probleme die hier aufgetreten sind, sind beispielsweise zu kleine Zähne (Abb. 4.8a), sodass diese nicht mehr ineinander greifen konnten.

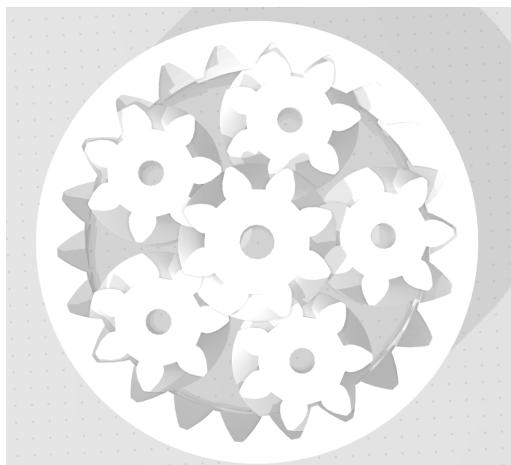
Desweiteren hängt die mögliche Übersetzung auch von der Anzahl der Planeten ab. Sind zu viele Planeten im Ring (Abb. 4.8c) überschneiden sie sich, und man muss somit auf weniger Planeten (Abb. 4.8b) zurückgreifen, was die Stabilität beeinträchtigt. Hierbei lässt sich auch der Kraftwinkel der Zähne einstellen. Ist der Winkel zu niedrig (Abb. 4.8d) rutschen die Zähne einfacher durch, verhaken sich jedoch nicht so stark.



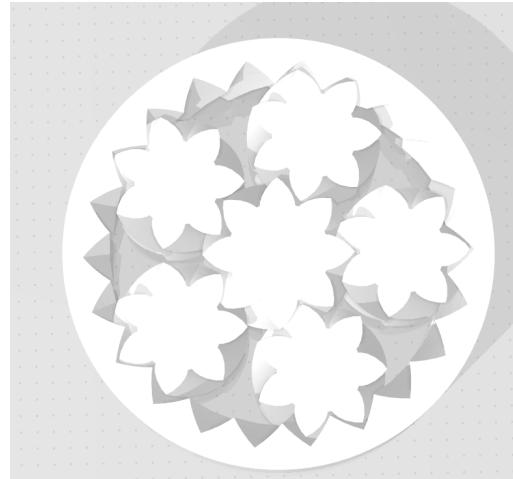
(a) Version 4 des Planetengetriebes



(b) Version 5 des Planetengetriebes



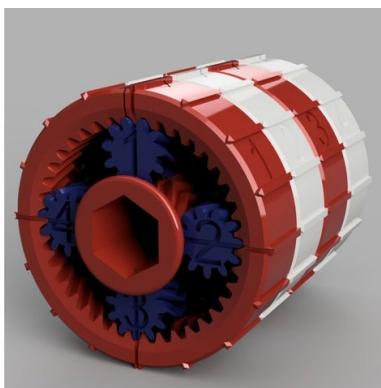
(c) Version 6 des Planetengetriebes



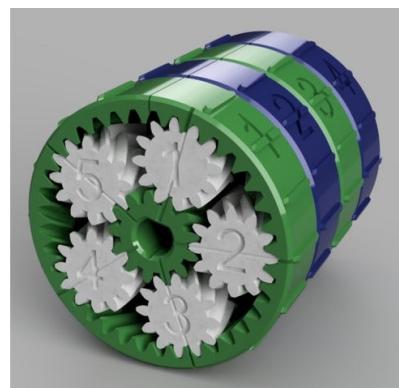
(d) Version 7 des Planetengetriebes

Abbildung 4.8: Iterationen der Getriebe

Da Versuche mit höheren Übersetzungen als 1:4 nie Erfolg hatten, wurden ebenfalls mehrstufige Getriebe (Abb. ) getestet, sogenannten “Composite Gear Sets”. Hierbei sind Übersetzungen von über 1:66 möglich, was für diese Anwendung perfekt wäre. Selbst bei mehrfachen Testdrucken der Zweistufigen Getriebe gelang es nicht, ein lauffähiges Getriebe mit ausreichendem Wirkungsgrad zu erreichen. Es gelang bei einigen Getrieben sie per Hand mit einiger Gewalt zu drehen, jedoch hatten die Motoren nicht genug Moment um überhaupt aus dem Stillstand loszudrehen.



(a) Version 8 des Planetengetriebes [2]

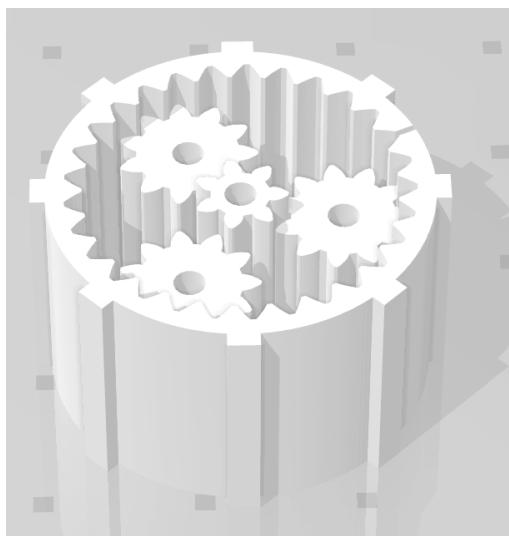


(b) Version 9 des Planetengetriebes [3]

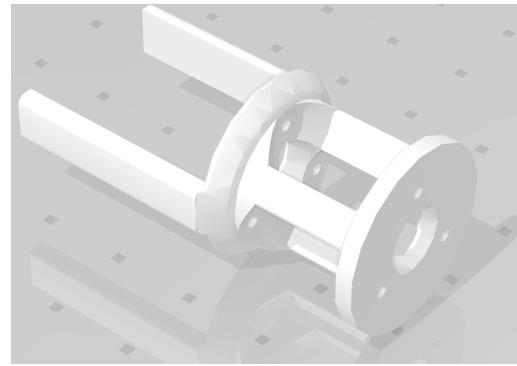
Abbildung 4.9: Mehrstufige Getriebe

Aus diesen Gründen wurde sich letztendlich auf einstufige Getriebe mit einer Übersetzung von  $1 : 4.14$  geeinigt. Obwohl dies zwar nicht ausreichend ist, um die Motoren von einer Lastdrehzahl von bis zu 6000 rpm auf eine lenkbare Geschwindigkeit zu übersetzen, wurde hier ebenfalls mit bedacht, dass weitere Steuerung durch Software mithilfe der H-Brücken möglich ist.

Dies führt zu dem finalen Design (Abb. 4.10a). Aufgrund der geringeren Reibung wurde sich hierbei für eine Geradverzahnung entschieden. Der 14mm lange Ring ist an einer Seite offen, sodass er aufgebogen werden kann um ihn um den Käfig (Abb. 4.10b) zu schließen. Bei dem Käfig handelt es sich um eine Verbindung der Innenfelge, die auf den Motor aufgesteckt wird mit der Außenfelge. Der Käfig erfüllt ebenfalls den Zweck die Planeten an ihrem Platz zu halten, sowie zusätzliche Stabilität der Achsen zu gewährleisten. Für die Achsen der Planeten dienen hier 1.5mm, die auf der Innenseite eingeklebt sind.



(a) Finale Version des Planetengetriebes



(b) Finale Version des Käfigs

Abbildung 4.10: Finale Getriebebaugruppe

## 4.6 ESP-32 vs. ESP-8266

Bei dem vom Lehrstuhl gestellten ESP-8266 handelt es sich um einen WiFi-fähigen Microcontroller der chinesischen Firma “espressif”. Dieser besitzt 17 GPIO Pins, wovon jedoch lediglich 11 Pins nutzbar sind, da 6 an den externen SPI Flash angeschlossen sind. Da dies wie in Tabelle 4.2 aufgezeigt nicht für unsere Zwecke reicht, musste auf den leistungsstärkeren ESP-32 ausgewichen werden.

Benötigte Pins	
4x PWM	Motor Enable
8x Output	Motor Richtung
2x I <sup>2</sup> C	Gyroskop
1x ADC	Batteriespannung
15 Pins	

Tabelle 4.2: Aufzählung benötigter Pins

Obwohl die Anzahl der Pins das ausschlaggebende Argument für einen Austausch des MicroControllers war, bringt dieser natürlich weitere Vorteile mit sich.

Beispielsweise profitiert die später genauer erklärte Website stark davon, auf einen zweiten Core ausgelagert werden zu können.

Siehe Tabelle 4.3 für einen Vergleich der beiden MicroController anhand der für dieses Projekt relevanten Faktoren.

	ESP-8266	ESP-32
Cores	single core	dual core
Max Frequenz	160 MHz	240 MHz
<b>GPIO</b>	<b>17 (11 nutzbar)</b>	<b>36 (30 nutzbar)</b>
SRAM	160 KB	520 KB
ADC Auflösung	10 bit	12 bit
Stromverbrauch	80 mA	260 mA
Preis (aus China)	2€	4€

Tabelle 4.3: Vorteile des ESP-32

## 4.7 User-Interface

Eine passende, nutzerfreundliche und optisch ansprechende UI hatte für das Projekt TEGGLA von Anfang an eine hohe Priorität. Es wurden zwei verschiedene UIs für das Projekt entwickelt, welche nachfolgend detailliert beschrieben werden.

### 4.7.1 Java (obsolete)

Das erste Konzept für eine passende UI wurde mit Java entwickelt. Java (und das Framework Java Swing) bieten grundsätzlich einige Möglichkeiten Benutzeroberflächen zu bauen und da Java eine der meistgenutzten Programmiersprachen ist, haben wir uns vorerst dafür entschieden. Die Java UI besteht aus zwei Fenstern, welche vorerst die Verbindung zum TEGGLA sicherstellen und anschließend die Steuerung und Kommunikation ermöglichen.

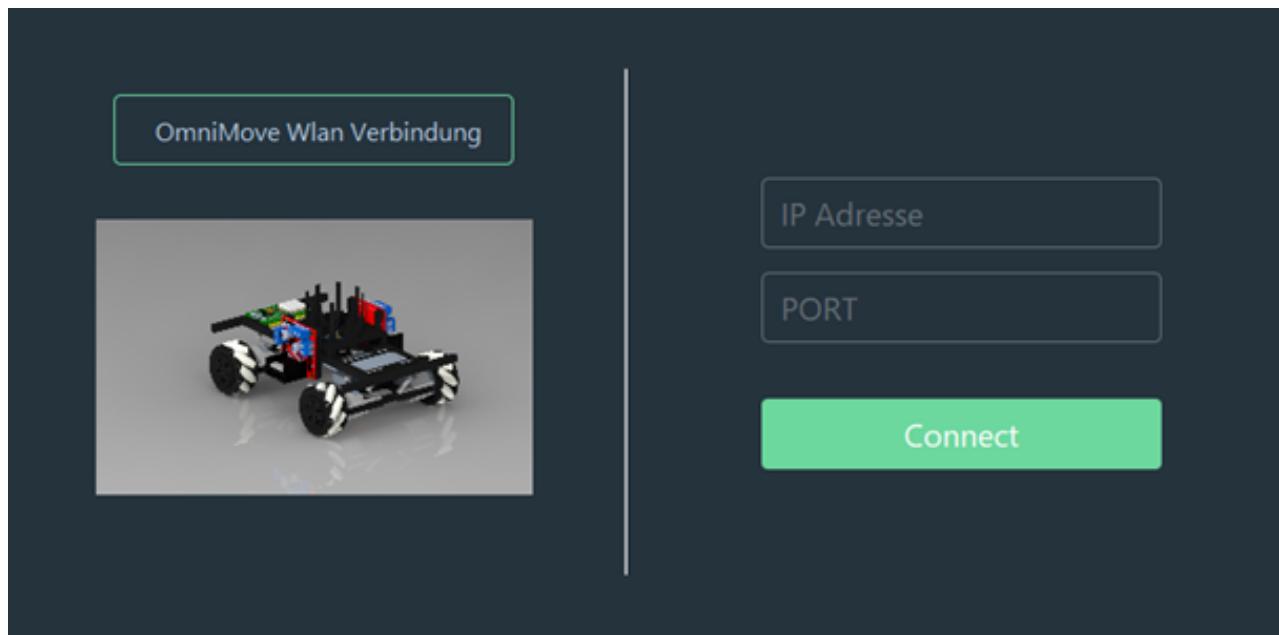


Abbildung 4.11: Java Fenster 1: Verbindungsaufbau

Im ersten Fenster (Abb. 4.11) müssen in die Textfelder „IP Adresse“ und „PORT“ die passenden Daten eingegeben werden, um anschließend mit dem „Connect“ Button eine Verbindung zum TEGGLA aufzubauen. Falls das nicht gelungen ist, bekommt der Nutzer eine Fehlermeldung als Pop-Up und muss die Daten erneut eingeben. Im Erfolgsfall erscheint eine kurze Erfolgsmeldung und es öffnet sich das zweite Fenster.

Im zweiten Fenster (Abb. 4.12) werden alle wichtigen Daten für die Steuerung und Kommunikation des TEGGLA angezeigt, nämlich Geschwindigkeit und Akkustand.

Die Entwicklung des zweiten Fensters war noch nicht ganz abgeschlossen, als Java durch die Verwendung der neuen UI mit JavaScript / HTML5 abgelöst wurde. Daher wäre dieses Fenster noch durch die Visualisierung der Motoransteuerung und -auslastung ergänzt worden.

Abschließend lässt sich festhalten, dass die Entwicklung einer UI mit Java (und Java Swing) durchaus Vorteile hat, aber das Erstellen einer optisch ansprechenden Benutzeroberfläche sehr zeitaufwändig ist. Für diese zwei Fenster (mit Logikschicht) wurden in etwa 1000 Zeilen Quellcode benötigt, was für größere Projekte mit mehr Komponenten nicht empfehlenswert ist. Der zugehörige Quellcode ist im Anhang des Berichts und kann bei Interesse daher noch genauer betrachtet werden.

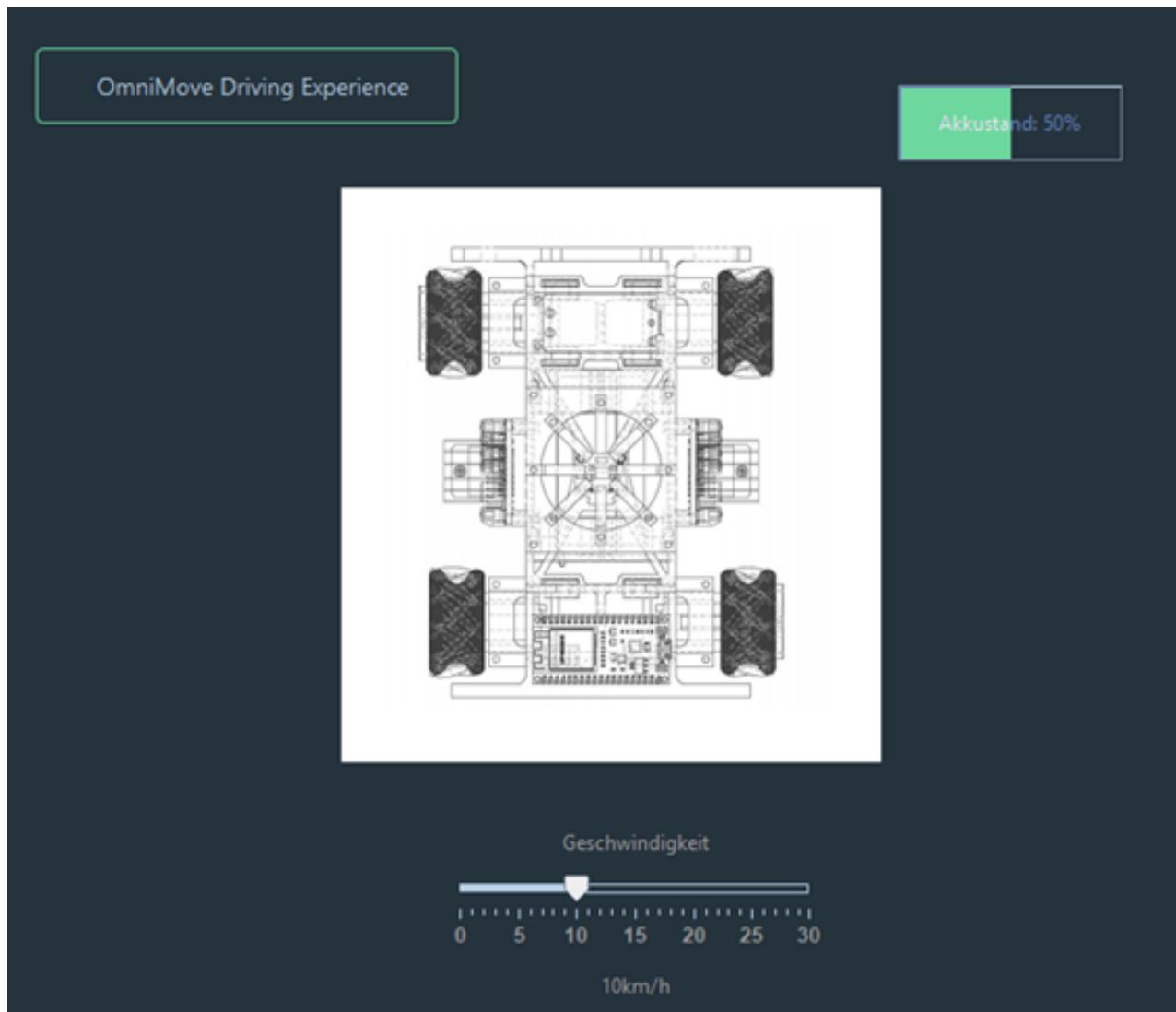


Abbildung 4.12: Java Fenster 2: Steuerung

#### 4.7.2 HTML5 und Controller-Anbindung

Das Java Programm wurde aus mehreren Gründen zugunsten einer auf HTML5 sowie JavaScript basierten Weboberfläche ersetzt:

- Native und einheitliche Unterstützung für Controller unterschiedlichster Marken in HTML5
- Unabhängig von Java Laufzeitumgebung, sowie Verfügbarkeit des Programms.  
(Hierbei muss nur ein Browser auf dem PC installiert sein.)
- Einfache Übertragung der Daten per WebSockets anstatt von “raw” Sockets, ohne ein eigenes “Frame” um die Daten bauen zu müssen

Die Erstellung der Website lässt sich sehr leicht durch die ESPAsyncWebServer Library für den ESP32 lösen.

Diese hostet direkt auf dem ESP32 einen WebServer der sowohl HTML5, JS, als auch CSS bereitstellen kann. Als Speicherort für diese Dateien wird das sogenannte Dateisystem SPIFFS verwendet, welches den Flashspeicher des ESP32 als Dateisystem benutzt, wie es beispielsweise aus Windows bekannt ist.

Eine Einschränkung ist die Limitierung auf eine gleichzeitige Verbindung zu dem Server. Dies wurde empirisch herausgefunden und somit konnte nicht sicher gesagt werden, ob es sich hier um eine Einschränkung aufgrund mangelnder Rechenleistung handelt oder ob die Library nicht mehr gleichzeitige Verbindungen unterstützt. Als Lösung dieses Problems, wurde nun die Anzahl der Verbindungen, die der WiFi Accesspoint akzeptiert, auf eins gesetzt.

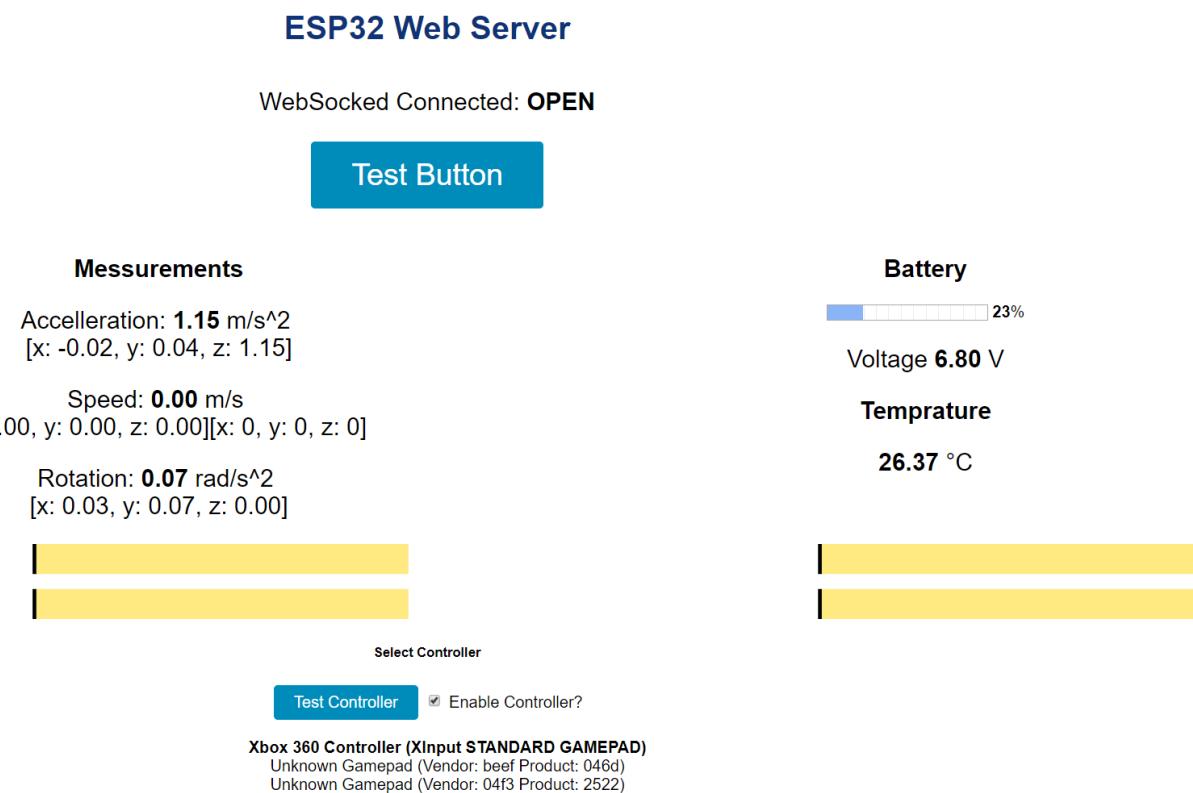


Abbildung 4.13: Bildschirmfoto Weboberfläche

In der Weboberfläche (Abb. 4.13) sind ebenfalls noch jegliche Messwerte hinterlegt, die das Fahrzeug sammelt.

Diese sind:

- Beschleunigung
- Geschwindigkeit
- Rotationsgeschwindigkeit
- Batteriespannung
- Temperatur
- Drehgeschwindigkeit der jeweiligen Motoren

Für die Wahl des Controllers ist eine auswählbare Liste aller angeschlossenen Controller am Ende der Seite vorhanden. Falls der Nutzer keinen Controller besitzt oder angeschlossen hat, kann durch das Abwählen der Checkbox auf die Steuerung per WASD, sowie die Pfeiltasten gewechselt werden.

### 4.7.3 Protokoll

Zum Übertragen wurde ein binäres Protokoll (Tabelle 4.4) entwickelt, um die Datenrate gering zu halten, sowie die Verarbeitung auf dem Microcontroller zu vereinfachen.

Da durch Websockets bereits ein Integrierter Frame geschickt wird, muss nicht bei jeder Nachricht die Länge sowie Anfangs- und Endbyte mitgeschickt werden, wie es sonst bei raw Sockets nötig gewesen wäre.

Hierbei wird jeder Wert als Int16 geschickt, um ein ausreichend großes Spektrum bei geringer Datenrate zu gewährleisten.

Am Anfang jeder Nachricht wird die ID ebenfalls als int16 geschickt, somit wären 65536 unterschiedliche Nachrichten erlaubt.

Um die Anzahl an Nachrichten zu verringern, wurden die Batteriespannung und Werte des Gyroskop zu einer kombinierte Nachricht zusammengefasst, um den Overhead gering zu halten.

Name	ID	Werte											
DRIVE	0	X-Achse L	Y-Achse L	X-Achse R	Y-Achse R								
GYRO	1	Speed-X	Speed-Y	Speed-Z	Accel-X	Accel-Y	Accel-Z	Rot-X	Rot-Y	Rot-Z			
BATTERY	2	Voltage											
MOTOR	3	Speed-M1	Speed-M2	Speed-M3	Speed-M4								
COMB	4	Speed-X	Speed-Y	Speed-Z	Accel-X	Accel-Y	Accel-Z	Rot-X	Rot-Y	Rot-Z	Battery	Temp	

Tabelle 4.4: Binäres Protokoll

## 4.8 Steuerung per (XBox) Controller

Wie bereits in der Dokumentation der Weboberfläche erwähnt, wird die Steuerung primär per Spiele Controller gelöst, beispielsweise einem XBox-Controller von Microsoft. Diese Entscheidung ist durch die erhöhte Mobilität motiviert.

Mit einer Steuerung per WASD bzw. per Pfeiltasten sind nur binäre Zustände messbar, gedrückt oder nicht gedrückt, volle Geschwindigkeit oder Stillstand. Im Vergleich dazu erlauben uns die JoySticks des Controllers durch unterschiedlich starke Auslenkung die Geschwindigkeit sehr variabel zu bestimmen.

Da hierbei ebenfalls der JoyStick in jegliche Richtungen bewegt werden kann, können ebenfalls die Mecanum-Räder so angesteuert werden, dass sie in genau diese Richtung fahren.

Jedoch sind hiermit nur zwei der drei Freiheitsgrade unseres Fahrzeugs abgedeckt. Um Rotationen um die eigene Achse mit variabler Geschwindigkeit zu steuern, wird die Eingabe des linken und des rechten JoySticks mit folgenden Formeln überlagert.

Sei hier  $controlSide$  Auslenkung des linken Sticks in X Richtung (nach rechts),  $controlFront$  Auslenkung des linken Sticks in Y Richtung (nach vorne) und  $controlTurn$  Auslenkung des rechten Sticks in X Richtung (nach rechts),

$$\theta = \text{atan2}(controlSide, controlFront) \quad (4.5)$$

$$V_d = \min(\sqrt{controlFront^2 + controlSide^2}, 1023) - \frac{controlTurn}{2} \quad (4.6)$$

$$V_\theta = \frac{controlTurn}{2} \quad (4.7)$$

## 4.9 PLA vs. TPU

Der Großteil der Teile für den TEGGLA wurde mit PLA Filament gedruckt. PLA steht für Polyactide, TPU steht für thermoplastisches Polyurethan. In diesem Kapitel wird auf die Unterschiede zwischen TPU und PLA eingegangen.

PLA hat im Vergleich ein höheres E-Modul, was zu einer höheren Zugfestigkeit und Steifigkeit führt. Auch in Sachen Verarbeitung ist PLA deutlich einfacher als TPU. Im Gegensatz zu TPU ist der Druckprozess mit PLA problemlos. Darüber hinaus ist TPU ein bisschen teurer als PLA.

Warum also TPU? TPU ist im Vergleich zu PLA viel flexibler und elastischer. Deswegen fiel die Wahl beim Rohstoff für die Sigmas und die Motorhalterungen auf TPU. Acht Sigmas verbinden die zwei Motorträger mit dem Hauptträger. Sie dienen hier quasi als Federung für die beiden

“Achsen”. Die Motorhalterungen sind das Bindeglied zwischen Motor und Motorträger. Es ist naheliegend, eine Federung und eine Befestigung für die Motoren aus einem elastischen Material zu drucken, auch wenn man mit einigen Nachteilen leben muss.

Um ein ansprechendes Design zu erhalten und um die Unterscheidung der Materialien zu erleichtern, wurde für den Roboter, sowie in den Skizzen, für PLA die Farbe schwarz verwendet, für TPU die Farbe weiß.

## 4.10 Eierhalter

Der Leichtbau-Anspruch lässt sich bei allen Komponenten des TEGGLA wiederfinden, sogar beim Eierhalter. Dieser wurde aus mehreren einzelnen Bauteilen erstellt, um mit möglichst wenig Gewicht die größte Stabilität für das Ei zu erreichen. Es wurde vorerst das Fundament des Eihalters gedruckt und anschließend mit 8 Eihalter-Segmenten verbunden. Somit kann das zu transportierende Ei schnell und sicher auf dem TEGGLA verwahrt werden.

Zusätzlich befindet sich der Eierhalter genau im Schwerpunkt des Fahrzeugs, um sicherzustellen, dass die aufgenommene Last gleichmäßig verteilt werden kann und die omnidirektionale Fahrweise des TEGGLA nicht beeinträchtigt werden kann.

## 4.11 Leichtbau

Bei der Konstruktion des TEGGLA wurde auf die Verwendung von massiven und schweren Bauteilen komplett verzichtet, wodurch das geringe Gesamtgewicht des Fahrzeugs (ca. 500 Gramm) erreicht worden ist. Dabei sind die schwersten Bauteile die Elektronikbauteile, auf die kein Einfluss genommen werden kann, wie die Motoren (je 36g), die H-Brücken (je 25g) sowie der Akku (62g).

Um ein omnidirektionales Fahren zu ermöglichen, benötigt der TEGGLA vier Motoren und zwei H-Brücken, welches das Gewicht des Fahrzeugs im Vergleich zu konventionellen Fahrzeugen erhöht. Die Leichtbau Anforderung ist daher in gewisser Weise in Konflikt mit einer innovativen Idee, schnellem Fahren und vor allem dem omnidirektionalen Fahren.

Durch die Verwendung einer elastischen Federung konnte auf ein Gehäuse für den Schutz der Elektronik verzichtet werden und zusätzlich konnten die dadurch entstandenen Hohlräume für technische Bauteile und Verbindungskabel genutzt werden.

## 5 Zukünftige Entwicklungsmöglichkeiten

Obwohl das Praktikum erfolgreich verlief, werden in diesem Kapitel noch Verbesserungsvorschläge und Ausbaumöglichkeiten beleuchtet.

## 5.1 Regler

Die Messwerte des Gyroskops können für einen Regelkreis (Abb. 5.1) benutzt werden.

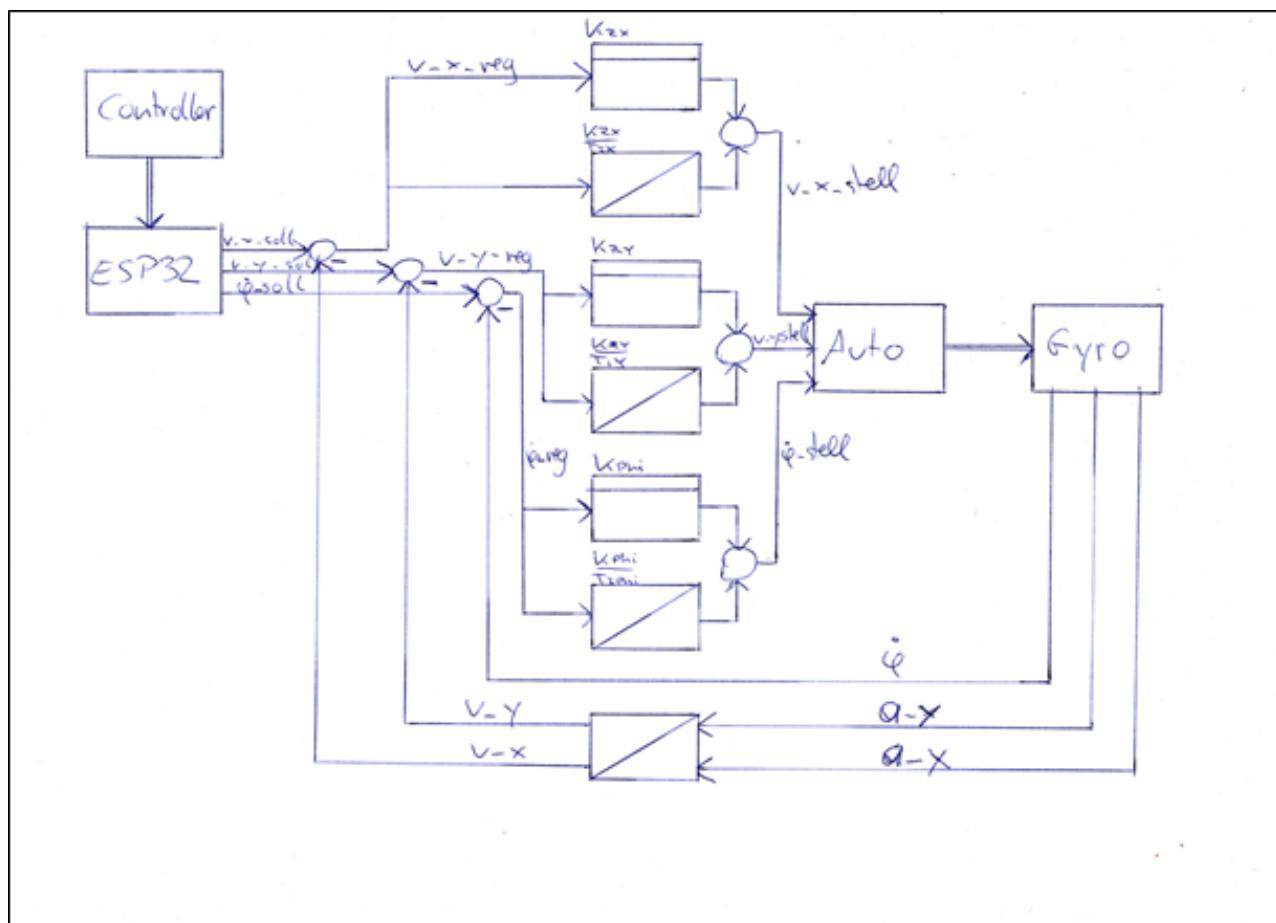


Abbildung 5.1: Blockschaltbild Regler

Durch die Eingaben des Spiele-Controllers und deren Interpretation durch das ESP-32 werden Sollwerte für Geschwindigkeiten in allen Achsen vorgegeben. Das Gyroskop misst Beschleunigung in X- und Y-Richtung und die Winkelgeschwindigkeit der Rotation.

Deswegen müssen die Beschleunigungswerte einmal integriert werden, um auf die momentane Ist-Geschwindigkeit in beiden Richtungen zu kommen. Das übernimmt der Integrator ganz unten.

Anschließend wird von den Soll- und Ist-Werten die Differenz gebildet. Über diese Regelungsdifferenz wird jeweils durch einen PI-Regler (Proportional und Integrationsglied) mit empirisch bestimmten Faktoren die Stellgröße ermittelt. Das Integrationsglied der PI-Regler ist dabei wichtig für die stationäre Genauigkeit.

Die Stellgrößen werden nach Umrechnung an die Motoren weitergeleitet, was wiederum zu veränderten Messgrößen am Gyroskop führt. Damit ist der Kreis geschlossen.

## 5.2 Verbesserungen

Stillstand bedeutet Rückschritt. Daher werden bei zukünftigen TEGGLA Versionen sowohl Hardware – als auch Software – Komponenten angepasst, um jedes Fahrzeug besser als dessen Vorgänger zu machen.

### 5.2.1 Hardware

Für das omnidirektionale Fahren mit den Mecanum-Rädern sind Stepper-Motoren besser geeignet als die aktuell verwendeten DC-Motoren, da diese eine vorgegebene Drehzahl besser halten können. Daher werden zukünftige TEGGLAs mit vier Stepper-Motoren ausgestattet sein.

Eine weitere hardwarenahe Verbesserung ist die Neugestaltung des Hauptschalter am Rahmen des Fahrzeugs. Dieser ist aktuell so konstruiert, dass der Hauptschalter bei einer Kollision des Fahrzeugs mit Objekten unbeabsichtigt betätigt werden kann. Daher wird in der nächsten Version des TEGGLA der Hauptschalter nach innen zeigen, um vor Kollisionen und unerwünschtem Betätigen geschützt zu sein.

Die Verbesserung des Planetengetriebes ist zwar ein zeitaufwändiges Vorhaben, jedoch würde sich das ebenfalls positiv auf die Übersetzungsrate und somit auf das Fahren auswirken. Durch das Anpassen von Stellgrößen im CAD Modell und durch einige Druckversuche kann immer weniger Spiel im Getriebe erreicht werden. Eine Alternative wäre die Verwendung eines besseren, genaueren 3D-Druckers, da beim 3D-Druck eines Planetengetriebes bereits sehr

geringe Druckungenaugkeiten problematisch sind.

Die vorerst letzten hardwarenahen Verbesserungsmöglichkeiten beziehen sich auf den Ladestecker und die Rollen der Mecanum-Räder. Der aktuell verwendete Ladestecker (XT60) ist für ein häufiges Aufladen des Fahrzeugs nicht gut geeignet, da sich das Kabel schwer ein- und ausstecken lässt.

Die Rollen der Mecanum-Räder sollen bei zukünftigen Versionen des TEGGLA mehr Grip haben, was eventuell durch die Verwendung eines anderen Filaments erreicht werden kann.

### 5.2.2 Software

Die Weboberfläche für die Kommunikation und Steuerung des TEGGLA soll fortlaufend erweitert und verbessert werden. Beispielsweise kann durch das Integrieren der Beschleunigung, welche vom Gyroscope gemessen wird, zusätzlich auch noch die aktuelle Geschwindigkeit des TEGGLA angezeigt werden.

Eine weitere softwarenahe Verbesserung des TEGGLA wäre das Beheben von Verbindungsproblemen. Bei dem Testen des fertigen Fahrzeugs kam es teilweise zu Verbindungsabbrüchen zum WLAN des TEGGLA. Dadurch konnte das Fahrzeug natürlich nicht mehr gesteuert werden und zusätzlich wurde der letzte erhaltene Befehl dauerhaft ausgeführt. Dieses Verbindungsproblem kann durch zwei Schritte verbessert werden: Vorerst sollte der Code so angepasst werden, dass bei einem Verbindungsabbruch kein Befehl mehr ausgeführt wird und der TEGGLA somit keine unkontrollierten Bewegungen durchführt. Anschließend können automatische Reconnects implementiert werden, welche bei einem Verbindungsabbruch in kleinen Zeitabständen versuchen, die Verbindung automatisch wiederherzustellen. Dadurch kann die Zeit ohne Verbindung mit dem TEGGLA minimiert werden.

# 6 Zusammenfassung

In diesem abschließenden Kapitel werden die gesammelten Erfahrungen reflektiert, sowie die wichtigsten erlernten, beziehungsweise verbesserten Fähigkeiten beim Entwickeln des TEGGLA zusammengefasst.

## 6.1 Fazit

Das Entwickeln des TEGGLA hat dem gesamten Team sehr viel Spaß gemacht. Um das erwünschte Ergebnis zu erreichen, wurde der Umgang mit vielen neuen Technologien erlernt beziehungsweise das bereits vorhandene Wissen vertieft. Vor allem das Konstruieren in CAD, die Anfertigung von technischen Zeichnungen und das tatsächliche Drucken und Zusammenbauen der Baugruppen hatten einen großen Mehrwert für unsere zukünftigen Projekte. Die wichtigsten verwendeten Technologien waren:

### **Creo**

Konstruieren aller Bauteile des TEGGLA.

### **C++**

Programmierung des MicroController auf einer Hardwarenahen Ebene.

### **JavaScript / HTML5**

Weboberfläche mit Sockets für die Kommunikation und Steuerung des TEGGLA.

### **Github**

Repository für die Sicherung, Versionierung, Verteilung und Aktualisierung aller Daten, technischen Zeichnungen, Skizzen, Berichte, Präsentationen und Quellcode.

## 6.2 Errungene Erfahrung

Der wohl zeitaufwändigste und schwierigste Aspekt des Projekts war der 3D-Druck eines Planetengetriebes, welches die erforderliche Übersetzung für das omnidirektionale Fahren bereitstellen kann. Bereits geringe Druckerungenauigkeiten führten zu nicht nutzbaren Getriebeversionen und daher wurden viele Anläufe benötigt, um ein passendes Planetengetriebe zu entwickeln. Für zukünftige Projekte mit 3D-Druckern wird das berücksichtigt und der Fokus bei der Entwicklung darauf gelegt.

Als sehr positiv war die zeitliche Planung und Zusammenarbeit des Teams zu bewerten. Trotz unerwarteten Mehraufwänden für Teilaufgaben, kam es zu keinerlei zeitlichen Engpässen und dadurch zu einem qualitativ hochwertigen Endprodukt. Eine möglichst präzise Aufgabendefinition und -verteilung am Projektanfang ist enorm wichtig, um dies sicherstellen zu können. Zusätzlich waren die Meilensteintermine des Lehrstuhls ebenfalls hilfreich, um sich an den Terminplan zu halten.

Die Nutzung eines GIT Repositories war ebenfalls äußerst gewinnbringend, da die Versionierung, Verteilung, Aktualisierung und Sicherung jederzeit und für alle Teammitglieder gewährleistet werden konnte.

Insgesamt wurden enorm viele Erfahrungen über den Entwicklungsprozess eines lenkbaren Fahrzeugs gesammelt, welche sich natürlich auch auf andere Produkte und Projekte abstrahieren lassen. Die einzelnen Entwicklungsschritte, nämlich die Auswahl einer Idee, das Anfertigen eines Lastenhefts, die Rollen- und Aufgabenverteilung innerhalb des Teams sowie die zeitliche Einhaltung von Terminen sind alles wichtige Kenntnisse für unsere zukünftigen Projekte.

# Anhang

# A Quellcode

## A.1 ESP32

```
1 #include "communication.h"
2
3 AsyncWebSocketClient *ws_client = nullptr;
4
5 void Communication::onWSData(AsyncWebSocket *server, AsyncWebSocketClient *client,
6     AwsEventType type, uint8_t *data, size_t len) {
7     ws_client = client;
8
9     int16_t *d16 = (int16_t *)data;
10    int16_t id = d16[0];
11
12    switch (id) {
13        case OmniMessageType::DRIVE:
14            onDrive(d16[1], d16[2], d16[3], d16[4]);
15            break;
16
17        default:
18            Serial.printf("No such command %i\n", id);
19            break;
20    }
21
22 void Communication::onDrive(int16_t x1, int16_t y1, int16_t x2, int16_t y2) {
23     // Serial.printf("Driving with speed: %i, %i and %i, %i now\n", x1, y1, x2, y2);
24     Movement::drive(y1, x1, x2);
25 }
26
27 void Communication::sendCurrGyro(XYZ speed, XYZ accel, XYZ rot) {
28     int16_t buff[10] = {OmniMessageType::CURR_GYRO, speed.x, speed.y, speed.z, accel.x,
29     accel.y, accel.z, rot.x, rot.y, rot.z};
30     ws->binaryAll((uint8_t *)buff, 20);
31 }
```

```

31
32 void Communication::sendCurrGyBatComb(int16_t sX, int16_t sY, int16_t sZ, int16_t aX,
33   int16_t aY, int16_t aZ, int16_t rX, int16_t rY, int16_t rZ, int16_t bat, int16_t temp)
34 {
35   int16_t buff[12] = {OmniMessageType::CURR_GY_BAT_COMB, sX, sY, sZ, aX, aY, aZ, rX, rY,
36   rZ, bat, temp};
37   ws->binaryAll((uint8_t *)buff, 24);
38 }
39
40 void Communication::sendCurrMotor(int16_t vl, int16_t vr, int16_t hl, int16_t hr) {
41   int16_t buff[5] = {OmniMessageType::CURR_MOTOR, vl, vr, hl, hr};
42   // Serial.println("should send now");
43   ws->binaryAll((uint8_t *)buff, 10);
44 }
45
46 void Communication::sendCurrBattery(int16_t cell1, int16_t cell2) {
47   int16_t buff[5] = {OmniMessageType::CURR_BATTERY, cell1, cell2};
48   // Serial.println("should send now");
49   ws->binaryAll((uint8_t *)buff, 10);
50 }

51 AsyncWebSocket *Communication::ws;

```

Listing A.1: communication.cpp

```

1 #pragma once
2
3 #include "movement.h"
4 #include <ESPAsyncWebServer.h>
5
6 /**
7  * Limitations due to js:
8  * Buffer has to be all the same type
9  *
10 * Type is int16 as well at the start of the array
11 *
12 */
13 enum OmniMessageType {
14   DRIVE = 0,           //L4 {x1} int16 [-1023, 1023] :: {y1} int16 [-1023, 1023] :: {x2} int16 [-1023, 1023] :: {y2} int16 [-1023, 1023]
15   CURR_GYRO = 1,        //L18 {speed m/s} (x) int16 (y) int16 (z) int16 :: {accel m/s^2} (x) int16 (y) int16 (z) int16

```

```

16     CURR_BATTERY = 2,      //L1 {cell1} int16 [0, 2^12]
17     CURR_MOTOR = 3,       //L4 {vl} int16 [-1023, 1023] :: {vr} int16 [-1023, 1023] :: {hl} int16 [-1023, 1023] :: {hr} int16 [-1023, 1023]
18     CURR_GY_BAT_COMB = 4, //L20 {speed m/s} (x) int16 (y) int16 (z) int16 :: {accel m/s^2} (x) int16 (y) int16 (z) int16 :: {rot} (x) int16 (y) int16 (z) int16 :: {bat} int16 [0, 2^12] :: {temp} int16
19 };
20
21 struct XYZ {
22     int16_t x;
23     int16_t y;
24     int16_t z;
25 };
26
27 class Communication {
28 public:
29     static void onWSData(AsyncWebSocket *server, AsyncWebSocketClient *client,
30     AwsEventType type, uint8_t *data, size_t len);
31     static void onDrive(int16_t x1, int16_t y1, int16_t x2, int16_t y2);
32     static void sendCurrGyro(XYZ speed, XYZ accel, XYZ rot);
33     static void sendCurrBattery(int16_t cell1, int16_t cell2);
34     static void sendCurrGyBatComb(int16_t sX, int16_t sY, int16_t sZ, int16_t aX, int16_t
35     aY, int16_t aZ, int16_t rX, int16_t rY, int16_t rZ, int16_t bat, int16_t temp);
36     static void sendCurrMotor(int16_t vl, int16_t vr, int16_t hl, int16_t hr);
37
38     static AsyncWebSocket *ws;
39 };

```

Listing A.2: communication.h

```

1 #include "movement.h"
2
3 // ----- Motor Class
4
5 // ##### Functions #####
6 void Motor::setSpeed(int16_t speed) {
7     // disable before changing direction
8     // (probably not needed) safety to prevent shoutthrough
9     ledcWrite(channel, 0);
10    if (speed > 0) {
11        digitalWrite(pin_dir1, LOW);
12        digitalWrite(pin_dir2, HIGH);

```

```
12 } else if (speed < 0) {
13     digitalWrite(pin_dir1, HIGH);
14     digitalWrite(pin_dir2, LOW);
15 } else {
16     digitalWrite(pin_dir1, LOW);
17     digitalWrite(pin_dir2, LOW);
18     return;
19 }
20
21 ledcWrite(channel, abs(speed));
22 }
23
24 void Motor::stop() {
25     ledcWrite(channel, 0);
26     digitalWrite(pin_dir1, LOW);
27     digitalWrite(pin_dir2, LOW);
28 }
29
30 // ----- Movement Class
31 // ##### Attributes #####
32 Motor Movement::MOTOR_VL = Motor(CH_MOTOR_VL, PIN_MOTOR_VL_DIR1, PIN_MOTOR_VL_DIR2);
33 Motor Movement::MOTOR_VR = Motor(CH_MOTOR_VR, PIN_MOTOR_VR_DIR1, PIN_MOTOR_VR_DIR2);
34 Motor Movement::MOTOR_HL = Motor(CH_MOTOR_HL, PIN_MOTOR_HL_DIR1, PIN_MOTOR_HL_DIR2);
35 Motor Movement::MOTOR_HR = Motor(CH_MOTOR_HR, PIN_MOTOR_HR_DIR1, PIN_MOTOR_HR_DIR2);
36
37 // ##### Functions #####
38 void Movement::initPWM() {
39     Serial.println("Initing PWMs");
40
41     // Setting all pins to output
42     pinMode(PIN_LED, OUTPUT);
43     pinMode(PIN_MOTOR_VL_EN, OUTPUT);
44     pinMode(PIN_MOTOR_VL_DIR1, OUTPUT);
45     pinMode(PIN_MOTOR_VL_DIR2, OUTPUT);
46
47     pinMode(PIN_MOTOR_VR_EN, OUTPUT);
48     pinMode(PIN_MOTOR_VR_DIR1, OUTPUT);
49     pinMode(PIN_MOTOR_VR_DIR2, OUTPUT);
50
51     pinMode(PIN_MOTOR_HL_EN, OUTPUT);
52     pinMode(PIN_MOTOR_HL_DIR1, OUTPUT);
53     pinMode(PIN_MOTOR_HL_DIR2, OUTPUT);
```

```
54
55     pinMode(PIN_MOTOR_HR_EN, OUTPUT);
56     pinMode(PIN_MOTOR_HR_DIR1, OUTPUT);
57     pinMode(PIN_MOTOR_HR_DIR2, OUTPUT);
58
59     // Disable all pins by default to not possibly cause shootthrough
60     digitalWrite(PIN_MOTOR_VL_EN, LOW);
61     digitalWrite(PIN_MOTOR_VL_DIR1, LOW);
62     digitalWrite(PIN_MOTOR_VL_DIR2, LOW);
63
64     digitalWrite(PIN_MOTOR_VR_EN, LOW);
65     digitalWrite(PIN_MOTOR_VR_DIR1, LOW);
66     digitalWrite(PIN_MOTOR_VR_DIR2, LOW);
67
68     digitalWrite(PIN_MOTOR_HL_EN, LOW);
69     digitalWrite(PIN_MOTOR_HL_DIR1, LOW);
70     digitalWrite(PIN_MOTOR_HL_DIR2, LOW);
71
72     digitalWrite(PIN_MOTOR_HR_EN, LOW);
73     digitalWrite(PIN_MOTOR_HR_DIR1, LOW);
74     digitalWrite(PIN_MOTOR_HR_DIR2, LOW);
75
76     // configure LED PWM functionalitites
77     ledcSetup(CH_LED, PWM_FREQ, PWM_RESOLUTION);
78     ledcSetup(CH_MOTOR_VL, PWM_FREQ, PWM_RESOLUTION);
79     ledcSetup(CH_MOTOR_VR, PWM_FREQ, PWM_RESOLUTION);
80     ledcSetup(CH_MOTOR_HL, PWM_FREQ, PWM_RESOLUTION);
81     ledcSetup(CH_MOTOR_HR, PWM_FREQ, PWM_RESOLUTION);
82
83     // attach the channel to the GPIO2 to be controlled
84     ledcAttachPin(PIN_LED, CH_LED);
85     ledcAttachPin(PIN_MOTOR_VL_EN, CH_MOTOR_VL);
86     ledcAttachPin(PIN_MOTOR_VR_EN, CH_MOTOR_VR);
87     ledcAttachPin(PIN_MOTOR_HL_EN, CH_MOTOR_HL);
88     ledcAttachPin(PIN_MOTOR_HR_EN, CH_MOTOR_HR);
89
90     Serial.println("Initiated PWMS");
91 }
92
93 void Movement::drive(int controlFront, int controlSide, int controlTurn) {
94     if (abs(controlFront) < CONTROLLER_LOWER_LIMIT && abs(controlSide) <
95         CONTROLLER_LOWER_LIMIT && abs(controlTurn) < CONTROLLER_LOWER_LIMIT) {
96         // Serial.println("Stopping Motors");
```

```
96     MOTOR_VL.stop();
97     MOTOR_VR.stop();
98     MOTOR_HL.stop();
99     MOTOR_HR.stop();
100
101    Communication::sendCurrMotor(0, 0, 0, 0);
102    return;
103}
104
105 int speedVL = 0;
106 int speedVR = 0;
107 int speedHL = 0;
108 int speedHR = 0;
109
110 if (abs(controlFront) > 0 && controlSide == 0 && controlTurn == 0) {
111     speedVL = controlFront;
112     speedVR = controlFront;
113     speedHL = controlFront;
114     speedHR = controlFront;
115 } else if (abs(controlSide) > 0 && controlFront == 0 && controlTurn == 0) {
116     speedVL = controlSide;
117     speedVR = -controlSide;
118     speedHL = -controlSide;
119     speedHR = controlSide;
120 } else if (controlSide == 0 && controlFront == 0 && abs(controlTurn) > 0) {
121     speedVL = controlTurn;
122     speedVR = -controlTurn;
123     speedHL = controlTurn;
124     speedHR = -controlTurn;
125 } else {
126     driveAlgorithm(controlFront, controlSide, controlTurn, &speedVL, &speedVR,
127     &speedHL, &speedHR);
128 }
129
130 speedVL = speedVL / 1023.0 * USEABLE_UPPER_LIMIT + (1023 - USEABLE_UPPER_LIMIT) *
131 sgn(speedVL);
132 speedVR = speedVR / 1023.0 * USEABLE_UPPER_LIMIT + (1023 - USEABLE_UPPER_LIMIT) *
133 sgn(speedVR);
134 speedHL = speedHL / 1023.0 * USEABLE_UPPER_LIMIT + (1023 - USEABLE_UPPER_LIMIT) *
135 sgn(speedHL);
136 speedHR = speedHR / 1023.0 * USEABLE_UPPER_LIMIT + (1023 - USEABLE_UPPER_LIMIT) *
137 sgn(speedHR);
```

```

134     MOTOR_VL.setSpeed(speedVL);
135     MOTOR_VR.setSpeed(speedVR);
136     MOTOR_HL.setSpeed(speedHL);
137     MOTOR_HR.setSpeed(speedHR);
138
139     Communication::sendCurrMotor(speedVL, speedVR, speedHL, speedHR);
140 }
141
142 void Movement::driveAlgorithm(int controlFront, int controlSide, int controlTurn, int
143 *speedVL, int *speedVR, int *speedHL, int *speedHR) {
144     double phi = atan2(controlSide, controlFront);
145
146     int vd = min((int)sqrt(controlFront * controlFront + controlSide * controlSide), 1023);
147     vd -= controlTurn / 2;
148     int vphi = controlTurn / 2;
149
150     double s = vd * sin(phi + PI / 4);
151     double c = vd * cos(phi + PI / 4);
152
153     *speedVL = s + vphi;
154     *speedVR = c - vphi;
155     *speedHL = c + vphi;
156     *speedHR = s - vphi;
157 }
```

Listing A.3: movement.cpp

```

1 #pragma once
2
3 #include "communication.h"
4 #include "util.h"
5 #include <Arduino.h>
6 #include <math.h>
7
8 class Motor {
9 private:
10     char channel;
11     char pin_dir1;
12     char pin_dir2;
13
14 public:
```

```
15     Motor(char channel, char pin_dir1, char pin_dir2) : channel(channel),
16     pin_dir1(pin_dir1), pin_dir2(pin_dir2){};
17     ~Motor() {}
18
19     /**
20      * speed in [-1023; 1023]
21      */
22     void setSpeed(int16_t speed);
23
24     void stop();
25 };
26
27 class Movement {
28 private:
29     /* data */
30 public:
31     static const char PIN_LED = 2;
32
33     static Motor MOTOR_VL;
34     static Motor MOTOR_VR;
35     static Motor MOTOR_HL;
36     static Motor MOTOR_HR;
37
38     /** Amount of values down from 1023 which can be used */
39     static const int USEABLE_UPPER_LIMIT = 700;
40     /** Wont move below this limit */
41     static const int CONTROLLER_LOWER_LIMIT = 50;
42
43     static const char PIN_MOTOR_VL_EN = 23;
44     static const char PIN_MOTOR_VL_DIR1 = 18;
45     static const char PIN_MOTOR_VL_DIR2 = 19;
46
47     static const char PIN_MOTOR_VR_EN = 32;
48     static const char PIN_MOTOR_VR_DIR1 = 12;
49     static const char PIN_MOTOR_VR_DIR2 = 13;
50
51     static const char PIN_MOTOR_HL_EN = 16;
52     static const char PIN_MOTOR_HL_DIR1 = 17;
53     static const char PIN_MOTOR_HL_DIR2 = 5;
54
55     static const char PIN_MOTOR_HR_EN = 4;
56     static const char PIN_MOTOR_HR_DIR1 = 3;
57     static const char PIN_MOTOR_HR_DIR2 = 15;
```

```
57
58     static const int PWM_FREQ = 500;
59
60     static const char CH_LED = 0;
61
62     static const char CH_MOTOR_VL = 1;
63
64     static const char CH_MOTOR_VR = 2;
65
66     static const char CH_MOTOR_HL = 3;
67
68     static const char CH_MOTOR_HR = 4;
69
70     static const char PWM_RESOLUTION = 10; //Resolution 8, 10, 12, 15
71
72
73     static void initPWM();
74
75
76     static void driveMotor(char channel, char dir1, char dir2, int speed);
77
78     /**
79      * Converts the given controls into wheel speeds
80      * Algorithm source:
81      * http://eprints.utm.edu.my/16543/1/Omni%20Directional%20Control%20Algorithm%20For%20Mecanum%20wheel%20drive.pdf
82      */
83
84     static void drive(int controlFront, int controlSide, int controlTurn);
85
86
87     private:
88
89         static void driveAlgorithm(int controlFront, int controlSide, int controlTurn, int
90             *speedVL, int *speedVR, int *speedHL, int *speedHR);
91
92     };
93
94 }
```

Listing A.4: movement.h

```
1 #include "util.h"
2
3 /************************************************************************
4  * high precision sine/cosine
5  *
6  * Source: https://gist.github.com/geraldyeo/988116
7  *
8  *
9  ******************************************************************************/
10 void coassin(float x, float *outCos, float *outSin) {
11     float sin, cos;
12
13     //always wrap input angle to -PI..PI
14     if (x < -3.14159265)
15         x += 6.28318531;
```

```
16     else if (x > 3.14159265)
17         x -= 6.28318531;
18
19 //compute sine
20 if (x < 0) {
21     sin = 1.27323954 * x + .405284735 * x * x;
22
23     if (sin < 0)
24         sin = .225 * (sin * -sin - sin) + sin;
25     else
26         sin = .225 * (sin * sin - sin) + sin;
27 } else {
28     sin = 1.27323954 * x - 0.405284735 * x * x;
29
30     if (sin < 0)
31         sin = .225 * (sin * -sin - sin) + sin;
32     else
33         sin = .225 * (sin * sin - sin) + sin;
34 }
35
36 //compute cosine: sin(x + PI/2) = cos(x)
37 x += 1.57079632;
38 if (x > 3.14159265)
39     x -= 6.28318531;
40
41 if (x < 0) {
42     cos = 1.27323954 * x + 0.405284735 * x * x;
43
44     if (cos < 0)
45         cos = .225 * (cos * -cos - cos) + cos;
46     else
47         cos = .225 * (cos * cos - cos) + cos;
48 } else {
49     cos = 1.27323954 * x - 0.405284735 * x * x;
50
51     if (cos < 0)
52         cos = .225 * (cos * -cos - cos) + cos;
53     else
54         cos = .225 * (cos * cos - cos) + cos;
55 }
56
57 *outSin = sin;
58 *outCos = cos;
```

```

59 }
60
61
62 /**
63  * Branchless signum function
64  */
65 int sgn(int val) {
66     return (0 < val) - (val < 0);
67 }

```

Listing A.5: util.cpp

```

1 #pragma once
2
3 void coassin(float x, float *outCos, float *outSin);
4
5 int sgn(int val);

```

Listing A.6: util.h

```

1 #include "MPU6050.h"
2 #include "communication.h"
3 #include "movement.h"
4 #include <Arduino.h>
5 #include <ESPAsyncWebServer.h>
6 #include <SPIFFS.h>
7 #include <WiFi.h>
8 #include <Wire.h>
9
10 // Replace with your network credentials
11 const char *ssid = "ESP32-OmniMove";
12 const char *password = "123456789";
13
14 // const byte DNS_PORT = 53;
15 const IPAddress apIP = IPAddress(192, 168, 4, 1);
16
17 AsyncWebServer server(80);
18 AsyncWebSocket ws("/ws");
19 MPU6050 mpu;
20

```

```
21 void onEvent(AsyncWebSocket *server, AsyncWebSocketClient *client, AwsEventType type, void
22 *arg, uint8_t *data, size_t len) {
23
24     switch (type) {
25         case WS_EVT_CONNECT:
26             Serial.printf("Client connected from %s\n", client->remoteIP().toString().c_str());
27             break;
28
29         case WS_EVT_DATA:
30             Communication::onWSData(server, client, type, data, len);
31             break;
32
33         default:
34             break;
35     }
36 }
37
38 void setup() {
39     // enableCore1WDT();
40     Serial.begin(115200);
41
42     Wire.begin();
43
44     mpu.initialize();
45
46     // mpu.CalibrateAccel_MP6500(6);
47     // mpu.CalibrateGyro(6);
48
49     // mpu.PrintActiveOffsets_MP6500();
50     mpu.setXGyroOffset(96);
51     mpu.setYGyroOffset(92);
52     mpu.setZGyroOffset(-20);
53
54     // Serial.printf("\n");
55     Movement::initPWM();
56
57     // enable AP with dns
58     WiFi.mode(WIFI_AP);
59
60     // Setup websockets
61     ws.onEvent(onEvent);
62     server.addHandler(&ws);
63     Communication::ws = &ws;
```

```
63     ws.enable(true);  
64  
65     // Initialize SPIFFS  
66     if (!SPIFFS.begin(true)) {  
67         while (true) {  
68             Serial.println("An Error has occurred while mounting SPIFFS");  
69             delay(1000);  
70         }  
71         return;  
72     } else {  
73         Serial.println("Mounted SPIFFS successfully");  
74     }  
75  
76     // Route for root / web page  
77     server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {  
78         // Serial.println("request on index");  
79         request->send(SPIFFS, "/index.html", String(), false, nullptr);  
80     });  
81  
82     // Route to load style.css file  
83     server.on("/style.css", HTTP_GET, [](AsyncWebServerRequest *request) {  
84         // Serial.println("request on style");  
85         request->send(SPIFFS, "/style.css", "text/css");  
86     });  
87  
88     // Route to load code.js file  
89     server.on("/code.js", HTTP_GET, [](AsyncWebServerRequest *request) {  
90         // Serial.println("request on code");  
91         request->send(SPIFFS, "/code.js", "text/javascript");  
92     });  
93  
94     // Route to load code.js file  
95     server.on("/progressbar.min.js", HTTP_GET, [](AsyncWebServerRequest *request) {  
96         // Serial.println("request on progressbar");  
97         request->send(SPIFFS, "/progressbar.min.js", "text/javascript");  
98     });  
99  
100    // WiFi.softAPConfig(apIP, apIP, IPAddress(255, 255, 255, 0));  
101    WiFi.setSleep(false);  
102    WiFi.softAP(ssid, password, 6, 1);  
103    delay(1000);  
104  
105    server.begin();
```

```

106 }
107
108 void loop() {
109
110     // put your main code here, to run repeatedly:
111     // dnsServer.processNextRequest();
112     uint16_t v = analogRead(39);
113     int16_t x, y, z, gx, gy, gz, temp;
114
115     temp = mpu.getTemperature();
116     mpu.getMotion6(&x, &y, &z, &gx, &gy, &gz);
117
118     // Serial.printf("x: %6.2fg, y: %6.2fg, z: %6.2fg, gx: %6.2f°/s, gy: %6.2f°/s, gz:
119     // %6.2f°/s, temp: %6.2f°C\r", x / 16384.0, y / 16384.0, z / 16384.0, gx / 250.0, gy /
120     // 250.0, gz / 250.0, temp / 340.0 + 36.53);
121     // power = (v / 4095 * 3.1 + .1) * 3;
122     Communication::sendCurrGyBatComb(0, 0, 0, x, y, z, gx, gy, gz, v, temp);
123     delay(1000);
124 }

```

Listing A.7: main.cpp

## A.2 Webpage

```

1 <!DOCTYPE html>
2 <html>
3
4 <head>
5     <title>ESP32 Web Server</title>
6     <meta charset="utf-8" />
7     <meta name="viewport" content="width=device-width, initial-scale=1">
8     <link rel="icon" href="data:, ">
9     <link rel="stylesheet" type="text/css" href="style.css">
10 </head>
11
12 <body>
13     <h1>ESP32 Web Server</h1>
14     <p>WebSocked Connected: <strong id="wsState">CONNECTING</strong></p>
15     <p><button class="button" onclick="sendSpeed()">Test Button</button></p>
16

```

```
17
18 <div class="wrapper">
19   <div class="left">
20     <h2>Measurements</h2>
21     <p>Acceleration: <strong id="valAccelTotal">0</strong> m/s2
22       <br /> <span id="valAccel">[x: 0, y: 0, z: 0]</span>
23     </p>
24
25     <p>Speed: <strong id="valSpeedTotal">0</strong> m/s
26       <br /> <span id="valSpeed">[x: 0, y: 0, z: 0]</span>
27
28     <p>Rotation: <strong id="valRotTotal">0</strong> rad/s2
29       <br /> <span id="valRot">[x: 0, y: 0, z: 0]</span></p>
30
31   </div>
32
33   <div class="right">
34     <h2>Battery</h2>
35     <progress id="progBattery" value="60" max="100"></progress>
36
37     <strong id="valBatteryTotal">-1</strong>%
38     <p>Voltage <strong id="valVolt">-1</strong> V</p>
39
40     <h2>Temperature</h2>
41     <p><strong id="valTemp">-1</strong> °C</p>
42   </div>
43 </div>
44
45
46 <div>
47   <svg viewBox="0 0 205 10" preserveAspectRatio="none" style="width: 100%; height: 100%;">
48     <path id="pathVL" d="M50,2 L100,2 0,2 50,2" stroke="#FFEA82" stroke-width="4" fill-opacity="0"
49       style="stroke-dasharray: 100, 100; stroke-dashoffset: 0;"></path>
50     <path id="pathHL" d="M50,8 L100,8 0,8 50,8" stroke="#FFEA82" stroke-width="4"
51       fill-opacity="0"
52       style="stroke-dasharray: 100, 100; stroke-dashoffset: 0;"></path>
53
54     <path id="pathVR" d="M155,2 L205,2 105,2 155,2" stroke="#FFEA82" stroke-width="4"
55       fill-opacity="0"
56       style="stroke-dasharray: 100, 100; stroke-dashoffset: 0;"></path>
```

```

55      <path id="pathHR" d="M155,8 L205,8 105,8 155,8" stroke="#FFEA82" stroke-width="4"
56      fill-opacity="0"
57      style="stroke-dasharray: 100, 100; stroke-dashoffset: 0;"></path>
58
59      <path d="M 50 0 L 50 4" stroke="black" stroke-width="0.5" fill="none" />
60      <path d="M 50 6 L 50 10" stroke="black" stroke-width="0.5" fill="none" />
61      <path d="M 155 0 L 155 4" stroke="black" stroke-width="0.5" fill="none" />
62      <path d="M 155 6 L 155 10" stroke="black" stroke-width="0.5" fill="none" />
63  </svg>
64 </div>
65 <div>
66  <h5>Select Controller</h5>
67  <button class="buttonsmall" onclick="updateControllerList()">Test Controller</button>
68  <input type="checkbox" id="controllerEnabled" checked>
69  <label for="controllerEnabled">Enable Controller?</label>
70  <ul id="controllerList">
71    <li onclick="onControllerSelected(0)">Test</li>
72
73  </ul>
74 </div>
75
76 </body>
77
78
79 <script type="text/javascript" src="progressbar.min.js"></script>
80 <script type="text/javascript" src="code.js"></script>
81
82 </html>

```

Listing A.8: index.html

```

1 let ws = new WebSocket("ws://192.168.4.1/ws");
2 // let ws = new WebSocket("ws://demos.kaazing.com/echo");
3 ws.binaryType = 'arraybuffer';
4
5 let barVL = new ProgressBar.Path('#pathVL', { easing: 'easeInOut', duration: 140, });
6 let barVR = new ProgressBar.Path('#pathVR', { easing: 'easeInOut', duration: 140, });
7 let barHL = new ProgressBar.Path('#pathHL', { easing: 'easeInOut', duration: 140, });
8 let barHR = new ProgressBar.Path('#pathHR', { easing: 'easeInOut', duration: 140, });
9 barVL.set(0.1)
10 barVR.set(0.1)

```

```
11 barHL.set(0.1)
12 barHR.set(0.1)
13
14 let selectedControllerIndex = -1;
15 window.onload = function () {
16     setInterval(controllerFunc, 100);
17     updateControllerList();
18 };
19
20 //##region >>> WebSocket
21 ws.onopen = (event) => {
22     document.getElementById("wsState").innerHTML = "OPEN";
23 };
24
25 ws.onerror = (event) => {
26     document.getElementById("wsState").innerHTML = "ERROR";
27 };
28
29 ws.onclose = (event) => {
30     document.getElementById("wsState").innerHTML = "CLOSED";
31 };
32
33 ws.onmessage = function (event) {
34     // console.log("WebSocket message received:", event);
35     let dv = new DataView(event.data);
36     let id = dv.getInt16(0, true);
37
38     switch (id) {
39         case 1: //gyro
40             let a = [dv.getInt16(2, true), dv.getInt16(4, true), dv.getInt16(6, true)];
41             let s = [dv.getInt16(8, true), dv.getInt16(10, true), dv.getInt16(12, true)];
42             let r = [dv.getInt16(2, true), dv.getInt16(4, true), dv.getInt16(6, true)];
43
44             document.getElementById("valAccelTotal").innerHTML = Math.sqrt(a[0] * a[0] + a[1] *
45             a[1] + a[2] * a[2]).toFixed(2);
46             document.getElementById("valSpeedTotal").innerHTML = Math.sqrt(a[0] * s[0] + s[1] *
47             s[1] + s[2] * s[2]).toFixed(2);
48             document.getElementById("valRotTotal").innerHTML = Math.sqrt(r[0] * r[0] + r[1] *
49             r[1] + r[2] * r[2]).toFixed(2);
50
51             document.getElementById("valAccel").innerHTML = "[x: " + a[0] + ", y: " + a[1] + ",
52             z: " + a[2] + "];
```

```

49     document.getElementById("valSpeed").innerHTML = "[x: " + s[0] + ", y: " + s[1] + ", 
50     z: " + s[2] + "];";
51     document.getElementById("valRot").innerHTML = "[x: " + r[0] + ", y: " + r[1] + ", z: 
52     " + r[2] + "];";
53     break;
54
55
56     case 2: //battery
57     let c1 = (dv.getInt16(2, true) / 4095 * 3.1 + .1) * 3;
58
59     document.getElementById("valVolt").innerHTML = c1.toFixed(2);
60     break;
61
62     case 3: //motors
63     const UPPER_LIMIT = 300;
64
65     let vl = dv.getInt16(2, true) / (1023 * 4);
66     let vr = dv.getInt16(4, true) / (1023 * 4);
67     let hl = dv.getInt16(6, true) / (1023 * 4);
68     let hr = dv.getInt16(8, true) / (1023 * 4);
69
70     barVL.animate(vl);
71     barVR.animate(vr);
72     barHL.animate(hl);
73     barHR.animate(hr);
74     break;
75
76
77     case 4: //gyroBatComb
78     let s1 = [dv.getInt16(2, true), dv.getInt16(4, true), dv.getInt16(6, true)];
79     let a1 = [dv.getInt16(8, true) / 16384.0, dv.getInt16(10, true) / 16384.0,
80     dv.getInt16(12, true) / 16384.0];
81
82     let r1 = [dv.getInt16(14, true) / 250.0, dv.getInt16(16, true) / 250.0,
83     dv.getInt16(18, true) / 250.0];
84
85
86     document.getElementById("valAccelTotal").innerHTML = Math.sqrt(a1[0] * a1[0] + a1[1]
87     * a1[1] + a1[2] * a1[2]).toFixed(2);
88     document.getElementById("valSpeedTotal").innerHTML = Math.sqrt(s1[0] * s1[0] + s1[1]
89     * s1[1] + s1[2] * s1[2]).toFixed(2);
90     document.getElementById("valRotTotal").innerHTML = Math.sqrt(r1[0] * r1[0] + r1[1] *
91     r1[1] + r1[2] * r1[2]).toFixed(2);
92
93     document.getElementById("valAccel").innerHTML = "[x: " + a1[0].toFixed(2) + ", y: "
94     + a1[1].toFixed(2) + ", z: " + a1[2].toFixed(2) + "];";

```

```
83     document.getElementById("valSpeed").innerHTML = "[x: " + s1[0].toFixed(2) + ", y: "
84     + s1[1].toFixed(2) + ", z: " + s1[2].toFixed(2) + "J]";
85     document.getElementById("valRot").innerHTML = "[x: " + r1[0].toFixed(2) + ", y: " +
86     r1[1].toFixed(2) + ", z: " + r1[2].toFixed(2) + "J]";
87
88
89
90     let bat = (dv.getInt16(20, true) / 4095 * 3.1 + .1) * 3;
91     let batPercent = (bat - 6.5) / (7.8 - 6.5) * 100;
92     let temp = dv.getInt16(22, true) / 340.0 + 36.53;
93
94
95     document.getElementById("valVolt").innerHTML = bat.toFixed(2);
96     document.getElementById("progBattery").value = batPercent.toFixed(2);
97     document.getElementById("valBatteryTotal").innerHTML = batPercent.toFixed(0);
98
99
100    document.getElementById("valTemp").innerHTML = temp.toFixed(2);
101    break;
102
103
104    default:
105        break;
106    }
107};
108
109 //##endregion
110
111
112 //##region >>> Controller
113 window.addEventListener("gamepadconnected", (event) => {
114     console.log("A gamepad connected:");
115     console.log(event.gamepad);
116
117     updateControllerList();
118 });
119
120
121
122 // Lets the user select which connected controller to use
123 function updateControllerList() {
```

```
124 let inner = ""  
125  
126 let pads = navigator.getGamepads();  
127 console.log(pads);  
128  
129 for (let i = 0; i < pads.length; i++) {  
130     const element = pads[i];  
131     if (element === null)  
132         continue;  
133     if (selectedControllerIndex == i) {  
134         inner += "<li onclick=\"onControllerSelected(" + i + ")\"><strong>" + element.id +  
135         "</strong></li>";  
136     } else {  
137         inner += "<li onclick=\"onControllerSelected(" + i + ")\">" + element.id + "</li>";  
138     }  
139 }  
140  
141     document.getElementById("controllerList").innerHTML = inner;  
142 }  
143  
144 // gets called when a Controller gets selected in the List  
145 function onControllerSelected(index) {  
146     selectedControllerIndex = index;  
147     updateControllerList();  
148 }  
149  
150 let lastX1 = 0, lastX2 = 0, lastY1 = 0, lastY2 = 0;  
151 function controllerFunc() {  
152     if (!document.getElementById("controllerEnabled").checked)  
153         return;  
154  
155     let pads = navigator.getGamepads();  
156     if (pads === null || pads === undefined || pads[selectedControllerIndex] === null ||  
157         pads[selectedControllerIndex] === undefined)  
158         return;  
159     x1 = Math.floor(pads[selectedControllerIndex].axes[0] * 1023);  
160     y1 = -Math.floor(pads[selectedControllerIndex].axes[1] * 1023);  
161     x2 = Math.floor(pads[selectedControllerIndex].axes[2] * 1023);  
162     y2 = -Math.floor(pads[selectedControllerIndex].axes[3] * 1023);  
163  
164     if (Math.abs(x1) < 200)
```

```
165     x1 -= x1;
166     if (Math.abs(y1) < 200)
167         y1 -= y1;
168     if (Math.abs(x2) < 200)
169         x2 -= x2;
170     if (Math.abs(y2) < 200)
171         y2 -= y2;
172
173     if (lastX1 != x1 || lastY1 != y1 || lastX2 != x2 || lastY2 != y2) {
174         console.log(x1 + ", " + y1 + " : " + x2 + ", " + y2);
175         if (ws.readyState == WebSocket.OPEN) {
176             let buf = new Int16Array([0, x1, y1, x2, y2])
177             ws.send(buf);
178         }
179         lastX1 = x1;
180         lastX2 = x2;
181         lastY1 = y1;
182         lastY2 = y2;
183     }
184
185 }
186 // #endregion
187
188 // #region >>> Keyboard Control
189 let dirX = 0;
190 let dirY = 0;
191 let rotX = 0;
192 // a: 65, s: 83, d: 68, w: 87
193 document.addEventListener('keydown', function (event) {
194     let c = event.keyCode;
195     switch (event.keyCode) {
196         case 65:
197             dirX = -1;
198             break;
199         case 68:
200             dirX = 1;
201             break;
202         case 87:
203             dirY = 1;
204             break;
205         case 83:
206             dirY = -1;
207             break;
```

```
208     case 39:
209         rotX = 1;
210         break;
211     case 37:
212         rotX = -1;
213         break;
214     case 27:
215         dirY = 0;
216         dirX = 0;
217         break;
218     default:
219         break;
220     }
221
222     if (lastX1 != dirX * 1023 || lastY1 != dirY * 1023 || lastX2 != rotX * 1023 || lastY2 != 0) {
223         console.log(dirX * 1023 + ", " + dirY * 1023 + " : " + rotX * 1023 + ", " + 0);
224         if (ws.readyState == WebSocket.OPEN) {
225             let buf = new Int16Array([0, dirX * 1023, dirY * 1023, rotX * 1023, 0])
226             ws.send(buf);
227         }
228         lastX1 = dirX * 1023;
229         lastY1 = dirY * 1023;
230         lastX2 = rotX * 1023;
231         lastY2 = 0;
232     }
233
234     // console.log("x: " + dirX + " y: " + dirY);
235 });
236
237 document.addEventListener('keyup', function (event) {
238     let c = event.keyCode;
239     switch (event.keyCode) {
240         case 65:
241         case 68:
242             dirX = 0;
243             break;
244         case 87:
245         case 83:
246             dirY = 0;
247             break;
248         case 39:
249         case 37:
```

```

250     rotX = 0;
251     break;
252     default:
253     break;
254 }
255 console.log("x: " + dirX + " y: " + dirY);
256
257 if (lastX1 != dirX * 1023 || lastY1 != dirY * 1023 || lastX2 != rotX * 1023 || lastY2 != 0) {
258     console.log(dirX * 1023 + ", " + dirY * 1023 + " : " + rotX * 1023 + ", " + 0);
259     if (ws.readyState == WebSocket.OPEN) {
260         let buf = new Int16Array([0, dirX * 1023, dirY * 1023, rotX * 1023, 0])
261         ws.send(buf);
262     }
263     lastX1 = dirX * 1023;
264     lastY1 = dirY * 1023;
265     lastX2 = rotX * 1023;
266     lastY2 = 0;
267 }
268 });
269 //endregion
270
271
272
273 function sendSpeed() {
274     console.log("speed");
275     let arr = new Int16Array([2, Math.random() * 4048, Math.random() * 4048]);
276     let arr2 = new Int16Array([1,
277         Math.random() * 1023, Math.random() * 1023, Math.random() * 1023,
278         Math.random() * 1023, Math.random() * 1023, Math.random() * 1023,
279         Math.random() * 1023, Math.random() * 1023, Math.random() * 1023]);
280     ws.send(arr);
281     ws.send(arr2);
282 }

```

Listing A.9: code.js

```

1 html {
2     font-family: Helvetica;
3     display: inline-block;
4     margin: 0px auto;
5     text-align: center;

```

```
6  }
7  h1{
8    color: #0F3376;
9    padding: 2vh;
10 }
11 p{
12   font-size: 1.5rem;
13 }
14 .button {
15   display: inline-block;
16   background-color: #008CBA;
17   border: none;
18   border-radius: 4px;
19   color: white;
20   padding: 16px 40px;
21   text-decoration: none;
22   font-size: 30px;
23   margin: 2px;
24   cursor: pointer;
25 }
26
27 .buttonsmall {
28   display: inline-block;
29   background-color: #008CBA;
30   border: none;
31   border-radius: 4px;
32   color: white;
33   padding: 8px 20px;
34   text-decoration: none;
35   font-size: 16px;
36   margin: 2px;
37   cursor: pointer;
38 }
39
40 .button2 {
41   background-color: #f44336;
42 }
43
44 .wrapper {
45   display: flex;
46 }
47
48 .left {
```

```
49     flex: 0 0 65%;  
50 }  
51  
52 .right {  
53     flex: 1;  
54 }
```

Listing A.10: style.css

## A.3 Java

```
1 package main;  
2 import Toaster.Toaster;  
3 import Utils.*;  
4  
5 import java.awt.*;  
6 import java.awt.event.*;  
7 import javax.swing.*;  
8  
9 public class ConnectUI extends JFrame {  
10  
11     /**  
12      *  
13      */  
14     private static final long serialVersionUID = 1L;  
15  
16     private final Toaster toaster;  
17  
18     public static void main(String[] args) {  
19         new ConnectUI();  
20     }  
21  
22     private ConnectUI() {  
23         JPanel mainJPanel = getMainJPanel();  
24  
25         addLogo(mainJPanel);  
26  
27         addSeparator(mainJPanel);  
28  
29         addIPTextField(mainJPanel);
```

```
30
31     addPortTextField(mainJPanel);
32
33     addConnectButton(mainJPanel);
34
35     getContentPane().add(mainJPanel);
36
37     pack();
38     setVisible(true);
39     toFront();
40
41     Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
42     setLocation(screenSize.width / 2 - getWidth() / 2, screenSize.height / 2 -
43     getHeight() / 2);
44
45     toaster = new Toaster(mainJPanel);
46 }
47
48 private JPanel getMainJPanel() {
49     setUndecorated(true);
50
51     Dimension size = new Dimension(800, 400);
52
53     JPanel panel1 = new JPanel();
54     panel1.setSize(size);
55     panel1.setPreferredSize(size);
56     panel1.setBackground(UIUtils.COLOR_BACKGROUND);
57     panel1.setLayout(null);
58
59     TextFieldPort titel = new TextFieldPort();
60     titel.setEnabled(false);
61     titel.setEditable(false);
62     titel.setHorizontalAlignment(SwingConstants.CENTER);
63     titel.setBorderColor(UIUtils.COLOR_INTERACTIVE);
64     titel.setFont(new Font("Segoe UI", Font.PLAIN, 16));
65     titel.setText("OmniMove Wlan Verbindung");
66     titel.setBounds(68, 57, 250, 44);
67     panel1.add(titel);
68
69     panel1.setFocusable(true);
70     panel1.requestFocus();
71
72     MouseAdapter ma = new MouseAdapter() {
```

```
72         int lastX, lastY;
73
74     @Override
75     public void mousePressed(MouseEvent e) {
76         lastX = e.getXOnScreen();
77         lastY = e.getYOnScreen();
78     }
79
80     @Override
81     public void mouseDragged(MouseEvent e) {
82         int x = e.getXOnScreen();
83         int y = e.getYOnScreen();
84         setLocation(getLocationOnScreen().x + x - lastX, getLocationOnScreen().y +
85         y - lastY);
86         lastX = x;
87         lastY = y;
88     }
89 }
90
91     panel1.addMouseListener(ma);
92     panel1.addMouseMotionListener(ma);
93
94     addWindowListener(new WindowAdapter() {
95         @Override
96         public void windowClosing(WindowEvent e) {
97             System.exit(0);
98         }
99     });
100
101     return panel1;
102 }
103
104     private void addSeparator(JPanel panel1) {
105         JSeparator separator1 = new JSeparator();
106         separator1.setOrientation(SwingConstants.VERTICAL);
107         separator1.setForeground(UIUtils.COLOR_OUTLINE);
108         panel1.add(separator1);
109         separator1.setBounds(387, 41, 2, 316);
110     }
111
112     private void addLogo(JPanel panel1) {
113         JLabel label1 = new JLabel();
114         //label1.setSize(300,300);
```

```
114     label1.setFocusable(false);
115     label1.setBounds(57, 123, 273, 197);
116     //ImageIcon robot = new ImageIcon(new
117     //ImageIcon("/robot.png").getImage().getScaledInstance(20, 20, Image.SCALE_DEFAULT));
118     //label1.setIcon(robot);
119     Image robot = new ImageIcon(this.getClass().getResource("/robot.png")).getImage();
120     label1.setIcon(new ImageIcon(robot));
121     panel1.add(label1);
122 }
123
124 private void addIPTextField(JPanel panel1) {
125     TextFieldPort usernameField = new TextFieldPort();
126     //usernameField.setBorderColor(new Color(102, 205, 170));
127
128     usernameField.setText(UIUtils.PLACEHOLDER_TEXT_IP);
129     usernameField.setForeground(UIUtils.COLOR_OUTLINE);
130     usernameField.setBorderColor(UIUtils.COLOR_OUTLINE);
131
132     usernameField.setBounds(473, 109, 250, 44);
133     usernameField.addFocusListener(new FocusListener() {
134         @Override
135         public void focusGained(FocusEvent e) {
136             if (usernameField.getText().equals(UIUtils.PLACEHOLDER_TEXT_IP)) {
137                 usernameField.setText("");
138             }
139             usernameField.setForeground(Color.white);
140             usernameField.setBorderColor(UIUtils.COLOR_INTERACTIVE);
141         }
142
143         @Override
144         public void focusLost(FocusEvent e) {
145             if (usernameField.getText().isEmpty()) {
146                 usernameField.setText(UIUtils.PLACEHOLDER_TEXT_IP);
147             }
148             usernameField.setForeground(UIUtils.COLOR_OUTLINE);
149             usernameField.setBorderColor(UIUtils.COLOR_OUTLINE);
150         }
151     });
152
153     panel1.add(usernameField);
154 }
155
156 private void addPortTextField(JPanel panel1) {
```

```
156     TextFieldIP passwordField = new TextFieldIP();
157     //passwordField.setBorderColor(new Color(102, 205, 170));
158
159     passwordField.setText(UIUtils.PLACEHOLDER_TEXT_PORT);
160     passwordField.setForeground(UIUtils.COLOR_OUTLINE);
161     passwordField.setBorderColor(UIUtils.COLOR_OUTLINE);
162
163     passwordField.setBounds(473, 168, 250, 44);
164     passwordField.addFocusListener(new FocusListener() {
165
166         @Override
167         public void focusGained(FocusEvent e) {
168             if (passwordField.getText().equals(UIUtils.PLACEHOLDER_TEXT_PORT)) {
169                 passwordField.setText("");
170             }
171             passwordField.setForeground(Color.white);
172             passwordField.setBorderColor(UIUtils.COLOR_INTERACTIVE);
173         }
174
175         @Override
176         public void focusLost(FocusEvent e) {
177             if (passwordField.getText().isEmpty()) {
178                 passwordField.setText(UIUtils.PLACEHOLDER_TEXT_PORT);
179             }
180             passwordField.setForeground(UIUtils.COLOR_OUTLINE);
181             passwordField.setBorderColor(UIUtils.COLOR_OUTLINE);
182         }
183     });
184
185     passwordField.addKeyListener(new KeyAdapter() {
186
187         @Override
188         public void keyTyped(KeyEvent e) {
189             if (e.getKeyChar() == KeyEvent.VK_ENTER)
190                 loginEventHandler();
191         }
192     });
193
194
195     private void addConnectButton(JPanel panel1) {
196         final Color[] connectColors = {UIUtils.COLOR_INTERACTIVE, Color.white};
197
198         JLabel connectButton = new JLabel() {
```

```
199     /**
200      *
201      */
202     private static final long serialVersionUID = 1L;
203
204     @Override
205     protected void paintComponent(Graphics g) {
206         Graphics2D g2 = UIUtils.get2dGraphics(g);
207         super.paintComponent(g2);
208
209         Insets insets = getInsets();
210         int w = getWidth() - insets.left - insets.right;
211         int h = getHeight() - insets.top - insets.bottom;
212         g2.setColor(connectColors[0]);
213         g2.fillRoundRect(insets.left, insets.top, w, h, UIUtils.ROUNDNESS,
214                           UIUtils.ROUNDNESS);
215
216         FontMetrics metrics = g2.getFontMetrics(UIUtils.FONT_GENERAL_UI);
217         int x2 = (getWidth() - metrics.stringWidth(UIUtils.BUTTON_TEXT_CONNECT)) /
218             2;
219         int y2 = ((getHeight() - metrics.getHeight()) / 2) + metrics.getAscent();
220         g2.setFont(UIUtils.FONT_GENERAL_UI);
221         g2.setColor(connectColors[1]);
222         g2.drawString(UIUtils.BUTTON_TEXT_CONNECT, x2, y2);
223     }
224 };
225
226 connectButton.addMouseListener(new MouseAdapter() {
227
228     @Override
229     public void mousePressed(MouseEvent e) {
230         loginEventHandler();
231     }
232
233     @Override
234     public void mouseEntered(MouseEvent e) {
235         connectColors[0] = UIUtils.COLOR_INTERACTIVE_DARKER;
236         connectColors[1] = UIUtils.OFFWHITE;
237         connectButton.repaint();
238     }
239
240     @Override
241     public void mouseExited(MouseEvent e) {
```

```
240         connectColors[0] = UIUtils.COLOR_INTERACTIVE;
241         connectColors[1] = Color.white;
242         connectButton.repaint();
243     }
244 );
245
246 connectButton.setBackground(UIUtils.COLOR_BACKGROUND);
247 connectButton.setBounds(473, 247, 250, 44);
248 connectButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
249 panel1.add(connectButton);
250 }
251
252 private void loginEventHandler() {
253     // hier Connect Code reinschreiben
254     toaster.warn("Login event");
255 }
256 }
```

Listing A.11: main/ConnectUI.java

```
1 package components;
2
3 import javax.swing.SwingUtilities;
4 import javax.swing.JFrame;
5 import javax.swing.JPanel;
6 import javax.swing.JComponent;
7
8 import java.awt.BorderLayout;
9 import java.awt.Graphics;
10 import java.awt.Color;
11 import java.awt.Dimension;
12 import javax.swing.border.EmptyBorder;
13
14 public class SpeedMeter extends JComponent {
15     private int meterWidth = 10;
16
17     private float speed = 0f;
18     private boolean breaks = false;
19     private boolean colorfade = false;
20
21     private Color background = Color.LIGHT_GRAY;
22     private Color border = Color.BLACK;
```

```
23     private Color fill = Color.GREEN;
24     private Color maxSpeedFill = Color.ORANGE;
25     private Color breakfill = Color.RED;
26
27
28     public void setSpeed(float speed) {
29         this.speed = speed;
30         repaint();
31     }
32
33     public void setBreaks(boolean breaks) {
34         this.breaks = breaks;
35         repaint();
36     }
37
38     public void setColorfade(boolean colorfade) {
39         this.colorfade = colorfade;
40     }
41
42     public void setMaxSpeedFill(Color maxSpeedFill) {
43         this.maxSpeedFill = maxSpeedFill;
44     }
45
46     public boolean isColorfade() {
47         return colorfade;
48     }
49
50     public Color getMaxSpeedFill() {
51         return maxSpeedFill;
52     }
53
54     public void toggleBreaks() {
55         setBreaks(!breaks);
56     }
57
58     public void setMeterWidth(int meterWidth) {
59         this.meterWidth = meterWidth;
60     }
61
62     public void setBackgroundFill(Color background) {
63         this.background = background;
64         repaint();
65     }
```

```
66
67     public void setBorderFill(Color border) {
68         this.border = border;
69         repaint();
70     }
71
72     public void setFill(Color fill) {
73         this.fill = fill;
74         repaint();
75     }
76
77     public void setBreakfill(Color breakfill) {
78         this.breakfill = breakfill;
79         repaint();
80     }
81
82     public Color getBackgroundFill() {
83         return background;
84     }
85
86     public Color getBorderFill() {
87         return border;
88     }
89
90     public Color getFill() {
91         return fill;
92     }
93
94     public Color getBreakfill() {
95         return breakfill;
96     }
97
98     @Override
99     protected void paintComponent(Graphics g) {
100         int w = Math.min(meterWidth, getWidth());
101         int h = getHeight();
102         int x = getWidth() / 2 - w / 2;
103         int y = 0;
104
105         g.setColor(background);
106         g.fillRect(x, y, w, h);
107
108         g.setColor(border);
```



```
148     pref.width = meterWidth;
149     return pref;
150 }
151
152 @Override
153 public void setPreferredSize(Dimension pref) {
154     super.setPreferredSize(pref);
155     setMeterWidth(pref.width);
156 }
157
158 public static void main(String[] args) {
159     SwingUtilities.invokeLater(() -> {
160         JFrame frame = new JFrame("Meter");
161         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
162
163         JPanel content = new JPanel(new BorderLayout());
164         content.setBorder(new EmptyBorder(25, 50, 25, 50));
165
166         SpeedMeter meter = new SpeedMeter();
167         meter.setPreferredSize(new Dimension(9, 100));
168         content.add(meter, BorderLayout.CENTER);
169
170         frame.setContentPane(content);
171         frame.pack();
172         frame.setLocationRelativeTo(null);
173         frame.setVisible(true);
174
175         new Thread(new SpeedTester(meter)).start();
176     });
177 }
178
179 static class SpeedTester implements Runnable {
180     final SpeedMeter meter;
181
182     SpeedTester(final SpeedMeter meter) {
183         this.meter = meter;
184     }
185
186     @Override
187     public void run() {
188         meter.setColorfade(true);
189         float speed = 0;
190         while (true) {
```

```

191     if (speed > 1) {
192         speed = -1;
193         meter.toggleBreaks();
194         try {
195             Thread.sleep(1000);
196         } catch (InterruptedException e) {
197             e.printStackTrace();
198         }
199     } else
200         speed += 0.01;
201     meter.setSpeed(speed);
202     meter.setBreaks(false);
203     try {
204         Thread.sleep(20);
205     } catch (InterruptedException e) {
206         e.printStackTrace();
207     }
208 }
209 }
210 }
211 }
```

Listing A.12: components/SpeedMeter.java

```

1 package Toaster;
2
3 import javax.swing.*;
4 import java.awt.*;
5 import java.awt.event.MouseAdapter;
6 import java.awt.event.MouseEvent;
7 import java.util.ArrayList;
8 import java.util.concurrent.atomic.AtomicInteger;
9
10 public class Toaster {
11     private static final int STARTING_Y_POS = 15;
12     private static final int SPACER_DISTANCE = 15;
13     private static final ArrayList<ToasterBody> toasterBodies = new ArrayList<>();
14     private final static AtomicInteger CURRENT_Y_OFFSET = new AtomicInteger();
15     private final JPanel panelToToastOn;
16
17     public Toaster(JPanel panelToToastOn) {
18         this.panelToToastOn = panelToToastOn;
```

```
19 }
20
21     public void error(String... messages) {
22         for (String s : messages) {
23             toast(s, new Color(181, 59, 86));
24         }
25     }
26
27     public void success(String... messages) {
28         for (String s : messages) {
29             toast(s, new Color(33, 181, 83));
30         }
31     }
32
33     public void info(String... messages) {
34         for (String s : messages) {
35             toast(s, new Color(13, 116, 181));
36         }
37     }
38
39     public void warn(String... messages) {
40         for (String s : messages) {
41             toast(s, new Color(181, 147, 10));
42         }
43     }
44
45     private void toast(String message, Color bgColor) {
46         ToasterBody toasterBody;
47
48         if (toasterBodies.isEmpty()) {
49             toasterBody = new ToasterBody(panelToToastOn, message, bgColor,
50 STARTING_Y_POS);
51             CURRENT_Y_OFFSET.set(STARTING_Y_POS + toasterBody.getHeightOfToast());
52         } else {
53             toasterBody = new ToasterBody(panelToToastOn, message, bgColor,
54 CURRENT_Y_OFFSET.get() + SPACER_DISTANCE);
55             CURRENT_Y_OFFSET.addAndGet(SPACER_DISTANCE + toasterBody.getHeightOfToast());
56         }
57
58         toasterBodies.add(toasterBody);
59
60         new Thread(() -> {
61             toasterBody.addMouseListener(new MouseAdapter() {
62                 @Override
63                 public void mouseEntered(MouseEvent e) {
64                     toasterBody.setAlpha(1.0f);
65                 }
66
67                 @Override
68                 public void mouseExited(MouseEvent e) {
69                     toasterBody.setAlpha(0.5f);
70                 }
71
72                 @Override
73                 public void mousePressed(MouseEvent e) {
74                     if (e.getButton() == MouseEvent.BUTTON1) {
75                         panelToToastOn.setAlpha(0.5f);
76                     }
77                 }
78
79                 @Override
80                 public void mouseReleased(MouseEvent e) {
81                     if (e.getButton() == MouseEvent.BUTTON1) {
82                         panelToToastOn.setAlpha(1.0f);
83                     }
84                 }
85             });
86         });
87     }
88 }
```

```
60     @Override
61     public void mousePressed(MouseEvent e) {
62         removeToast(toasterBody);
63     }
64 );
65
66     panelToToastOn.add(toasterBody, 0);
67     panelToToastOn.repaint();
68
69     try {
70         Thread.sleep(6000);
71         removeToast(toasterBody);
72     } catch (InterruptedException e) {
73         e.printStackTrace();
74     }
75 }).start();
76 }
77
78 private synchronized void removeToast(ToasterBody toasterBody) {
79     if (!toasterBody.getStopDisplaying()) {
80         toasterBody.setStopDisplaying(true);
81
82         toasterBodies.forEach(toasterBody1 -> {
83             if (toasterBodies.indexOf(toasterBody1) >=
84                 toasterBodies.indexOf(toasterBody)) {
85                 toasterBody1.setyPos(toasterBody1.getyPos() -
86                     toasterBody.getHeightOfToast() - SPACER_DISTANCE);
87             }
88         });
89
90         toasterBodies.remove(toasterBody);
91
92         CURRENT_Y_OFFSET.set(CURRENT_Y_OFFSET.get() - SPACER_DISTANCE -
93             toasterBody.getHeightOfToast());
94
95         panelToToastOn.remove(toasterBody);
96         panelToToastOn.repaint();
97     }
98 }
```

Listing A.13: Toaster/Toaster.java

```
1 package Toaster;
2
3 import Utils.UIUtils;
4
5 import javax.swing.*;
6 import java.awt.*;
7
8 class ToasterBody extends JPanel {
9     /**
10      *
11      */
12     private static final long serialVersionUID = 1L;
13     private static final int TOAST_PADDING = 15;
14     private final int toastWidth;
15     private final String message;
16     private final Color c;
17     private volatile boolean stopDisplaying;
18     private int heightOfToast, stringPosX, stringPosY, yPos;
19     private JPanel panelToToastOn;
20
21     public ToasterBody(JPanel panelToToastOn, String message, Color bgColor, int yPos) {
22         this.panelToToastOn = panelToToastOn;
23         this.message = message;
24         this.yPos = yPos;
25         this.c = bgColor;
26
27         FontMetrics metrics = getFontMetrics(UIUtils.FONT_GENERAL_UI);
28         int stringWidth = metrics.stringWidth(this.message);
29
30         toastWidth = stringWidth + (TOAST_PADDING * 2);
31         heightOfToast = metrics.getHeight() + TOAST_PADDING;
32         setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
33         setOpaque(false);
34         setBounds((panelToToastOn.getWidth() - toastWidth) / 2, (int)
35 -(Math.round(heightOfToast / 10.0) * 10), toastWidth, heightOfToast);
36
37         stringPosX = (getWidth() - stringWidth) / 2;
38         stringPosY = ((getHeight() - metrics.getHeight()) / 2) + metrics.getAscent();
39
40         new Thread(() -> {
41             while (getBounds().y < yPos) {
42                 int i1 = (yPos - getBounds().y) / 10;
43                 i1 = i1 <= 0 ? 1 : i1;
44             }
45         }).start();
46     }
47
48     public void paintComponent(Graphics g) {
49         super.paintComponent(g);
50         g.setColor(c);
51         g.drawString(message, stringPosX, stringPosY);
52     }
53
54     public void stop() {
55         stopDisplaying = true;
56     }
57 }
```

```
43         setBounds((panelToToastOn.getWidth() - toastWidth) / 2, getBounds().y +
44         i1, toastWidth, heightOfToast);
45         repaint();
46         try {
47             Thread.sleep(5);
48         } catch (Exception ignored) {
49         }
50     }).start();
51 }
52
53 @Override
54 protected void paintComponent(Graphics g) {
55     Graphics2D g2 = UIUtils.get2dGraphics(g);
56     super.paintComponent(g2);
57
58     //Background
59     g2.setColor(c);
60     g2.fillRoundRect(0, 0, getWidth(), getHeight(), UIUtils.ROUNDNESS,
61     UIUtils.ROUNDNESS);
62
63     // Font
64     g2.setFont(UIUtils.FONT_GENERAL_UI);
65     g2.setColor(Color.white);
66     g2.drawString(message, stringPosX, stringPosY);
67 }
68
69 public int getHeightOfToast() {
70     return heightOfToast;
71 }
72
73 public synchronized boolean getStopDisplaying() {
74     return stopDisplaying;
75 }
76
77 public synchronized void setStopDisplaying(boolean hasStoppedDisplaying) {
78     this.stopDisplaying = hasStoppedDisplaying;
79 }
80
81 public void setyPos(int yPos) {
82     this.yPos = yPos;
83 //     setBounds((panelToToastOn.getWidth() - toastWidth) / 2, yPos, toastWidth,
84     heightOfToast);
```

```

83
84     new Thread(() -> {
85         while (getBounds().y > yPos) {
86             int i1 = Math.abs((yPos - getBounds().y) / 10);
87             i1 = i1 <= 0 ? 1 : i1;
88             setBounds((panelToToastOn.getWidth() - toastWidth) / 2, getBounds().y -
89             i1, toastWidth, heightOfToast);
90             repaint();
91             try {
92                 Thread.sleep(5);
93             } catch (Exception ignored) {
94             }
95         }
96     }).start();
97
98     public int getyPos() {
99         return yPos;
100    }
101}

```

Listing A.14: Toaster/ToasterBody.java

```

1 package Utils;
2
3 import java.awt.*;
4 import java.util.HashMap;
5
6 public class UIUtils {
7     public static final Font FONT_GENERAL_UI = new Font("Segoe UI", Font.PLAIN, 20);
8
9     public static final Color COLOR_OUTLINE = new Color(103, 112, 120);
10    public static final Color COLOR_BACKGROUND = new Color(37, 51, 61);
11    public static final Color COLOR_INTERACTIVE = new Color(108, 216, 158);
12    public static final Color COLOR_INTERACTIVE_DARKER = new Color(87, 171, 127);
13    public static final Color OFFWHITE = new Color(229, 229, 229);
14
15    public static final String BUTTON_TEXT_CONNECT = "Connect";
16
17    public static final String Bild = "robot.png";
18
19    public static final String PLACEHOLDER_TEXT_IP = "IP Adresse";

```

```
20     public static final String PLACEHOLDER_TEXT_PORT = "PORT";
21
22     public static final int ROUNDNESS = 8;
23
24     public static Graphics2D get2dGraphics(Graphics g) {
25         Graphics2D g2 = (Graphics2D) g;
26         g2.addRenderingHints(new HashMap<RenderingHints.Key, Object>() {/*
27             *
28             */
29             private static final long serialVersionUID = 1L;
30
31             {
32                 put(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
33                 put(RenderingHints.KEY_TEXT_ANTIALIASING,
34                     RenderingHints.VALUE_TEXT_ANTIALIAS_LCD_HRGB);
35             }});
36         return g2;
37     }
38 }
```

Listing A.15: Utils/UIUtils.java

```
1 package Utils;
2
3 import javax.swing.*;
4 import java.awt.*;
5 import java.awt.geom.RoundRectangle2D;
6
7
8 public class TextFieldIP extends JTextField {
9     /**
10      *
11      */
12     private static final long serialVersionUID = 1L;
13
14     private Shape shape;
15     private Color borderColor = UIUtils.COLOR_OUTLINE;
16
17     public TextFieldIP() {
18         setOpaque(false);
19         setBackground(UIUtils.COLOR_BACKGROUND);
20         setForeground(Color.white);
```

```

21     setCaretColor(Color.white);
22     setCursor(Cursor.getPredefinedCursor(Cursor.TEXT_CURSOR));
23     setMargin(new Insets(2, 10, 2, 2));
24     setHorizontalAlignment(SwingConstants.LEFT);
25     setFont(UIUtils.FONT_GENERAL_UI);
26 }
27
28 protected void paintComponent(Graphics g) {
29     Graphics2D g2 = UIUtils.get2dGraphics(g);
30     g2.setColor(getBackground());
31     g2.fillRoundRect(0, 0, getWidth() - 1, getHeight() - 1, UIUtils.ROUNDNESS,
32     UIUtils.ROUNDNESS);
33     super.paintComponent(g2);
34 }
35
36 protected void paintBorder(Graphics g) {
37     Graphics2D g2 = UIUtils.get2dGraphics(g);
38     g2.setColor(borderColor);
39     g2.drawRoundRect(0, 0, getWidth() - 1, getHeight() - 1, UIUtils.ROUNDNESS,
40     UIUtils.ROUNDNESS);
41 }
42
43 public boolean contains(int x, int y) {
44     if (shape == null || !shape.getBounds().equals(bounds)) {
45         shape = new RoundRectangle2D.Float(0, 0, getWidth() - 1, getHeight() - 1,
46         UIUtils.ROUNDNESS, UIUtils.ROUNDNESS);
47     }
48     return shape.contains(x, y);
49 }
50
51 public void setBorderColor(Color color) {
52     borderColor = color;
53     repaint();
54 }
55 }
```

Listing A.16: Utils/TextFieldIP.java

```

1 package Utils;
2
3 import javax.swing.*;
4 import java.awt.*;
```

```
5 import java.awt.geom.RoundRectangle2D;
6
7
8 public class TextFieldPort extends JTextField {
9     /**
10      *
11      */
12     private static final long serialVersionUID = 1L;
13
14     private Shape shape;
15     private Color borderColor = UIUtils.COLOR_INTERACTIVE;
16
17     public TextFieldPort() {
18         setOpaque(false);
19         setBackground(UIUtils.COLOR_BACKGROUND);
20         setForeground(Color.white);
21         setCaretColor(Color.white);
22         setCursor(Cursor.getPredefinedCursor(Cursor.TEXT_CURSOR));
23         setMargin(new Insets(2, 10, 2, 2));
24         setHorizontalAlignment(SwingConstants.LEFT);
25        setFont(UIUtils.FONT_GENERAL_UI);
26     }
27
28     protected void paintComponent(Graphics g) {
29         Graphics2D g2 = UIUtils.get2dGraphics(g);
30         g2.setColor(getBackground());
31         g2.fillRoundRect(0, 0, getWidth() - 1, getHeight() - 1, UIUtils.ROUNDNESS,
32             UIUtils.ROUNDNESS);
33         super.paintComponent(g2);
34     }
35
36     protected void paintBorder(Graphics g) {
37         Graphics2D g2 = UIUtils.get2dGraphics(g);
38         g2.setColor(borderColor);
39         g2.drawRoundRect(0, 0, getWidth() - 1, getHeight() - 1, UIUtils.ROUNDNESS,
40             UIUtils.ROUNDNESS);
41
42     public boolean contains(int x, int y) {
43         if (shape == null || !shape.getBounds().equals(getBounds())) {
44             shape = new RoundRectangle2D.Float(0, 0, getWidth() - 1, getHeight() - 1,
45                 UIUtils.ROUNDNESS, UIUtils.ROUNDNESS);
46         }
47     }
48 }
```

```
45     return shape.contains(x, y);
46 }
47
48 public void setBorderColor(Color color) {
49     borderColor = color;
50     repaint();
51 }
52 }
```

Listing A.17: Utils/TextFieldPort.java

```
1 package zweitesFenster;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         new ZweitesFenster();
7
8     }
9
10 }
```

Listing A.18: main/Main.java

```
1 package zweitesFenster;
2
3 import java.awt.BorderLayout;
4 import java.awt.Dimension;
5 import java.awt.Font;
6 import java.awt.Image;
7 import java.awt.Toolkit;
8 import java.awt.event.MouseAdapter;
9 import java.awt.event.MouseEvent;
10 import java.awt.event.WindowAdapter;
11 import java.awt.event.WindowEvent;
12 import javax.swing.ImageIcon;
13 import javax.swing.JFrame;
14 import javax.swing.JLabel;
15 import javax.swing.JPanel;
16 import javax.swing.JProgressBar;
```

```
17 import javax.swing.JSlider;
18 import javax.swing.JTextField;
19 import javax.swing.SwingConstants;
20 import Utils.TextFieldPort;
21 import Utils.UIUtils;
22 import java.awt.Color;
23
24 public class ZweitesFenster extends JFrame{
25
26     /**
27      *
28      */
29
30     private static final long serialVersionUID = 1L;
31
32
33     ZweitesFenster(){
34
35
36         JPanel mainPanel = getMainJPanel();
37
38         addLogo(mainPanel);
39         battery(mainPanel);
40         speed(mainPanel);
41
42         getContentPane().add(mainPanel, BorderLayout.SOUTH);
43
44         pack();
45         setVisible(true);
46         toFront();
47
48         Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
49         setLocation(screenSize.width / 2 - getWidth() / 2, screenSize.height / 2 -
50         getHeight() / 2);
51
52
53
54     private JPanel getMainJPanel() {
55         setUndecorated(true);
56
57         Dimension size = new Dimension(700, 600);
58     }
```

```
59     JPanel panel1 = new JPanel();
60     panel1.setSize(size);
61     panel1.setPreferredSize(size);
62     panel1.setBackground(UIUtils.COLOR_BACKGROUND);
63     panel1.setLayout(null);
64
65     TextFieldPort titel = new TextFieldPort();
66     titel.setEnabled(false);
67     titel.setEditable(false);
68     titel.setHorizontalAlignment(SwingConstants.CENTER);
69     //titel.setBorderColor(UIUtils.COLOR_INTERACTIVE);
70     titel.setFont(new Font("Segoe UI", Font.PLAIN, 14));
71     titel.setText("OmniMove Driving Experience");
72     titel.setBounds(20, 25, 250, 45);
73     panel1.add(titel);
74     panel1.setFocusable(true);
75     panel1.requestFocus();
76
77     MouseAdapter ma = new MouseAdapter() {
78         int lastX, lastY;
79
80         @Override
81         public void mousePressed(MouseEvent e) {
82             lastX = e.getXOnScreen();
83             lastY = e.getYOnScreen();
84         }
85
86         @Override
87         public void mouseDragged(MouseEvent e) {
88             int x = e.getXOnScreen();
89             int y = e.getYOnScreen();
90             setLocation(getLocationOnScreen().x + x - lastX, getLocationOnScreen().y +
91             y - lastY);
92             lastX = x;
93             lastY = y;
94         }
95     };
96
97     panel1.addMouseListerner(ma);
98     panel1.addMouseMotionListerner(ma);
99
100    addWindowListerner(new WindowAdapter() {
101        @Override
```

```
101     public void windowClosing(WindowEvent e) {
102         System.exit(0);
103     }
104 }
105
106     return panel1;
107 }
108
109
110
111 private void addLogo(JPanel panel1) {
112     JLabel label1 = new JLabel();
113     label1.setFocusable(false);
114     label1.setVisible(true);
115     label1.setBounds(201,94,320,369);
116     Image robot = new ImageIcon(this.getClass().getResource("/robot2.png")).getImage();
117     label1.setIcon(new ImageIcon(robot));
118     panel1.add(label1);
119 }
120
121
122 private void battery(JPanel panel2) {
123     int akkustand;
124     akkustand = 50;
125
126     JProgressBar battery_number = new JProgressBar();
127     battery_number.setBackground(UIUtils.COLOR_BACKGROUND);
128     battery_number.setForeground(UIUtils.COLOR_INTERACTIVE);
129     battery_number.setEnabled(false);
130     battery_number.setLocation(531, 47);
131     battery_number.setMinimum(0);
132     battery_number.setMaximum(100);
133     battery_number.setFocusable(false);
134     battery_number.setSize(133,45);
135     battery_number.setString("Akkustand: "+akkustand+"%");
136     battery_number.setStringPainted(true);
137     battery_number.setFont(new Font("Segoe UI", Font.PLAIN, 12));
138
139     //default values
140     battery_number.setValue(50);
141
142     panel2.add(battery_number);
143 }
```

```
144 }
145
146
147 private void speed(JPanel panel3) {
148
149     int kmh = 10;
150
151     JPanel combination = new JPanel();
152     combination.setSize(222, 101);
153     combination.setVisible(true);
154     combination.setBackground(UIUtils.COLOR_BACKGROUND);
155
156     JLabel geschwindigkeit = new JLabel();
157     geschwindigkeit.setHorizontalAlignment(SwingConstants.CENTER);
158     geschwindigkeit.setText("Geschwindigkeit");
159     geschwindigkeit.setFont(new Font("Segoe UI", Font.PLAIN, 12));
160     geschwindigkeit.setFocusable(false);
161     geschwindigkeit.setEnabled(false);
162
163     JSlider speed = new JSlider(JSlider.HORIZONTAL, 0, 30, 0);
164     speed.setSize(123, 35);
165     speed.setBackground(UIUtils.COLOR_BACKGROUND);
166     speed.setEnabled(false);
167     speed.setFocusable(false);
168     speed.setLocation(292, 495);
169     speed.setMajorTickSpacing(5);
170     speed.setMinorTickSpacing(1);
171     speed.setPaintTicks(true);
172     speed.setPaintLabels(true);
173
174     speed.setMaximum(30);
175     speed.setMinimum(0);
176     speed.setValue(kmh);
177
178     JLabel geschwindigkeit2 = new JLabel();
179     geschwindigkeit2.setHorizontalAlignment(SwingConstants.CENTER);
180     geschwindigkeit2.setText(" " +kmh+ " km/h");
181     geschwindigkeit2.setVisible(true);
182     geschwindigkeit2.setFont(new Font("Segoe UI", Font.PLAIN, 12));
183     geschwindigkeit2.setFocusable(false);
184     geschwindigkeit2.setEnabled(false);
185
186     combination.setLayout(new BorderLayout());
```

```
187     combination.add(geschwindigkeit, BorderLayout.NORTH);
188     combination.add(speed, BorderLayout.CENTER);
189     combination.add(geschwindigkeit2, BorderLayout.SOUTH);
190     combination.setLocation(264,488);
191
192     panel3.add(combination);
193
194 }
195
196
197 }
```

Listing A.19: main/ZweitesFenster.java

# B Technische Zeichnungen

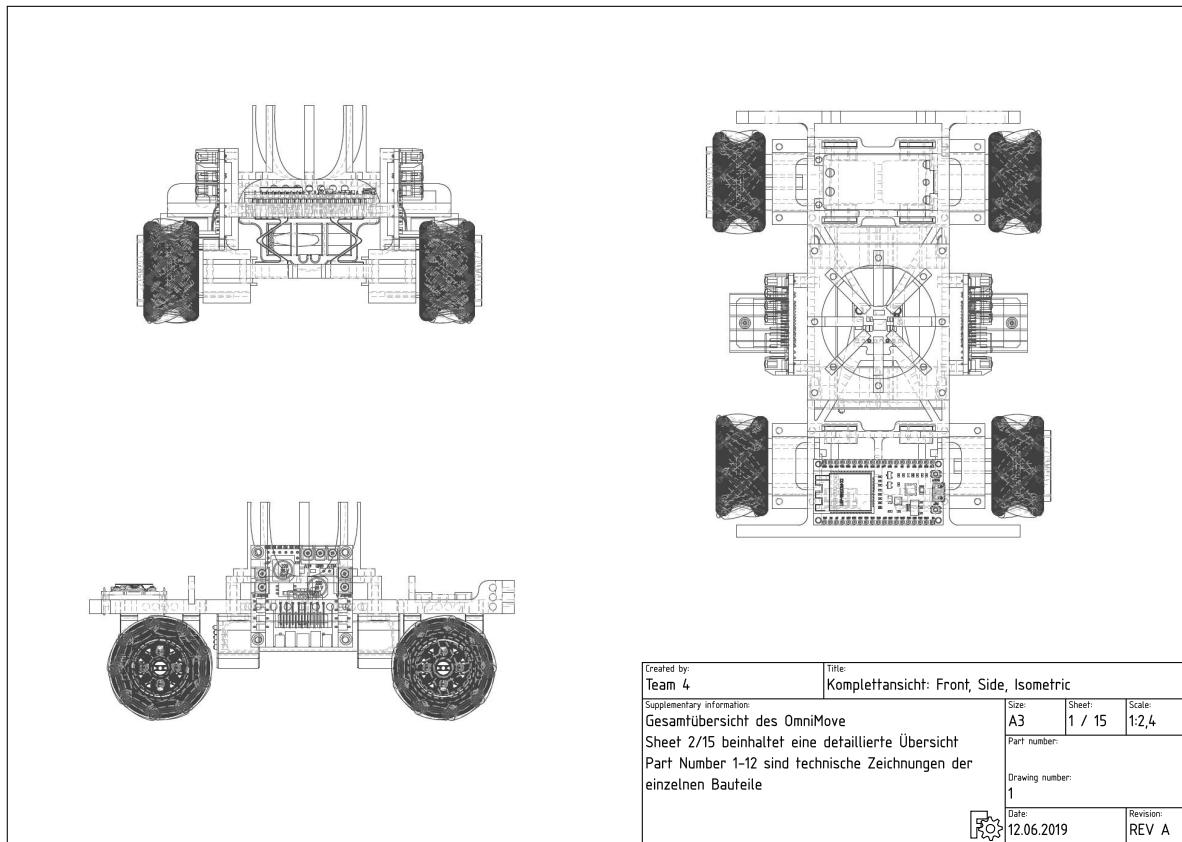


Abbildung B.1: Komplettansicht

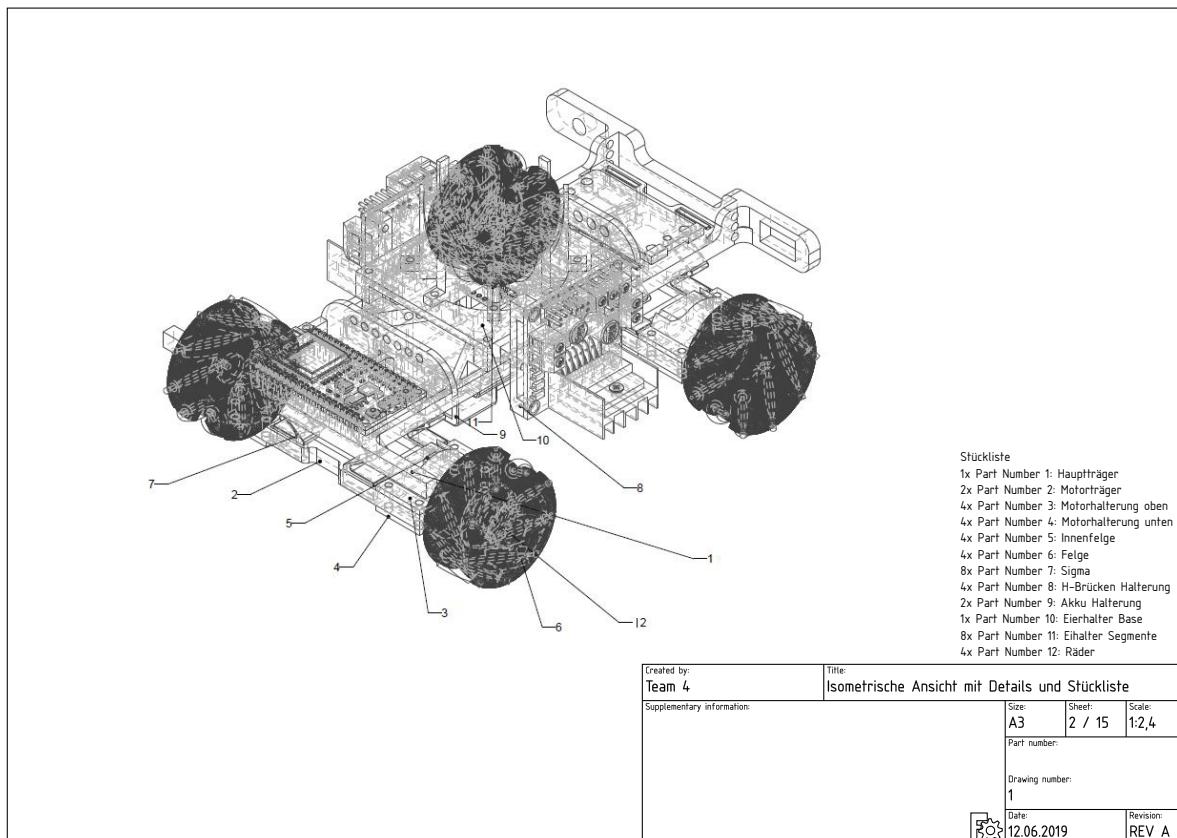


Abbildung B.2: Isometrisch und Stückliste

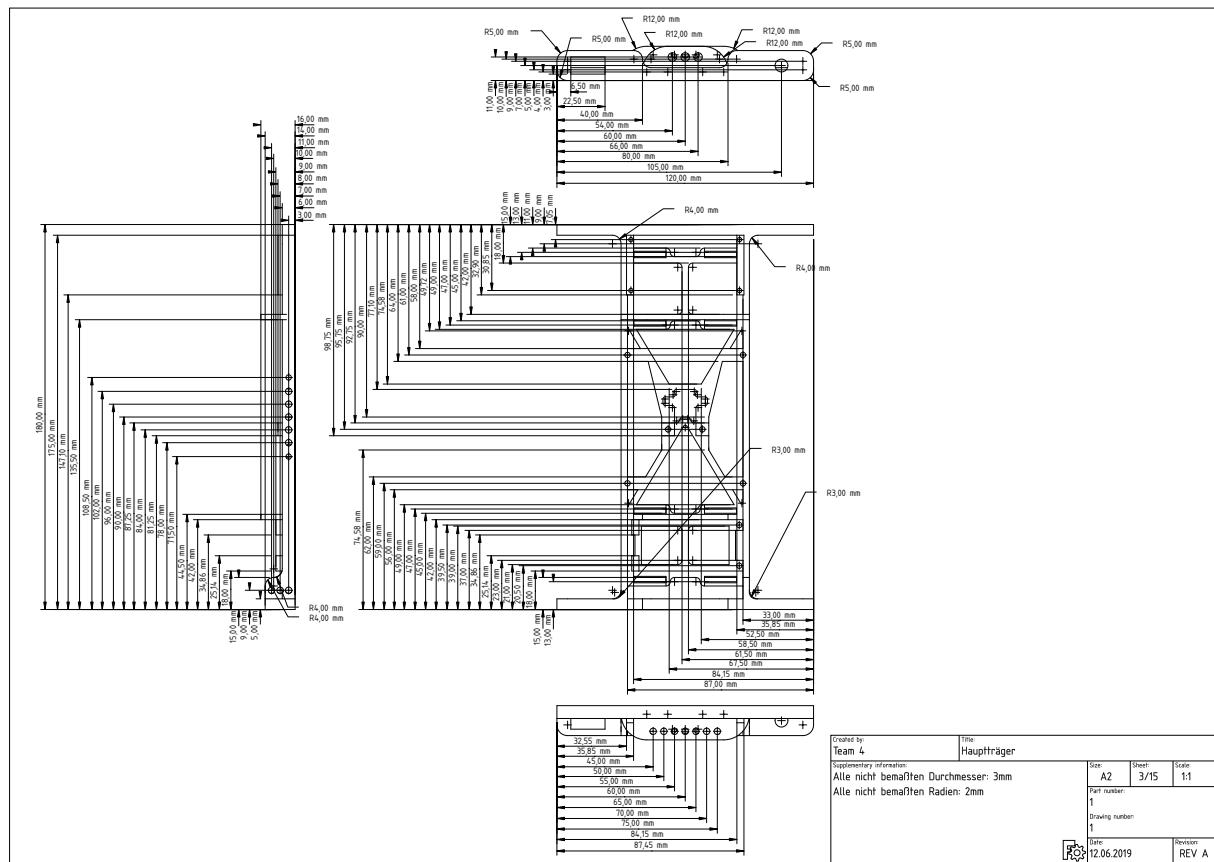


Abbildung B.3: Hauptträger

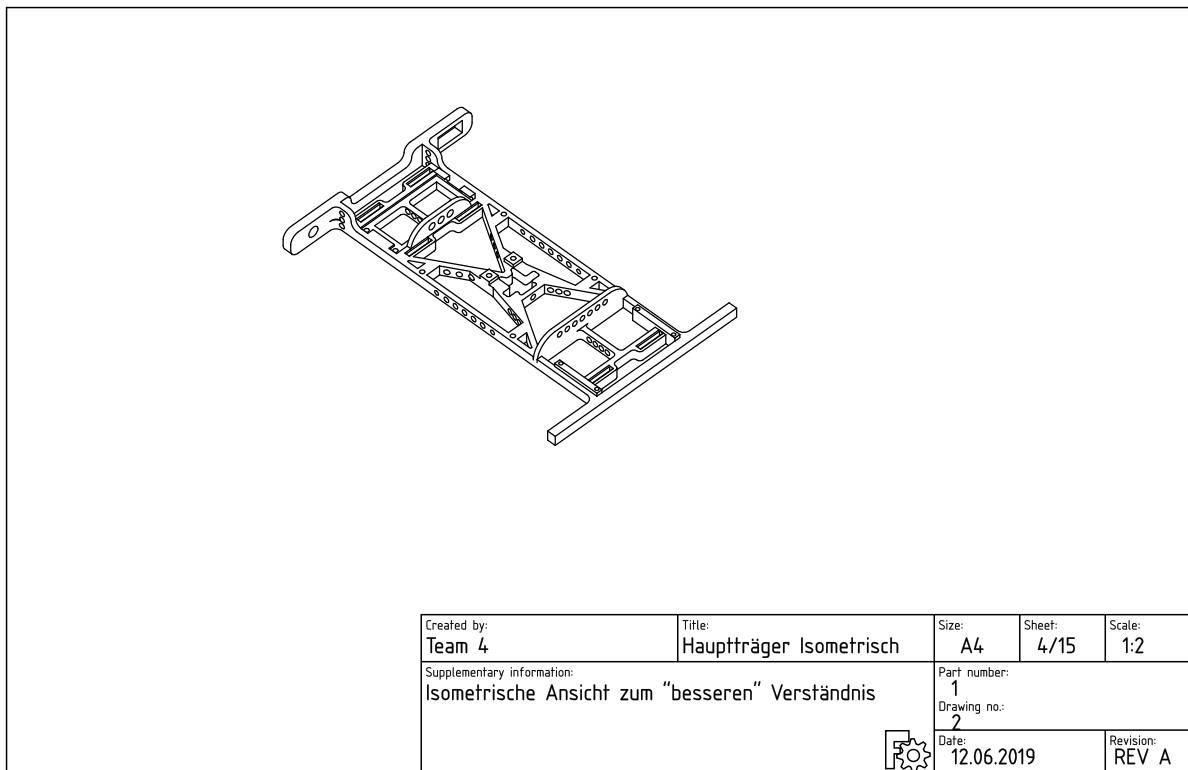


Abbildung B.4: Hauptträger Isometrisch

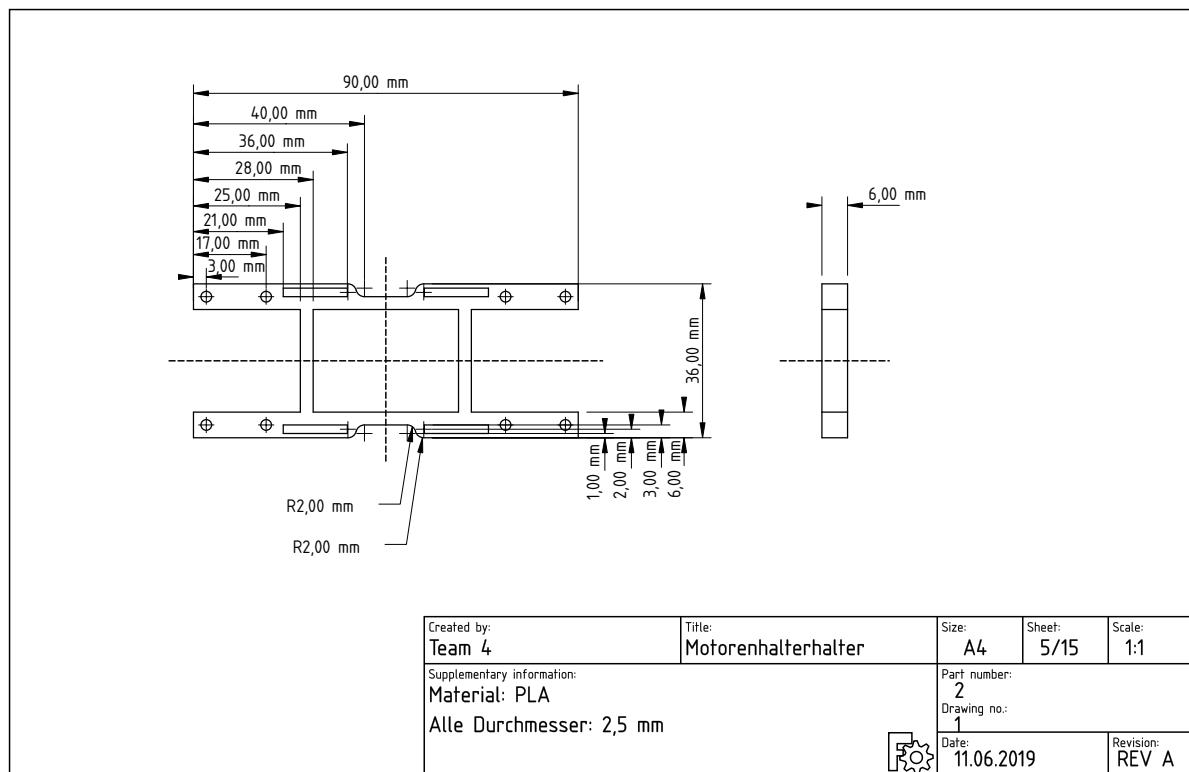


Abbildung B.5: Motorenhalterhalter

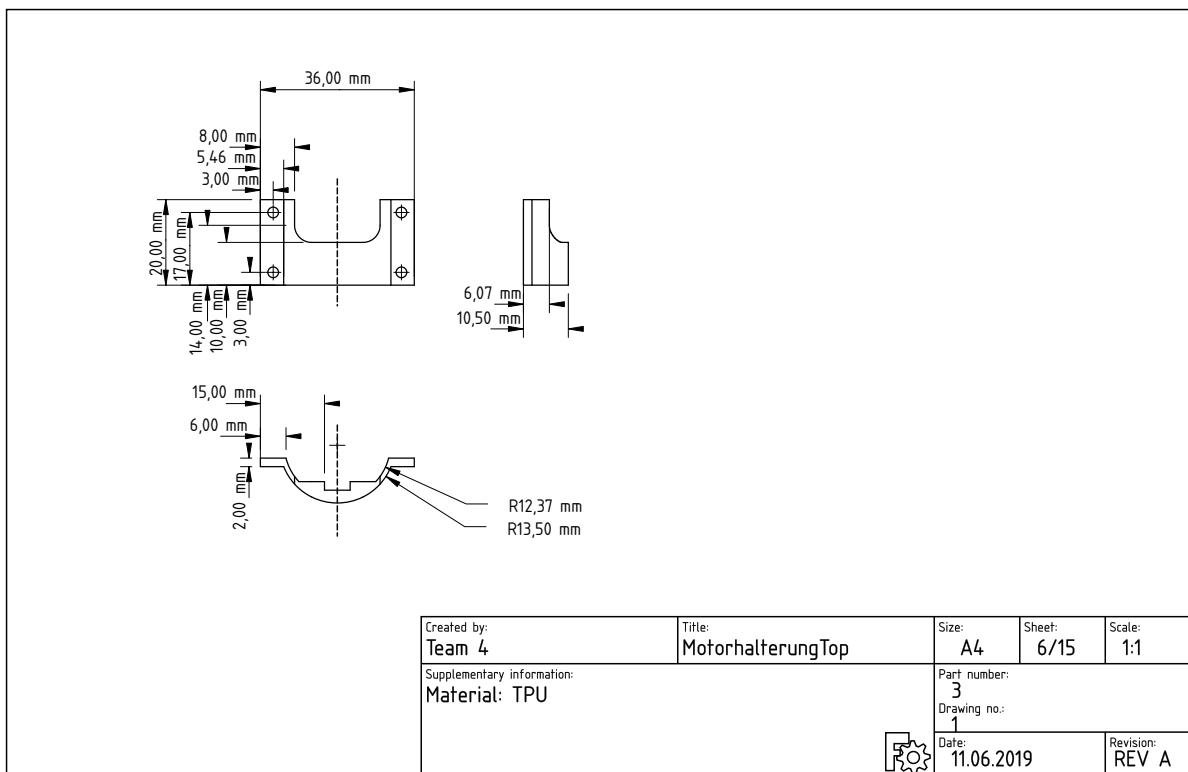


Abbildung B.6: Motorenhalterung Oben

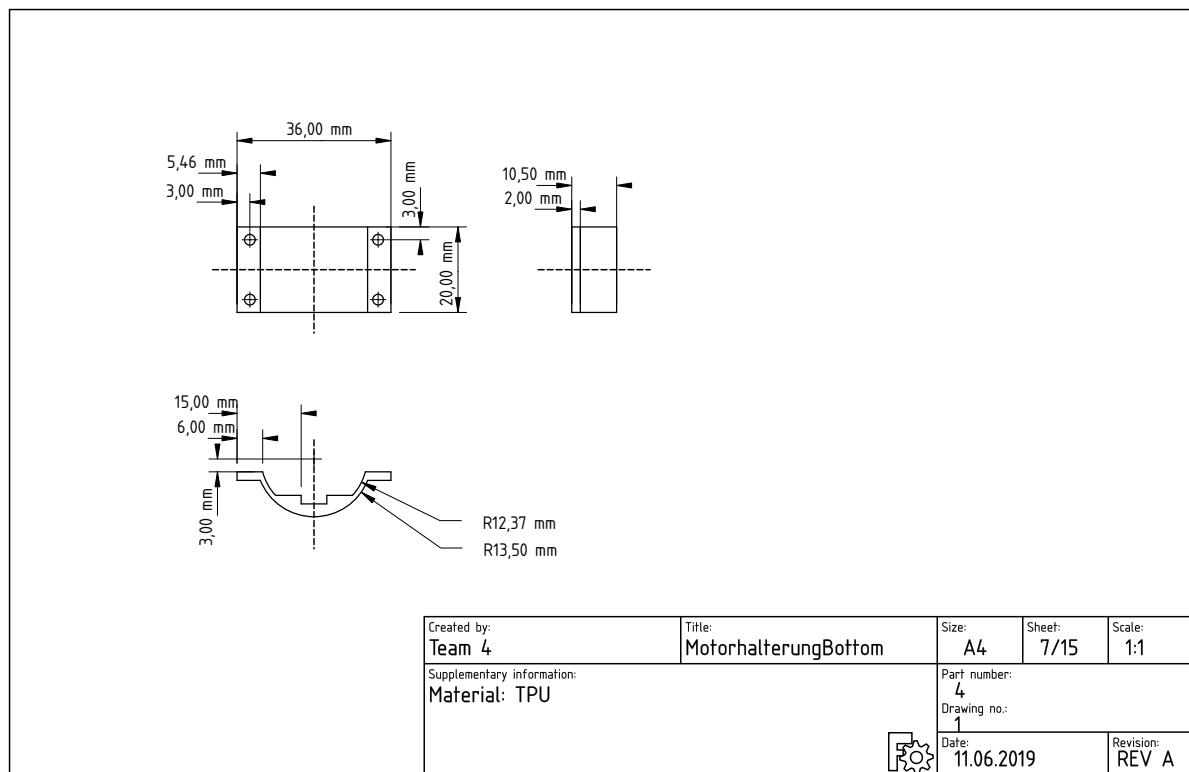


Abbildung B.7: Motorenhalterung Unten

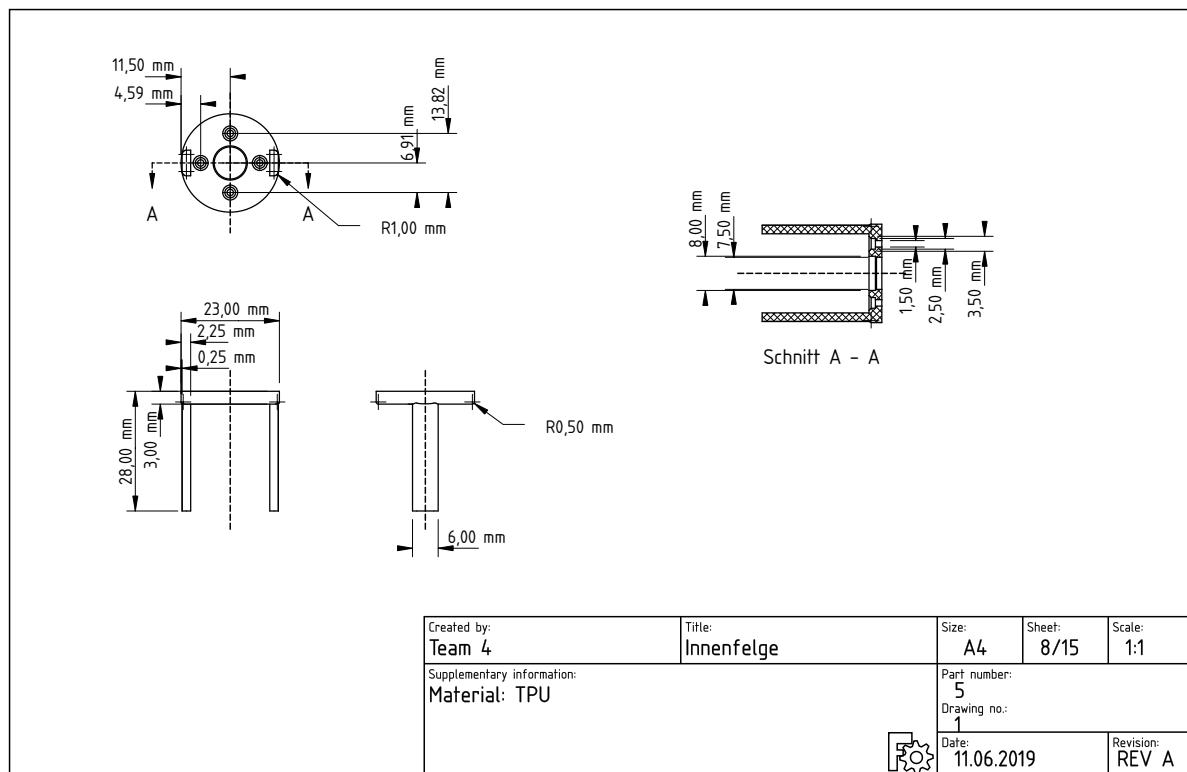


Abbildung B.8: Innenfelge

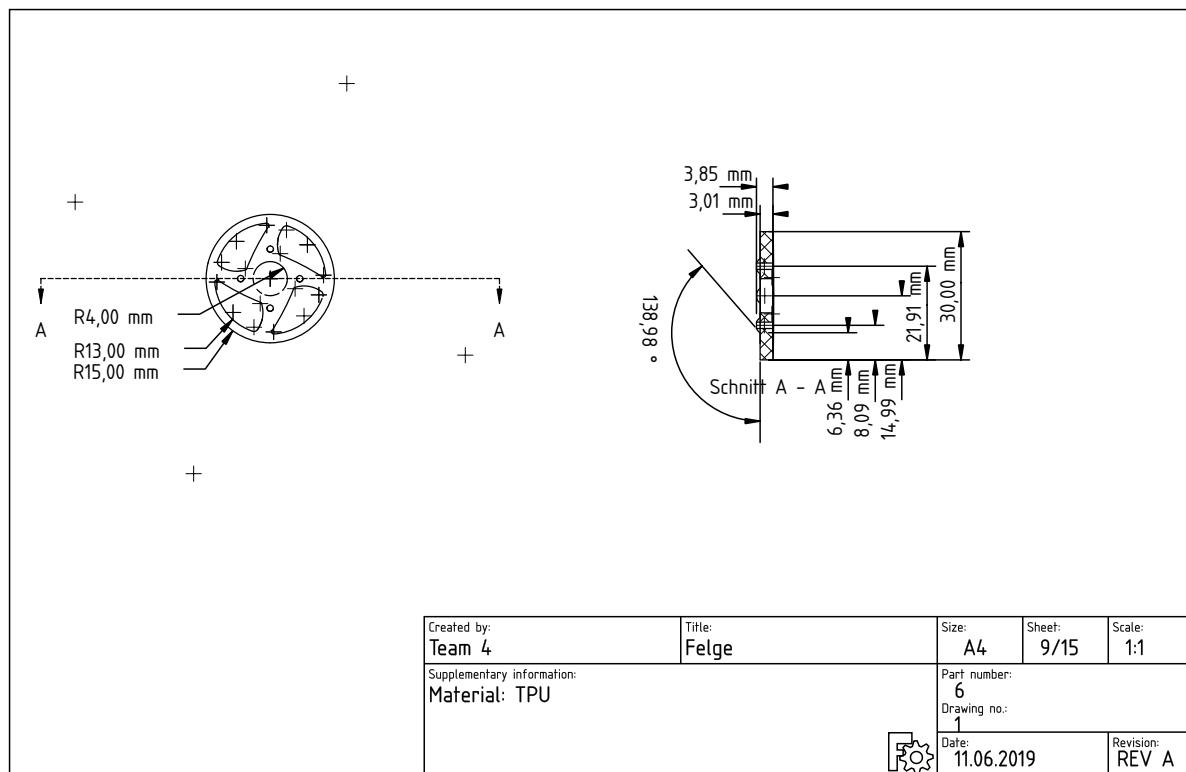


Abbildung B.9: Felge

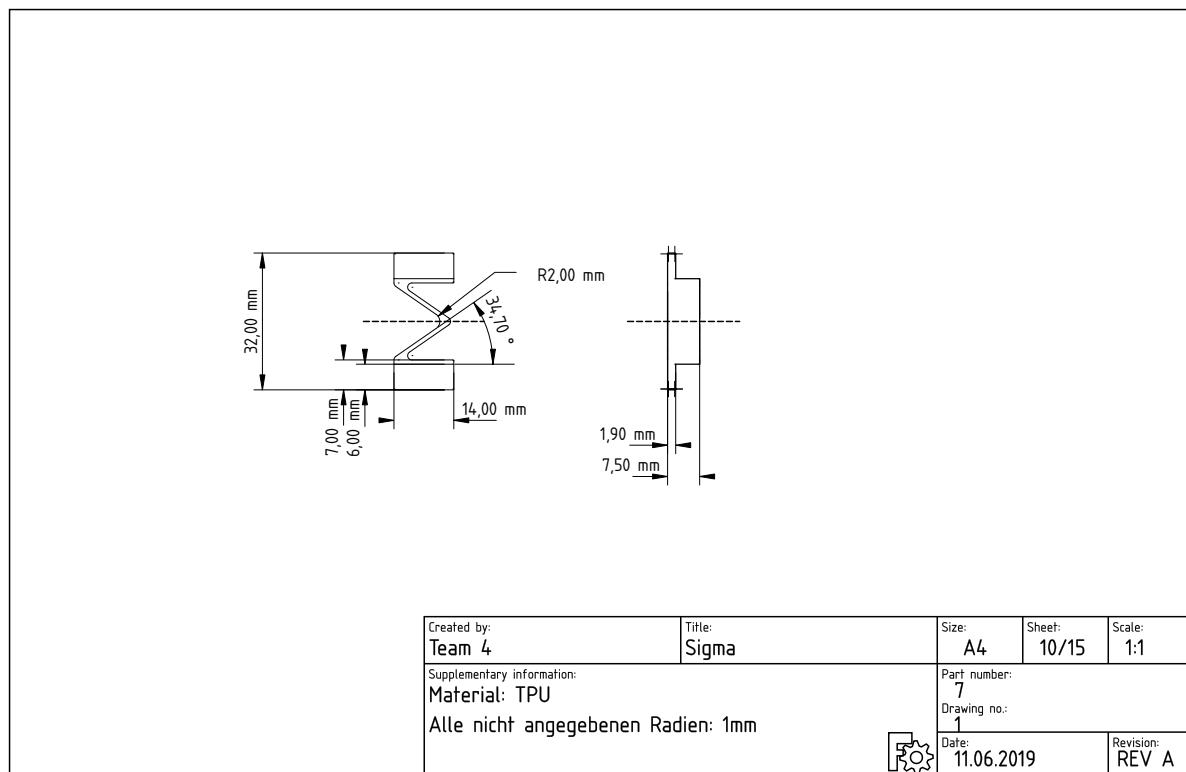


Abbildung B.10: Sigma

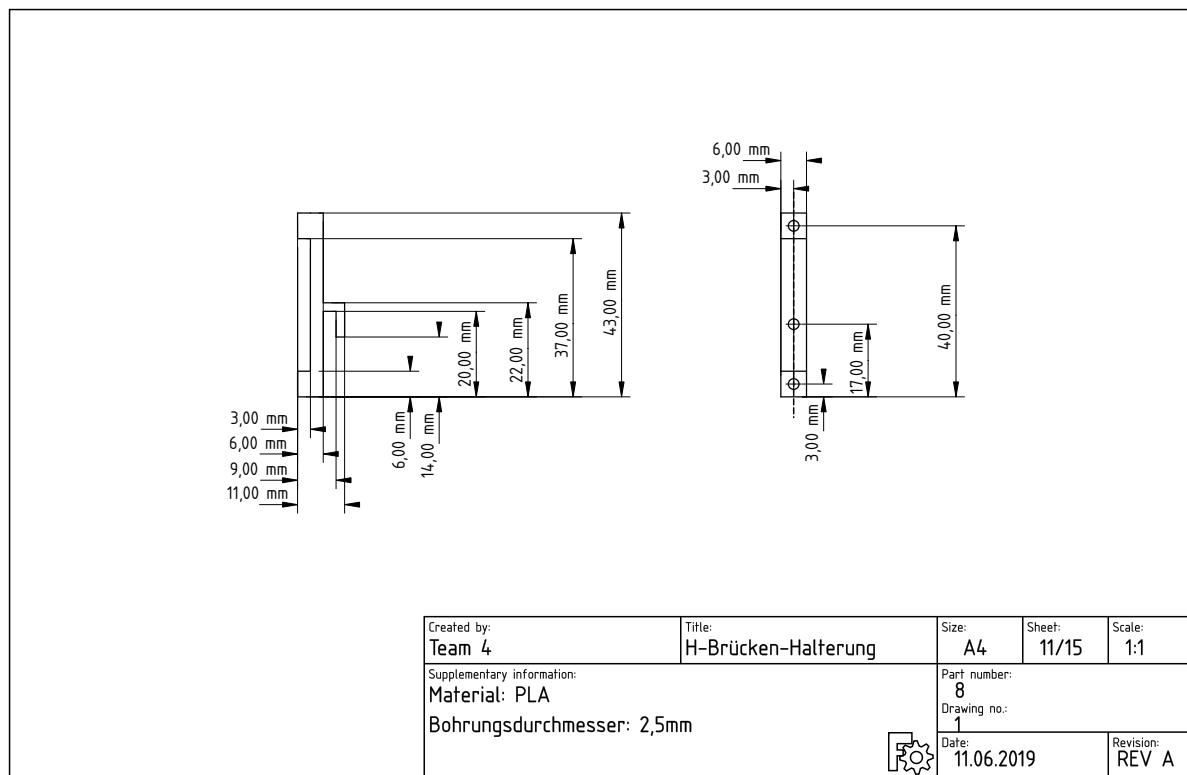


Abbildung B.11: H-Brücke-Halterung

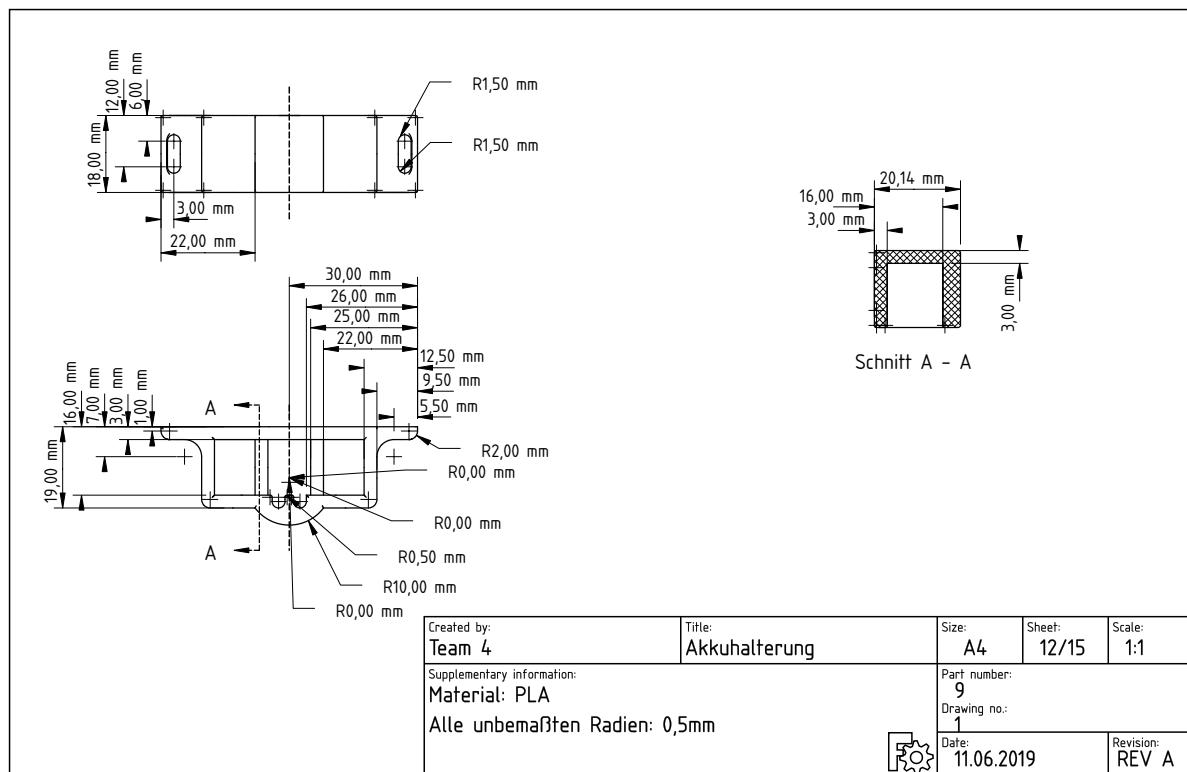


Abbildung B.12: Akkuhalterung

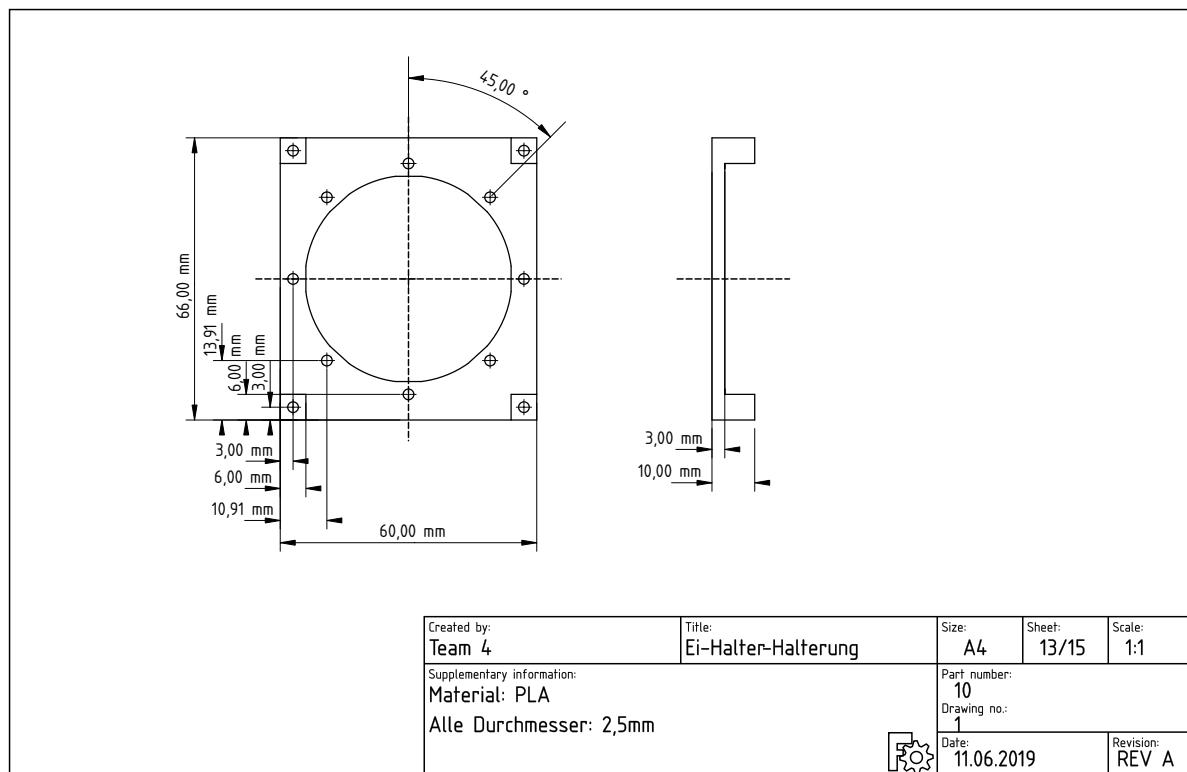


Abbildung B.13: Ei-Halterung-Halterung

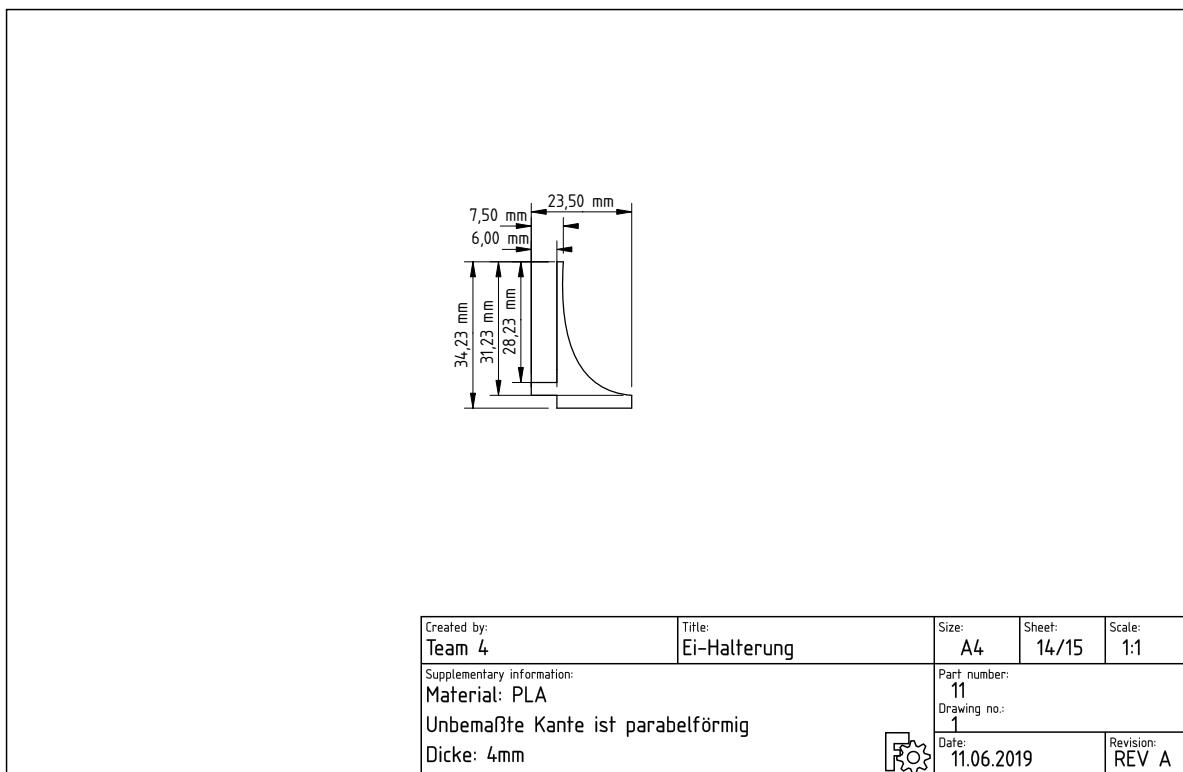


Abbildung B.14: Ei-Halterung

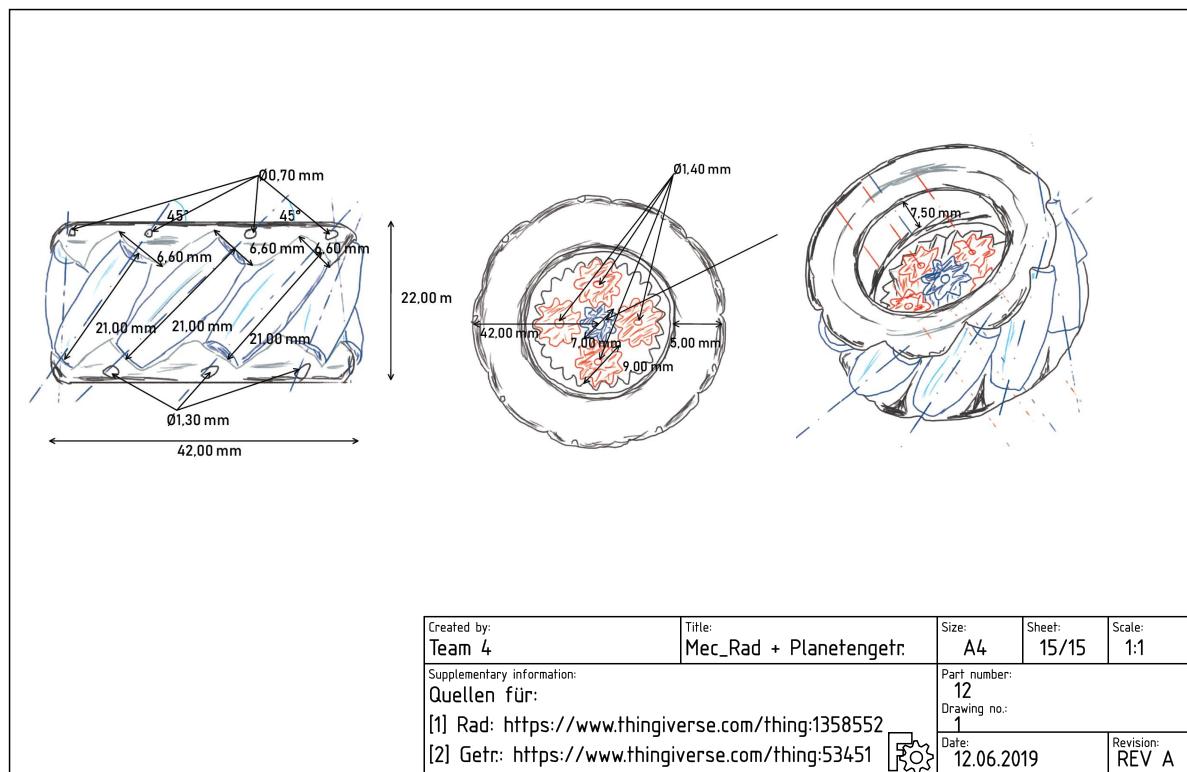


Abbildung B.15: Reifen und Planetengetriebe

# Literaturverzeichnis

- [1] EMMETT: *Gear Bearing*. <https://www.thingiverse.com/thing:53451>, Februar 2016
- [2] EMMETT: *Gear Bearing*. <https://www.thingiverse.com/thing:2666785>, Februar 2016
- [3] EMMETT: *Gear Bearing*. <https://www.thingiverse.com/thing:2277105>, Februar 2016
- [4] INNOART, Jonah: *44mm Mecanum Wheel*. <http://www.thingiverse.com/thing:1358552>, Februar 2016
- [5] ROBOTEQ (Hrsg.): *Driving Mecanum Wheels Omnidirectional Robots*. <https://www.roboteq.com/index.php/applications/applications-blog/entry/driving-mecanum-wheels-omnidirectional-robots>: RoboteQ, Oktober 2015

# Listings

A.1	communication.cpp	39
A.2	communication.h	40
A.3	movement.cpp	41
A.4	movement.h	45
A.5	util.cpp	47
A.6	util.h	49
A.7	main.cpp	49
A.8	index.html	52
A.9	code.js	54
A.10	style.css	61
A.11	main/ConnectUI.java	63
A.12	components/SpeedMeter.java	69
A.13	Toaster/Toaster.java	74
A.14	Toaster/ToasterBody.java	77
A.15	Utils/UIUtils.java	79
A.16	Utils/TextFieldIP.java	80
A.17	Utils/TextFieldPort.java	81
A.18	main/Main.java	83
A.19	main/ZweitesFenster.java	83

# Abbildungsverzeichnis

3.1	Skizze Seggway	7
3.2	Skizze Weggchair	9
3.3	Skizze TEGGLA	10
3.4	Morphologischer Kasten	11
3.5	Graphische Oberfläche für Git (GitKraken)	13
3.6	CAD Programm “Creo Parametrics”	14
4.1	Übersicht der TEGGLA-Komponenten	16
4.2	Schaltplan (gezeichnet mit Fritzing)	18
4.3	Freiheitsgrade von Mecanum [5]	19
4.4	Formeln für die individuellen Motoren [5]	20
4.6	OpenSCAN zum Erstellen von Planetengetrieben	21
4.7	Im Rad integrierte Getriebe	22
4.8	Iterationen der Getriebe	23
4.9	Mehrstufige Getriebe	24
4.10	Finale Getriebebaugruppe	25
4.11	Java Fenster 1: Verbindungsaufbau	27
4.12	Java Fenster 2: Steuerung	28
4.13	Bildschirmfoto Weboberfläche	29
5.1	Blockschaltbild Regler	33
B.1	Komplettansicht	89
B.2	Isometrisch und Stückliste	90
B.3	Hauptträger	91
B.4	Hauptträger Isometrisch	92
B.5	Motorenhalterhalter	93
B.6	Motorenhalterung Oben	94
B.7	Motorenhalterung Unten	95
B.8	Innenfelge	96

B.9 Felge . . . . .	97
B.10 Sigma . . . . .	98
B.11 H-Brücken-Halterung . . . . .	99
B.12 Akkuhalterung . . . . .	100
B.13 Ei-Halterung-Halterung . . . . .	101
B.14 Ei-Halterung . . . . .	102
B.15 Reifen und Planetengetriebe . . . . .	103

# Tabellenverzeichnis

2.1	Anforderungsliste	2
3.1	Stückliste Seggway	8
3.2	Kostenübersicht	12
4.1	Stückliste der TEGGLA-Komponenten	17
4.2	Aufzählung benötigter Pins	25
4.3	Vorteile des ESP-32	26
4.4	Binäres Protokoll	30