

# DIPLOMARBEIT

Gesamtprojekt

## **RoboGlove - Bionische Hand**

**3D-Druck, Mechanik, User-Interface Programmierung**

Amir Al-Maytah      5BHEL      Betreuer: Prof. Dipl.-Ing. Christoph Diemberger

**Mikrokontroller-Programmierung, Testmanagement, Gesamtintegration**

Fabian Schweitzer      5BHEL      Betreuer: Prof. Dipl.-Ing. Christoph Diemberger

**Hardwareentwicklung, PCB-Design, Projektleitung**

Ladislaus Szabo      5BHEL      Betreuer: Prof. Dipl.-Ing. Christoph Diemberger

---

Ausgeführt im Schuljahr 2023/2024

Abgabevermerk:

Datum: 13.2.2024

übernommen von:



## Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen Hilfsmittel als die angegebenen benutzt habe. Die Stellen, die anderen Werken (gilt ebenso für Werke aus elektronischen Datenbanken oder aus dem Internet) wörtlich oder sinngemäß entnommen sind, habe ich unter Angabe der Quelle und Einhaltung der Regeln wissenschaftlichen Zitierens kenntlich gemacht. Diese Versicherung umfasst auch in der Arbeit verwendete bildliche Darstellungen, Tabellen, Skizzen und Zeichnungen. Für die Erstellung der Arbeit habe ich auch folgende Hilfsmittel generativer KI-Tools (z. B. ChatGPT, Grammarly Go, Midjourney) zu folgendem Zweck verwendet: [Bitte hier Einsatzgebiet anführen.]. Die verwendeten Hilfsmittel wurden vollständig und wahrheitsgetreu inkl. Produktversion und Prompt ausgewiesen.

---

Amir Al-Maytah

---

Fabian Schweitzer

---

Ladislaus Szabo



## **Danksagung**

Wir möchten uns herzlich bei unserem Betreuer Dipl.-Ing. Christoph Diemberger bedanken, der uns bei diesem Projekt grundlegend unterstützt und motiviert hat.

Des Weiteren gilt unser Dank auch Fachlehrer Robert Offner und allen anderen Lehrpersonen, die uns in der Werkstatt betreut und geholfen haben.

Wir danken allen, die uns im Rahmen dieses Projekts zur Seite standen.



## **DIPLOMA THESIS**

### **DOCUMENTATION**

<b>Authors</b>	Amir Al-Maytah, Fabian Schweitzer, Ladislaus Szabo
<b>From</b>	5BHEL 2023/2024
<b>Academic year</b>	
<b>Topic</b>	RoboGlove - Bionische Hand
<b>CO-operation partners</b>	

<b>Assignment of tasks</b>	As part of a feasibility study, a robotic hand is to be controlled by means of a glove. For this purpose, various sensors must be tested for accuracy and the evaluated data then transmitted wirelessly to the robotic hand. The positions of the fingers are to be displayed in a user interface.
----------------------------	---

<b>Realization</b>	The movements of the human hand are recorded with Flex sensors on the glove. The required data is then transmitted to the robotic hand via Bluetooth. A website will serve as the user interface.
--------------------	---

<b>Results</b>	The hand movements can be correctly evaluated and transmitted. The user wears the glove and grasps an object. The servo motors interpret the received data by means of the PCB and enable the robot hand to also grasp the same object. When the hand is opened, the robot hand must also move back to its starting position.
----------------	---



**HTBLVA Wien 20**  
College of  
Electronic and Technical Information  
Technology

**Diploma  
Exam**

**Illustrative graph, photo  
(with explanation)**



**Die Schule der Technik**

Final construction of the System

**Participation in competitions,  
Awards**

**Accessibility of  
Diploma Thesis**

Department administration

**Approval  
(Date/Signature)**

Examiner

Head of College/Department

**DIPLOMARBEIT**  
**DOKUMENTATION**

<b>Namen der Verfasser/innen</b>	Amir Al-Maytah, Fabian Schweitzer, Ladislaus Szabo
<b>Jahrgang Schuljahr</b>	5BHEL 2023/2024
<b>Thema der Diplomarbeit</b>	RoboGlove - Bionische Hand
<b>Kooperationspartner</b>	

<b>Aufgabenstellung</b>	Im Rahmen eines Diplomprojekts soll eine Roboterhand mittels eines Handschuhs gesteuert werden. Dazu müssen verschiedene Sensoren auf Genauigkeit getestet und anschließend die ausgewerteten Daten kabellos an die Roboterhand übertragen werden. Die Bewegungen der Finger sollen mit Motoren nachgebildet werden und die Stellungen dieser soll im User-Interface dargestellt werden.
-------------------------	--

<b>Realisierung</b>	Die Bewegungen der menschlichen Hand werden mit Flexsensoren am Handschuh erfasst. Die benötigten Daten werden anschließend über ein drahtloses Protokoll an die Roboterhand übertragen. Die Bewegungen der Roboterfinger werden mit Servomotoren realisiert. Als User-Interface soll eine Application dienen.
---------------------	--

<b>Ergebnisse</b>	Die Handbewegungen können korrekt ausgewertet und übertragen werden. Der Benutzer trägt den Handschuh und greift ein Objekt. Die Servomotoren interpretieren mittels des PCBs die empfangenen Daten und ermöglichen es der Roboterhand ebenfalls das gleiche Objekt zu greifen. Beim Öffnen der Hand muss sich die Roboterhand in ihre Ausgangsstellung zurückbewegen.
-------------------	--



Die Schule der Technik

**HTBLVA Wien 20**  
**Höhere Technische Lehranstalt für**  
Elektronik und Technische Informatik

**Reife- und**  
**Diplomprüfung**

**Typische Grafik, Foto etc.**  
**(mit Erläuterung)**



**Die Schule der Technik**

Vollständiger Aufbau des Systems

**Teilnahme an Wettbewerben,**  
**Auszeichnungen**

**Möglichkeiten der**  
**Einsichtnahme in die Arbeit**

Abteilungsadministration

**Approbation**  
**(Datum/Unterschrift)**

Prüfer/Prüferin

Direktor/Direktorin  
Abteilungsvorstand/Abteilungsvorständin

# **Inhaltsverzeichnis**

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
1.1 Ausgangslage und grundlegende Motivation . . . . .	1
1.2 Themenerläuterung . . . . .	1
1.3 Grundgliederung der Arbeit . . . . .	2
1.4 Untersuchungsanliegen der individuellen Themenstellungen . . . . .	2
<b>2 Lastenheft</b>	<b>3</b>
<b>3 Vorwissenschaftlicher Theorieteil</b>	<b>7</b>
3.1 Platzhalter Thema Amir . . . . .	7
3.2 Platzhalter Thema Fabian . . . . .	7
3.3 Sicherheitsaspekte der Mensch-Roboter-Kollaboration . . . . .	7
<b>4 Theoretische Grundlagen des Diplomprojekts</b>	<b>8</b>
4.1 Flexsensor . . . . .	8
4.2 Servomotor . . . . .	9
4.3 ESP-NOW <b>Fabian</b> . . . . .	9
4.3.1 OSI-Modell . . . . .	11
4.3.1.1 Physical Layer . . . . .	12
4.3.1.2 Data Link Layer . . . . .	12
4.3.1.3 Network Layer . . . . .	12
4.3.1.4 Transport Layer . . . . .	12
4.3.1.5 Session Layer . . . . .	13
4.3.1.6 Presentation Layer . . . . .	13
4.3.1.7 Application Layer . . . . .	13
4.3.1.8 ESP-NOW Schichten . . . . .	13
4.3.1.9 IEEE 802.11b/g/n Standard . . . . .	13
4.4 OPV Technologien <b>Laci</b> . . . . .	14
4.4.1 Single-Supply . . . . .	14
4.4.2 Dual-Supply . . . . .	15
4.4.3 Rail-to-Rail Technologie . . . . .	15
4.5 Analog-Digital-Wandler . . . . .	16
4.6 Multiplexer . . . . .	16
4.7 PWM-Signale . . . . .	17
4.8 I2C Funktionsweise . . . . .	17
4.9 UART Funktionsweise . . . . .	18
4.10 WiFi . . . . .	19
<b>5 Grundlegende Systemkonzepte</b>	<b>21</b>
5.1 Gesamtsystemkonzept . . . . .	21
5.1.1 Systembeschreibung . . . . .	21
5.2 Eingabesubsystem . . . . .	22
5.3 Ausgabesubsystem . . . . .	23

<b>6 Mechanische Realisierung</b>	<b>24</b>
6.1 Handschuh <b>Amir</b> . . . . .	24
6.1.1 Platzhalter für diverse Unterpunkte von <b>Amir</b> . . . . .	24
6.2 Roboterhand <b>Amir</b> . . . . .	25
6.2.1 Der Grundgedanke . . . . .	25
6.2.1.1 Fertigung und CAD . . . . .	25
6.2.2 Versuchsaufbauten . . . . .	26
6.2.2.1 Aufbau . . . . .	26
6.2.2.2 Versuchsresultat . . . . .	26
6.2.3 Erste Testhand . . . . .	27
6.2.3.1 Konzept Einführung . . . . .	27
6.2.3.2 Realisierung und Zusammenbau . . . . .	27
6.2.3.3 Fazit und Ergebnisse . . . . .	33
6.2.4 Zweite Testhand (Inmoov) . . . . .	35
6.2.4.1 Einführung . . . . .	35
6.2.4.2 Mechanismus . . . . .	35
6.2.4.3 Fazit und Ergebnisse . . . . .	38
6.2.5 Dritte Testhand (Projekt-Silikonhand) . . . . .	40
6.2.5.1 Das Konzept . . . . .	40
6.2.5.2 Gussform . . . . .	41
6.2.5.3 Gelenke . . . . .	44
6.2.5.4 Entwurf einer neuen Gussform . . . . .	46
6.2.5.5 Gussform und Gussversuch . . . . .	49
6.2.6 Vierte Hand (Hybrid-Konzept) . . . . .	54
6.2.6.1 Erfahrung und Konzepteklärung . . . . .	54
6.2.6.2 Erster Entwurf . . . . .	55
6.2.6.3 Verbesserungen – zweiter Entwurf . . . . .	57
6.2.6.4 Realisierung und Aufbau . . . . .	58
6.2.6.5 Versuche . . . . .	58
<b>7 Hardware Realisierung</b>	<b>59</b>
7.1 Eingabesubsystem . . . . .	59
7.1.1 Grundlegende Voraussetzungen <b>Laci</b> . . . . .	59
7.1.2 Überlegungen, Simulationen und Berechnungen <b>Laci</b> . . . . .	59
7.1.2.1 Bewegungserfassung der Fingerbeugung . . . . .	59
7.1.2.2 Auslesen der Sensoren . . . . .	59
7.1.2.3 Berechnung des Tiefpassfilters . . . . .	61
7.1.2.4 Berechnung der OPV Verstärkung und Dimensionierung des Shuntwiderstands . . . . .	61
7.1.2.5 Umwandlung der Differenzwerte in ein geeignetes Format . . . . .	61
7.1.2.6 Vervielfachung der Schaltung für alle Flexsensoren . . . . .	62
7.1.2.7 Bewegungserfassung der Handgelenksdrehung . . . . .	63
7.1.2.8 Mikrokontroller . . . . .	63
7.1.2.9 <b>Akkuversorgung</b> . . . . .	64
7.1.3 Versuchsaufbauten und Messungen <b>Laci</b> . . . . .	65
7.1.3.1 Messschaltung der Flexsensoren . . . . .	65

7.1.3.2	Messergebnisse . . . . .	66
7.1.4	Schaltungsdesign <i>Laci</i> . . . . .	70
7.1.4.1	Externe Anschlüsse . . . . .	70
7.1.4.2	Mikrokontroller . . . . .	71
7.1.4.3	Mikrokontroller Buttons . . . . .	72
7.1.4.4	Status LEDs . . . . .	72
7.1.4.5	Multiplexer . . . . .	73
7.1.4.6	Operationsverstärker . . . . .	73
7.1.4.7	Analog-Digital-Wandler . . . . .	74
7.1.4.8	Gyroskop-Sensor . . . . .	75
7.1.5	Platinendesign <i>Laci</i> . . . . .	75
7.1.5.1	Erste Testplatine . . . . .	75
7.1.5.2	Zweite Version der Platine . . . . .	77
7.2	Roboterhand . . . . .	82
7.2.1	Grundlegende Voraussetzungen . . . . .	82
7.2.2	Überlegungen, Simulationen und Berechnungen <i>Laci</i> . . . . .	82
7.2.2.1	Aktorik zur Fingerbewegung . . . . .	82
7.2.2.2	Datenempfang . . . . .	82
7.2.2.3	Ansteuerung der Motoren . . . . .	82
7.2.2.4	Positions -und Kraftmessung der Roboterfinger . . . . .	83
7.2.3	Versuchsaufbauten und Messungen <i>Laci</i> . . . . .	86
7.2.3.1	Messschaltung der Griffkraftkontrolle . . . . .	86
7.2.4	Schaltungsdesign <i>Laci</i> . . . . .	90
7.2.4.1	Externe Anschlüsse . . . . .	90
7.2.4.2	Serial-UART-Bridge . . . . .	92
7.2.4.3	Mikrokontroller . . . . .	93
7.2.4.4	Mikrokontroller Buttons . . . . .	94
7.2.4.5	Versorgung . . . . .	95
7.2.4.6	PWM-Controller . . . . .	95
7.2.4.7	Servoanschlüsse . . . . .	96
7.2.4.8	Strommessung zur Griffkraftkontrolle . . . . .	97
7.2.4.9	Status LEDs . . . . .	99
7.2.5	Platinendesign <i>Laci</i> . . . . .	100
7.2.5.1	Erste Testplatine . . . . .	100
7.2.5.2	Zweite Version der Platine . . . . .	103
<b>8</b>	<b>Software Realisierung</b> . . . . .	<b>107</b>
8.1	Handschuh . . . . .	107
8.1.1	Konzepte und Überlegungen <i>Fabian</i> . . . . .	107
8.1.1.1	Datenübertragung . . . . .	107
8.1.1.2	Programmiersprache/Entwicklungsumgebung . . . . .	107
8.1.1.3	Benutzerfreundlichkeit . . . . .	108
8.1.1.4	Testen . . . . .	108
8.1.1.5	Allgemeines Konzept . . . . .	109
8.1.1.6	Minimaler & maximaler Widerstandswert . . . . .	109
8.1.1.7	Toleranz bei Werten . . . . .	110

8.1.1.8	Verworfene Ideen . . . . .	110
8.1.2	Realisierung und Gliederung <b>Fabian</b> . . . . .	111
8.1.2.1	Realisierung . . . . .	111
8.1.2.2	Gliederung . . . . .	112
8.1.2.3	Wichtige Codezeilen . . . . .	113
8.2	Roboterhand <b>Amir</b> . . . . .	117
8.2.1	Grundlegende Voraussetzungen . . . . .	117
8.2.2	Realisierung: Überblick . . . . .	117
8.2.2.1	Programmfluss . . . . .	117
8.2.2.2	Effiziente Gliederung essentieller Systemkomponenten . . . . .	120
8.2.3	Kommunikation - Protokolle . . . . .	121
8.2.3.1	Drahtlose Kommunikation . . . . .	121
8.2.3.2	Serielle Kommunikation . . . . .	122
8.2.4	Individuelle Komponenten . . . . .	123
8.2.4.1	Parser und Interpretation: drahtloser Datenströme . . . . .	123
8.2.4.2	Parser und Interpretation: serieller Datenströme . . . . .	124
8.2.4.3	proaktiver Datenkorrekturalgorithmus . . . . .	124
8.2.4.4	Datenaufbereitung und Anweisung für Servoaktoren . . . . .	126
8.2.5	Fremdkomponenten . . . . .	127
8.2.5.1	MCP3X21 . . . . .	127
8.2.5.2	HCPA9685 . . . . .	127
8.2.5.3	ESP-NOW und Wifi.h . . . . .	127
8.3	User Interface . . . . .	128
8.3.1	Grundlegende Voraussetzungen . . . . .	128
8.3.2	Entwicklungsumgebung <b>Laci</b> . . . . .	128
8.3.3	Konzepte und Überlegungen <b>Laci</b> . . . . .	128
8.3.3.1	Allgemeines Konzept . . . . .	128
8.3.4	Realisierung des Frontends <b>Laci</b> . . . . .	130
8.3.4.1	Erläuterung der Grafikoberfläche . . . . .	130
8.3.5	Realisierung des Backends <b>Laci</b> . . . . .	132
<b>9</b>	<b>Tests und Messungen</b>	<b>133</b>
9.1	Widerstandsmessung der Flexsensoren <b>Fabian</b> . . . . .	133
<b>10</b>	<b>Ergebnisse und Erkenntnisse</b>	<b>137</b>
10.1	Mechanik <b>Amir</b> . . . . .	137
10.2	Hardware <b>Laci</b> . . . . .	137
10.3	Software . . . . .	138
10.3.1	Eingabesubsystem <b>Fabian</b> . . . . .	138
10.3.2	Ausgabesubsystem <b>Amir</b> . . . . .	139
10.4	Gesamtintegration <b>Fabian</b> . . . . .	139
<b>11</b>	<b>Ausblick</b>	<b>142</b>
<b>12</b>	<b>Anhang</b>	<b>144</b>
<b>13</b>	<b>Abkürzungsverzeichnis</b>	<b>144</b>

## INHALTSVERZEICHNIS

---

<b>14 Projektmanagement</b>	<b>145</b>
14.1 Projektstrukturplan . . . . .	145
14.2 Milestoneplan . . . . .	145
14.3 Gantt-Diagramm . . . . .	145
14.4 Arbeitstunden . . . . .	146
<b>15 Programme, Installationen und Plugins</b>	<b>157</b>
15.1 Fusion 360 . . . . .	157
15.2 UltiMaker Cura . . . . .	157
15.3 KiCad . . . . .	157
15.4 Arduino IDE . . . . .	157
15.5 QT Framework . . . . .	157
<b>16 Fertigungsunterlagen</b>	<b>158</b>
16.1 Mechanik . . . . .	158
16.1.1 Skizzen und Konzepte . . . . .	158
16.1.2 CAD-Zeichnungen . . . . .	158
16.1.3 3D-Modelle . . . . .	158
16.2 Hardware . . . . .	159
16.2.1 Skizzen und Konzepte . . . . .	159
16.2.2 Stromlaufpläne . . . . .	160
16.2.3 Platinenlayouts . . . . .	167
16.2.4 Bestückungslisten . . . . .	173
16.3 Software . . . . .	185
16.3.1 Skizzen und Konzepte . . . . .	185
16.3.2 Diagramme . . . . .	185
16.4 Software . . . . .	185
16.4.1 Skizzen und Konzepte . . . . .	185
16.4.2 Code . . . . .	188
16.4.2.1 Eingabesubsystem . . . . .	188
16.4.2.2 Ausgabesubsystem . . . . .	188
16.4.2.3 User-Interface . . . . .	188
<b>17 Abbildungsverzeichnis</b>	<b>189</b>
<b>18 Tabellenverzeichnis</b>	<b>190</b>
<b>19 Literaturverzeichnis</b>	<b>191</b>

---

# **1 Einleitung**

In einer sterilen Umgebung ist es unerlässlich, eine keimfreie Methode zu haben, um Aufgaben zu erledigen, die normalerweise von menschlichen Händen ausgeführt werden. Beispielsweise muss ein Objekt gegriffen und an eine andere Position innerhalb dieser geschlossenen Umgebung bewegt werden. Zusätzlich ist es hilfreich, eine Möglichkeit zur Analyse gefährlicher Güter zu haben. Diese Anforderungen können durch eine bionische Hand erfüllt werden. Diese Hand wird drahtlos vom Menschen über eine, in einen Handschuh integrierte elektronische Schaltung, gesteuert. Weitere Hardware-, Software- und Mechanikkomponenten sind erforderlich, um den gewünschten Anwendungsfall zu realisieren. Ein Prototyp eines Roboterarms wird entwickelt, der je nach Einsatzgebiet drahtlos gesteuert und vielfältig eingesetzt werden kann.

Diese Diplomarbeit wurde von mehreren Autoren verfasst und ist in verschiedene Kapitel unterteilt, um eine klare Struktur zu gewährleisten. Dadurch können die benötigten Informationen schnell gefunden werden. Zunächst werden grundlegende theoretische Kenntnisse festgehalten, um den Einstieg in das Thema zu erleichtern. Anschließend beschreiben wir die Umsetzung der Mechanik, gefolgt von der Hardware und der Software. Alle relevanten Aspekte der bionischen Hand werden in diesen Hauptkapiteln abgedeckt. Darauf hinaus werden Tests, Messungen und die während der Entwicklungsphase erzielten Ergebnisse aufgeführt und ausgewertet. Die abschließenden Erkenntnisse geben einen Überblick darüber, inwiefern die gewünschte Realisierung möglich ist. Jedes dieser Hauptkapitel enthält auch Unterkapitel, um die jeweilige Thematik detaillierter zu beschreiben.

In dieser Diplomarbeit sind alle durchgeführten Überlegungen und Entwicklungsschritte genau dokumentiert. Dadurch wird der gesamte Arbeitsfortschritt nachvollziehbar. Die verwendeten Informationen stammen aus eigener Erfahrung, Internetrecherchen oder Fachliteratur. Falls externe Quellen genutzt wurden, sind diese in Fußnoten mit entsprechenden Verweisen gekennzeichnet.

## **1.1 Ausgangslage und grundlegende Motivation**

Es gibt Produktionsbereiche, die klinisch sauber gehalten werden müssen, da durch Menschen Kontaminationen entstehen können. Für dieses Vorhaben soll ein Prototyp einer Roboterhand, die über einen Sensorhandschuh kabellos gesteuert wird, entwickelt und aufgebaut werden. Diverse Parameter der Roboterhand sollen in einem Interface für den Benutzer dargestellt werden.

## **1.2 Themenerläuterung**

Ziel des Projekts ist es, eine Roboterhand zu bauen, die über einen kabellos verbundenen Handschuh gesteuert werden kann. Das Ziel ist es, Daten der Fingergelenkssensoren auszulesen, zu übertragen und die Bewegungen mit der Roboterhand nachzustellen. Endziel ist es, Daten richtig zu verarbeiten und die Roboterhand entsprechend zu bewegen. Daten sollen in einem Interface dargestellt werden.

### **1.3 Grundgliederung der Arbeit**

---

Es soll eine Erfassung der Sensordaten möglich sein. Diese sollen übertragen und empfangen werden können. Die Roboterhand soll die Finger nach der Vorgabe des Handschuhs bewegen können. Als Endergebnis soll eine halbvolle 500mL Plastikflasche umschlossen und in der Luft gehalten werden. In dem Interface, dem User-Interface, sollen wichtige Daten dargestellt und Parameter verändert werden können.

### **1.3 Grundgliederung der Arbeit**

Folgende Schüler des TGMs haben, das in Unterabschnitt 1.2 erläuterte Projekt, entwickelt und gefertigt:

Projektleiter: Ladislaus Szabo (Hardwareentwicklung, PCB-Design, Projektleitung)

Mitarbeiter: Fabian Schweitzer (Mikrokontroller-Programmierung, Testmanagement, Gesamtintegration)

Amir Al-Maytah (3D-Druck, Mechanik, Userinterface-Programmierung)

Die folgenden beiden Betreuer haben uns jeder Zeit geholfen:

Dipl.Ing. Christoph Diemberger

Fachlehrer Robert Offner

### **1.4 Untersuchungsanliegen der individuellen Themenstellungen**

Mit dieser Diplomarbeit soll eine Roboterhand nach einem fertigen Design aufgebaut werden, die mit einem kabellos angebundenen Handschuh gesteuert wird. Zu diesem Zweck müssen verschiedene Sensoren getestet werden, die die Fingergelenksstellung messen. Deren Daten müssen ausgelesen und mittels kabelloser Schnittstelle an die Roboterhand übertragen werden (Schweitzer). Um die Fingerbewegungen an der Roboterhand nachzustellen, muss folglich eine Motoransteuerung entwickelt werden (Szabo). Um die elektronischen Bauelemente unterzubringen, muss ein entsprechendes Gehäuse, in Form einer menschlichen Hand beziehungsweise eines Arms, gefertigt werden. Zusätzlich soll ein Interface erstellt werden, in dem der Benutzer Daten des Roboterarms und des Handschuhs einsehen kann. (Al-Maytah)

---

## 2 Lastenheft

Tabelle 1: Lastenheft

Requirement Nr.	Requirement Text	Mandatory (Y/N)	Erfüllungsgrad
1	Es soll ein Handschuh (Eingabe) entwickelt werden, mit dem man eine Roboterhand (Ausgabe) steuern kann.	Y	
2	Der Handschuh soll kabellos mit der Roboterhand verbunden werden können.	Y	
3	Die Bewegungen der Roboterhand sollen möglichst gut die einer echten Hand immitieren.	Y	
4	Die Roboterhand ist ein fertiges Design (3D Druck), dass im Projekt integriert wird.	Y	
5	Die Bewegungserfassung soll mit entsprechender Sensorik erfolgen.	Y	
6	- Flex Sensoren für die Finger	Y	
7	- Gyro Sensor für die Drehung des Handgelenks	N	
8	Die Datenauswertung soll mit einem Mikrokontroller erfolgen.	Y	
9	Eine Platine für das Verbinden aller Komponenten am Handschuh muss gebaut werden.	Y	
10	- soll auf den Handrücken des Handschuhs passen	Y	

Tabelle 2: Lastenheft

Requirement Nr.	Requirement Text	Mandatory (Y/N)	Erfüllungsgrad
11	- Anschlussmöglichkeiten für einen Akku (Akkupack oder LiPo), plus alternativ für eine externe Stromversorgung müssen vorhanden sein	Y	
12	- USB Anschluss zum Programmieren des ESP32	Y	
13	- Upload und Reset Button für ESP32	Y	
14	- Annähen?	N	
15	Mit einem ADC, sollen mindestens 30 verschiedene Positionen der Flexsensoren detektiert werden können.	Y	
16	- diese Positionen sollen wiederherstellbar sein	Y	
17	Das Maximalgewicht des Handschuhs soll 500g nicht übersteigen.	Y	
18	Fertigung mittels 3D-Druck (fertiges Design)	Y	
19	- metallische Gelenke für die Finger	Y	
20	- Abmessungen der Hand und Finger wie eine echte !!	Y	
21	Drucksensoren an den Fingerspitzen	N	
22	Nachstellung der Bewegungen mit Motoren	Y	
23	- Servomotoren	Y	
24	Die Finger sollen sich kontrolliert zum und vom Handballen weg bewegen können.	Y	

Tabelle 3: Lastenheft

Requirement Nr.	Requirement Text	Mandatory (Y/N)	Erfüllungsgrad
25	Die Finger sollen sich seitlich bewegen können.	N	
26	Die Finger sollen sich zitterfrei bewegen können.	Y	
27	Die Finger sollen sich störungsfrei bewegen können.	Y	
28	- stabile kabellose Verbindung !! (störungstolerantes Kommunikationsprotokoll)	Y	
29	- stabile Spannungsversorgung für jeden einzelnen Motor	Y	
30	- Mikrokontroller für die Kommunikation mit dem Handschuh und die Steuerung der Servomotoren (ESP32)	Y	
31	- Anschluss für externes Netzteil	Y	
32	- USB Anschlus zum Programmieren des ESP32	Y	
33	- Upload -und Reset Button für ESP32	Y	
34	Jeder einzelne Servomotor soll eine Stromüberwachung haben.	Y	
35	- mit dieser soll die Griffkraft der Finger kontrolliert werden können. (leicht - mittel - stark)	Y	
36	- die Parameter der Griffkraft soll eingestellt werden können.	Y	

---

Tabelle 4: Lastenheft

Requirement Nr.	Requirement Text	Mandatory (Y/N)	Erfüllungsgrad
37	Die Hand soll eine 500mL Plastikflasche, die zur Hälfte gefüllt ist, als Endziel greifen können und in der Luft halten.	Y	
38	Das User Interface soll folgende Elemente aufweisen:	Y	
39	- 3D Modell der Hand	N	
40	- Griffkraftanzeige (in kg)	Y	
41	- Anzeige der Servostellung (Winkel)	Y	
42	- in Form einer Applikation	Y	

---

### **3 Vorwissenschaftlicher Theorienteil**

#### **3.1 Platzhalter Thema Amir**

#### **3.2 Platzhalter Thema Fabian**

#### **3.3 Sicherheitsaspekte der Mensch-Roboter-Kollaboration**

---

## 4 Theoretische Grundlagen des Diplomprojekts

### 4.1 Flexsensor

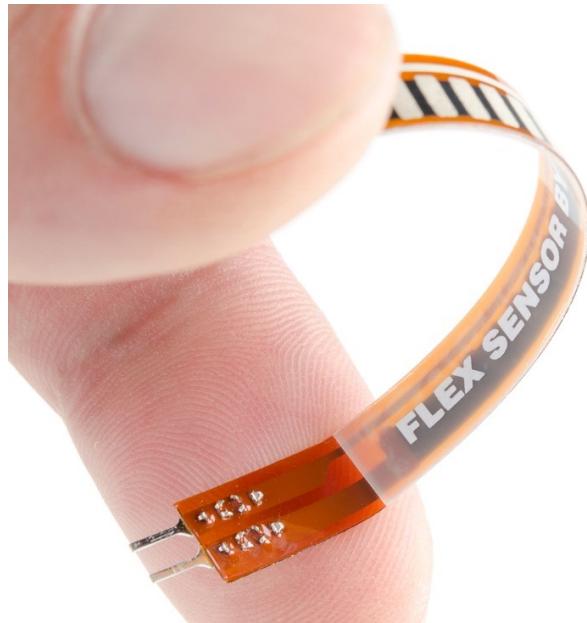


Abbildung 1: Flexsensor

[https://commons.wikimedia.org/wiki/File:Flex\\_Sensor.jpg#/media/File:Flex\\_Sensor.jpg](https://commons.wikimedia.org/wiki/File:Flex_Sensor.jpg#/media/File:Flex_Sensor.jpg)

Ein Flexsensor (Biegesensor) misst die Biegung des aus Kunststoff bestehenden Streifens. Die Kohlenstoffbeschichtung macht diesen dann zu einem variablen Widerstand. Je mehr der Flexsensor gebogen wird, desto größer wird der Widerstand. Die Biegung kann somit gemessen werden. Diese Information wird dann in ein elektrisches Signal umgewandelt, was auch für das Einlesen in den Mikrocontroller nötig ist. Diese Flexsensoren können beispielsweise auf einem Handschuh angenäht werden, um zu messen, wie sehr ein Finger abgebogen wurde.

## 4.2 Servomotor



Abbildung 2: Servomotor

(Quelle: [https://3.bp.blogspot.com/-\\_Qhk9H-K3tI/UVQhc8-bxSI/AAAAAAAABT4/bWZ8dEICtZY/s640/Futaba\\_S3003\\_Servo.jpg](https://3.bp.blogspot.com/-_Qhk9H-K3tI/UVQhc8-bxSI/AAAAAAAABT4/bWZ8dEICtZY/s640/Futaba_S3003_Servo.jpg))

Ein Servomotor ist ein kleiner Elektromotor. Er besteht aus einer Steuereinheit, sowie einer Antriebseinheit. Ein Servo kann die Winkelposition einstellen, die Drehgeschwindigkeit variieren und dadurch auch die Beschleunigung beeinflussen. Dies funktioniert über einen Sensor, der die Positionsbestimmung durchführen kann. Der Servo kann über die Software festlegen, wie sich dieser drehen soll. Durch Sollwerte wird angegeben, wie weit sich dieser drehen soll. Für die Ansteuerung von einzelnen 3D-gedruckten Fingern ist ein Servo die optimale Wahl, da dieser die an ihm befestigten Schnüre spannen oder entlasten kann.

## 4.3 ESP-NOW **Fabian**

Das ESP-NOW Protokoll macht es möglich, dass man Daten problemlos per Funkstrecke versenden kann. Dabei kann dieses auf dem ESP32 und ESP8266 verwendet werden, da nur diese dafür ausgewählt wurden. Der Hersteller "Espressif" hat dieses Protokoll entwickelt. Es können bis zu 250 Bytes pro Sendevorgang ausgetauscht werden. Dies ist auch bis zu einer maximalen Anzahl von zwanzig ESP32/ESP8266 Slaves möglich. Das Mischen der beiden genannten Boards ist

### 4.3 ESP-NOW Fabian

---

möglich. Dabei kann auch jeder der ESPs jeweils als Transmitter, Receiver oder Transceiver verwendet werden. Ein Vorteil dieser Übertragungsmöglichkeit ist, dass die dafür benötigten Bibliotheken in der Arduino IDE zu finden sind. Diese sind nämlich schon standardmäßig im Paket für den ESP32 und ESP8266 vorhanden.

Als Erstes muss die MAC-Adresse des jeweiligen ESPs identifiziert werden. Bei Bedarf kann diese auch geändert werden. Es wird hierfür also kein Router benötigt. Diese besteht aus sechs Bytes. Angegeben wird sie im Hexadezimalsystem, wobei zwischen Zweierblöcken von Buchstaben und Zahlen jeweils ein Doppelpunkt steht. Um auf alle WIFI-Funktionen nutzen zu können, muss auch die Bibliothek für WIFI eingebunden werden. Wichtig ist, dass WIFI zuvor aktiviert wurde, da sonst keine Daten übertragen werden können. Durch den Aufruf der Funktion `esp_now_init()` kann ESP-NOW initialisiert werden, mit `esp_now_deinit()` kann es wieder rückgängig gemacht werden. Durch die Funktion `esp_now_add_peer()` können neue Geräte, an die man dann etwas sendet, in die Liste der Geräte aufgenommen werden. Zu beachten ist, dass Kanäle von 0 bis 14 vorhanden sind, wobei 0 für den aktuellen Kanal steht. Darüber wird dann auch gesendet, außer man stellt einen spezifischen Kanal ein, auf dem sich das lokale Gerät befindet.

Um Daten zu senden, muss die Funktion `esp_now_send()` aufgerufen werden. Dadurch können dann Daten gesendet werden. Die Funktion `esp_now_register_send_cb()` ist dafür da, um die Sende-Callback-Funktion zu registrieren. Es wird ein Wert zurückgeliefert, der angibt, ob die Daten erfolgreich gesendet wurden oder nicht. Bei einem erfolgreichen Sendevorgang wird `ESP_NOW_SEND_SUCCESS` zurückgeliefert. Dies passiert, wenn sie erfolgreich am MAC-Layer der Empfängerseite empfangen wurden, ansonsten `ESP_NOW_SEND_FAIL`. Warum eine Fehlermeldung ausgegeben wird, kann nicht nur einen Grund haben. Beispielsweise existiert die angestrebte MAC-Adresse nicht, die Kanäle sind nicht identisch oder der Action frame geht bei der drahtlosen Übertragung verloren. Um doppelte Daten auszuschließen, ist es möglich, dass man eine Sequenznummer verwendet, um zu überprüfen, ob es eine fehlerbehaftete Übertragung sein könnte. Ein zu kurzes Intervall zwischen dem Senden der verschiedenen Daten kann dazu führen, dass eine Störung auftreten kann. Die Sende-Callback-Funktion kann dadurch gestört werden. Es sollte deshalb auf jeden Fall der Moment abgewartet werden, bis die Sende-Callback-Funktion eine Antwort liefert hat. Dabei wird diese Funktion jedes Mal mit hoher Priorität beim Wifi-Task ausgeführt. In der Callback-Funktion sollen auf jeden Fall nur die wichtigsten Operationen abgearbeitet werden.

Um Daten zu empfangen, muss die Funktion `esp_now_register_recv_cb()` aufgerufen werden. Dies ist dafür verantwortlich, um die Empfangs-Callback-Funktion zu realisieren. Genauso, wie beim Sender, wird hierzu der WIFI-Task verwendet, weshalb nur die nötigsten Operationen abgearbeitet werden sollen.

Mit der Funktion `esp_wifi_config_espnow_rate()` kann die ESP-NOW Datenrate angegeben werden. Dabei muss beachtet werden, dass die gewünschte Schnittstelle bereits konfiguriert wurde. Eine Energiesparen Funktion kann über `esp_wifi_connectionless_module_set_wake_interval()` festgelegt werden. Hier meist durch eine Wand, in seltenen Fällen sogar durch zwei, abhängig von der Dicke kann man dann einstellen, wann sich der ESP einschalten soll.

Im Freien, ohne störende Wände oder Möbel, kann ESP-NOW Daten auf jeden Fall 100 bis 250

Meter weit übertragen. In geschlossenen Räumen funktioniert die Übertragung und anderen störenden Einflüssen.<sup>1</sup>

Das OSI-Modell wird bei dem ESP-NOW Modell von sieben Schichten auf zwei reduziert. Daten müssen somit nicht den Network-Layer, den Transport-Layer, den Session-Layer, den Presentation-Layer und den Application-Layer durchlaufen. Nur der Data-Link-Layer (SLIP/PPP/MTU), sowie der Physical-Layer (802.11b/g/n), werden verwendet. Der große Vorteil liegt außerdem darin, dass man einige Dinge nicht berücksichtigen muss. Es gibt keine Paketköpfe oder Entpacker auf jeder Schicht. Dies führt zu einer schnelleren Reaktion, zu weniger Overhead und die Paketverluste werden dadurch in überlasteten Netzen reduziert. Dies führt zu einer deutlichen Verbesserung der möglichen Verzögerung, indem diese sehr niedrig gehalten werden kann.<sup>2</sup>

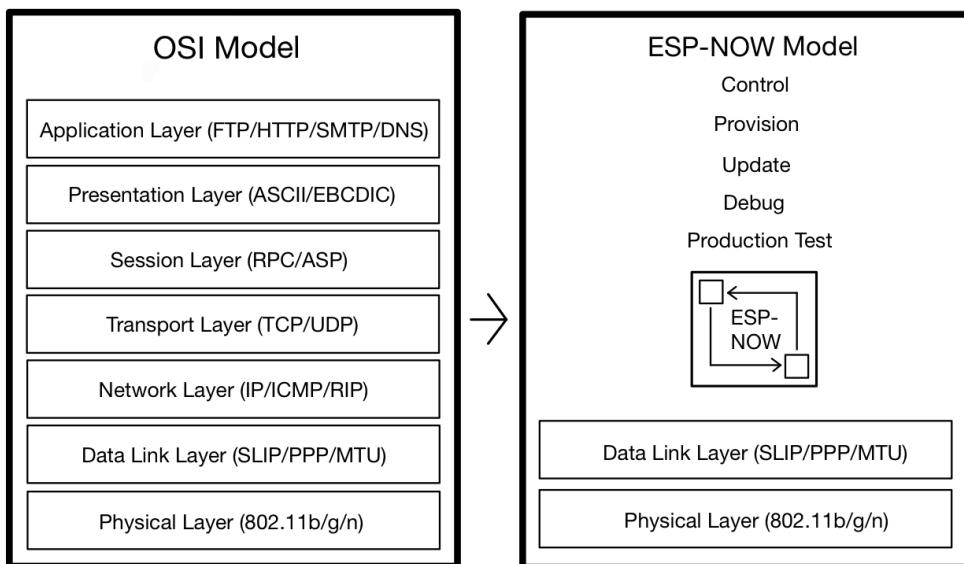


Abbildung 3: ESPNOW - Model

#### 4.3.1 OSI-Modell

Das OSI-Modell ist standardisiert und ermöglicht die Kommunikation zwischen den verschiedensten Computersystemen. OSI (Open Systems Interconnection) wurde von der ISO (International Organization for Standardization) erfunden. Es besteht aus 7 Schichten (application layer, presentation layer, session layer, transport layer, network layer, data link layer und physical layer). Wichtig ist, dass jede einzelne Schichte eine andere Aufgabe erfüllt. Durch das System der

<sup>1</sup><https://wolles-elektronikkiste.de/esp-now#range>, (Stand: 21.02.2024)

<sup>2</sup><https://www.espressif.com/en/solutions/low-power-solutions/esp-now>, (Stand: 21.02.2024)

einzelnen Schichten kann die Fehlersuche vereinfacht werden, da man die jeweiligen durchforsten kann. Zu beachten ist allerdings, dass diese untereinander ebenfalls Daten austauschen und untereinander kommunizieren können. Dabei ist jede Schicht nach international standardisierten Protokollen definiert. Das bedeutet, dass gewisse Regeln zur Kommunikation eingehalten werden müssen. Dabei ist es möglich, dass sich ein Protokoll über mehrere Schichten verteilt.

Beim Sender und Empfänger werden diese sieben Schichten jeweils einmal angewandt.

### 4.3.1.1 Physical Layer

Beschreibt die mechanische und funktionale Schnittstelle zum Übertragungsmedium. Die Datenübertragung der physischen Geräte, wie Schalter, Kabel, usw. ist hiermit gemeint. Daten werden in einen Bitstrom umgewandelt. Zu beachten ist, dass das Übertragungsmedium an sich nicht zur 1. Schicht gehört, nur die Datenübertragung bei Einschalten des physischen Geräts gehört in diese.

### 4.3.1.2 Data Link Layer

Beschreibt die Sicherungsschicht, die eine stabile und funktionierende Verbindung zwischen Endgerät und Übertragungsmedium herstellt. Der Data Link Layer verarbeitet Pakete vom Network Layer, spaltet diese in kleinere Teile namens Frames und ist für den Datentransfer zwischen zwei Geräten im selben Netzwerk verantwortlich. Die physikalische Adressierung von Datenpaketen wird hier durchgeführt. Außerdem werden auch eine Fehlererkennung, Fehlerbehebung und Datenflusskontrolle angewandt.

### 4.3.1.3 Network Layer

Beschreibt den regulierten Datentransfer zwischen zwei verschiedenen Netzwerken. Hier werden einzelne Segmente auf dem Sendergerät von dem Transport Layer in Fragmente aufgeteilt und auf dem Empfängergerät dann wieder zusammengefügt. Die logische Adressierung findet ebenfalls statt.

### 4.3.1.4 Transport Layer

Beschreibt das Bindeglied zwischen den transportorientierten und den anwendungsorientierten Schichten. Wie immer, werden die Daten in kleinere Bestandteile zerlegt. Header-Informationen und eine Fehlerkontrolle werden von den Segmenten beinhaltet, falls nicht alle Daten erfolgreich angekommen sind oder fallengelassen wurden, kann dies hilfreich sein.

#### 4.3.1.5 Session Layer

Beschreibt die Verbindung zwischen den Endsystemen. Session nennt man den zeitlichen Bereich zwischen dem Starten und Beenden der Kommunikation. Wichtig ist auch, dass dieser Layer dafür verantwortlich ist, dass genug lange Daten übertragen werden können und die Verbindung nicht schon zuvor beendet wird. Für die Effizienz soll allerdings nach einem abgeschlossenen Vorgang die Verbindung unterbrochen werden.

#### 4.3.1.6 Presentation Layer

Beschreibt die Aufbereitung der Daten, sodass der Nutzer diese verstehen kann. Damit ist gemeint, dass beispielsweise verschlüsselte Daten zuerst entschlüsselt werden müssen, bevor sie der Nutzer verstehen kann. Die Komprimierung und Dekomprimierung von Daten werden hier auch durchgeführt.

#### 4.3.1.7 Application Layer

Beschreibt die Bereitstellung von Funktionen für Anwendungen und die Verbindung zu den unteren Schichten. Die Dateneingabe und Datenausgabe werden hier ebenfalls durchgeführt. Softwareanwendungen sind auf den Application Layer angewiesen ([http](#), [SMTP](#), ...). Diese Protokolle sind für die Kommunikation notwendig und standardisiert.

#### 4.3.1.8 ESP-NOW Schichten

Es gibt bei ESP-NOW zwei Schichten, den Data Link Layer und den Physical Layer (siehe Unterunterabschnitt 4.3.1). Dabei beruht der Physical Layer auf dem IEEE 802.11b/g/n Standard. Da jedoch nicht so viele Schichten, wie bei dem OSI-Modell vorhanden sind, gibt es mehrere Möglichkeiten, wo Fehler liegen können und es ist nicht so leicht den Fehler an der richtigen Stelle zu suchen.

#### 4.3.1.9 IEEE 802.11b/g/n Standard

Es ist ein internationaler Standard für die drahtlose Kommunikation, der von Electrical and Electronics Engineers (IEEE) entwickelt wurde. Besonders wird Wert auf die geringe Datenrate und dem daraus folgend geringen Stromverbrauch Wert gelegt. Der 802.11b Standard hat eine maximale Datenübertragungsrate von 11Mbit/s, 802.11g eine maximale Datenübertragungsrate von 54Mbit/s und 802.11, kann bis zu 600Mbit/s übertragen. Der letztgenannte Standard ist der einzige, der auch das 5GHz-Band nutzen kann, alle drei können jedoch das 2,4GHz-Band zum Datenaustausch verwenden.

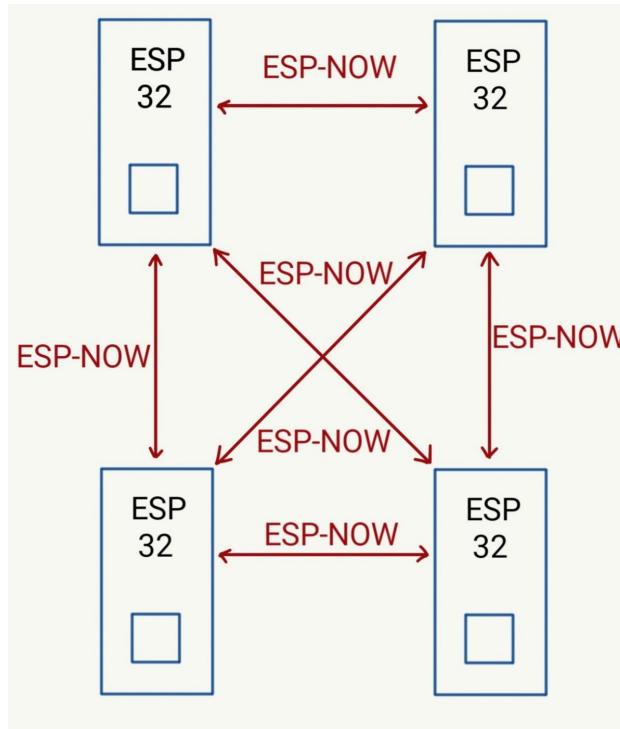


Abbildung 4: ESPNOW - Theorie

## 4.4 OPV Technologien Laci

Ein Operationsverstärker, kurz OPV, ist ein analoger Gleichspannungsverstärker, der sich durch seine üblicherweise sehr hohe Verstärkung auszeichnet. Der elektroische Grundbaustein ist ein Differenzverstärker, der durch äußere Beschaltungen angepasst werden kann. In der grundlegendsten Funktion, verstärkt ein OPV die Differenz der beiden Eingangsspannungen am "+" und "-" Anschluss. Das Ergebnis dieses Verstärkungsprozesses wird anschließend über "Vout" ausgegeben.

Operationsverstärker können mit einem sogenannten "Single-Supply" oder "Dual-Supply" versorgt werden. Dies bezieht sich auf die Art der Versorgungsspannung.

### 4.4.1 Single-Supply

Bei dieser Art der Versorgung, wird ausschließlich eine positive Spannung und das Groundpotential genutzt. Das bedeutet, dass der OPV theoretisch einen maximalen Ausgangsbereich von GND bis VCC haben kann. Negative Spannungen können nicht ausgegeben werden.

#### 4.4.2 Dual-Supply

Beim Dual-Supply Betrieb, wird der OPV von einer positiven und negativen Spannung versorgt. Das bedeutet, dass der OPV nun die Fähigkeit besitzt auch negative Spannungen auszugeben und der Ausgangsspannungsbereich somit größer als beim Single-Supply ist.

#### 4.4.3 Rail-to-Rail Technologie

Hat ein OPV die sogenannte "Rail-to-Rail Technologie", so ist die Rede von Spannweite des Ausgangsspannungsbereichs. Bei diesem Feature ist der Operationsverstärker dazu fähig fast bis auf das positive und negative Versorgungspotential Spannung auszugeben. Dies ist oft von Vorteil, da dadurch die Versorgungsspannungen nicht übermäßig hoch sind und somit die Schaltungen mit niedrigeren Spannungen arbeiten kann. Dies ist auch für die Benutzer der entsprechenden Endgeräte sicherer.

In Abbildung 5 sind die Ausgangsspannungsbereiche eines normalen OPVs, in blau, und eines OPVs mit Rail-to-Rail Technologie, in Orange, dargestellt.

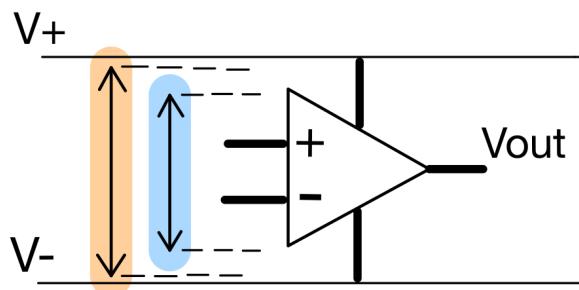


Abbildung 5: OPV Ausgangsspannungsbereiche

## 4.5 Analog-Digital-Wandler

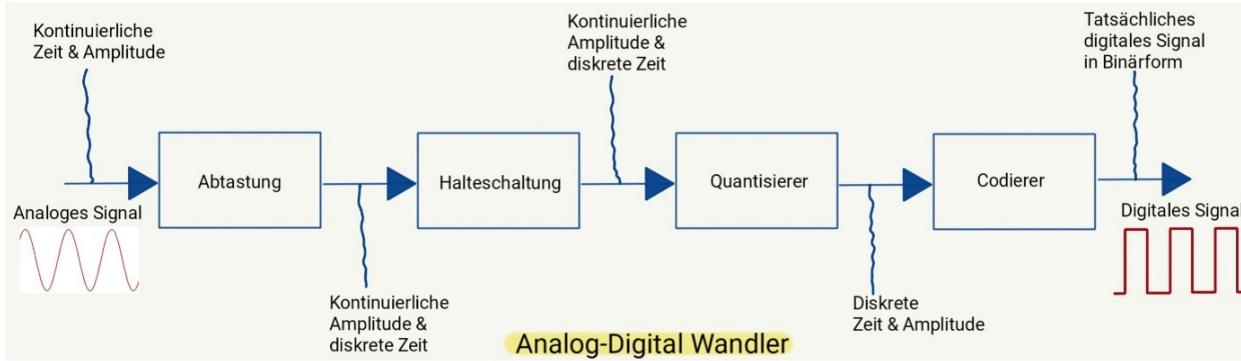


Abbildung 6: Blockschaltbild Analog-Digital-Wandler

Ein ADC (Analog-Digital-Converter) wandelt kontinuierliche analoge Signale (physikalische Größen) in digitale Signale um. Dies ist notwendig, damit ein Prozessor die Werte verarbeiten kann. Deshalb kommen ADCs in elektronischen Schaltungen häufig vor.

**Abtastung:** Es werden kontinuierliche analoge Signale mit einer bestimmten Abtastrate abgetastet. Das kontinuierliche Signal wird hierbei zu einem zeitdiskreten, wobei die Amplitude stets kontinuierlich bleibt.

**Halteschaltung:**

Die nach der Abtastung vorhandenen Messwerte werden in der Halteschaltung so lange zurückgehalten, bis ein neuer die Halteschaltung erreicht.

**Quantisierer:**

Die Messwerte werden hier in diskrete Amplitudenwerte aufgeteilt. Das Signal ist ab dem Quantisierer zeitdiskret und besitzt nun eine diskrete Amplitude.

**Codierer:**

Der Codierer wandelt dann das quantisierte Signal in ein binäres um. Am Ende ist dann ein digitales Signal in Binärform vorhanden.

## 4.6 Multiplexer

Untenstehend kann man noch anhand einer einfachen Logikschaltung eines 2-zu-1 Multiplexers die Funktionsweise eines Multiplexers. Grundsätzlich wird über elektrische Signale einer von mehreren Eingangssignalen an den Ausgang durchgeschaltet. Meist funktioniert dies durch Halbleiterelemente. Das Steuersignal s schaltet noch zusätzlich.

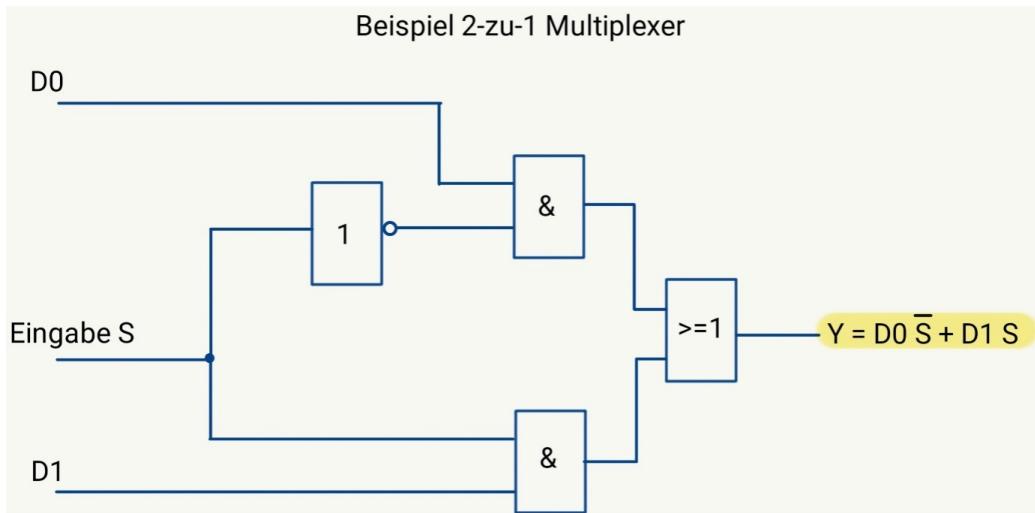


Abbildung 7: Multiplexer Beispiel

## 4.7 PWM-Signale

## 4.8 I2C Funktionsweise

### I2C Funktionsweise

I2C beschreibt eine Möglichkeit, über die zwei verschiedene Geräte miteinander kommunizieren können. Die Daten werden hintereinander gesendet und dies über zwei Datenleitungen. Es kann dabei mit bis zu 128 Teilnehmern kommuniziert werden. Am Ende interpretiert der I2C-Bus die Daten und steuert den Ablauf. Bei unserem Projekt ist der ESP32 der Master und die Flexsensoren beispielsweise die Slaves. Der Master ist dann der wichtige Bestandteil, da dieser für alle anderen Slaves derjenige ist, der sagt, was welches Bauteil machen soll. Dabei hängen alle Geräte beziehungsweise Bauteile an zwei Leitungen. Eine heißt SDA (serial data) und ist für die serielle Übertragung der Daten verantwortlich. Jedoch ist zu erwähnen, dass diese bidirektional benutzt wird. Das heißt es kann in beide Richtungen gesendet werden (Master zum Slave und umgekehrt). Es muss aber der Master regeln, also entweder Daten senden oder welche von einem Slave anfordern. Die zweite Leitung heißt SCL (serial clock) und ist für den Takt zuständig, der für alle per I2C verbundenen Geräte gilt. Es wird also ein synchroner Bus verwendet.

**Beispiel Protokoll I2C** Die Daten sind grundsätzlich in Datenpaketen verpackt, jedes besteht aus 1 Byte. Wenn nichts kommuniziert wird, dann hat SDA und SCL den Wert 1. Nachdem eine Start-Condition gesendet wurde, durch die alle Slaves wissen, dass sie nun auf den Master jederzeit reagieren sollen, wird die Adresse des Slaves gesendet, mit dem nun Daten ausgetauscht werden sollen. 7 Bit werden hierfür standardmäßig verwendet. Danach wird ein weiteres Bit gesetzt. Dieses kann den Wert 0, was Schreiben bedeutet, oder 1, was lesen bedeutet, haben. Nun ist der Slave an der Reihe. Dieser muss ebenso 0, wenn die Kommunikation bereitsteht, an den Master zurücksenden. Der Master sendet dann oder empfängt Daten vom Slave. Sobald alle Daten

versendet wurden, kommt die Bestätigung. Der ganze Vorgang des Senden und Empfangens wird am Ende dann durch die Stop-Condition beendet. SDA wird hier auf 1 gesetzt, währenddessen SCL auch den Wert 1 besitzt.

<http://fmh-studios.de/theorie/informationstechnik/i2c-bus/>

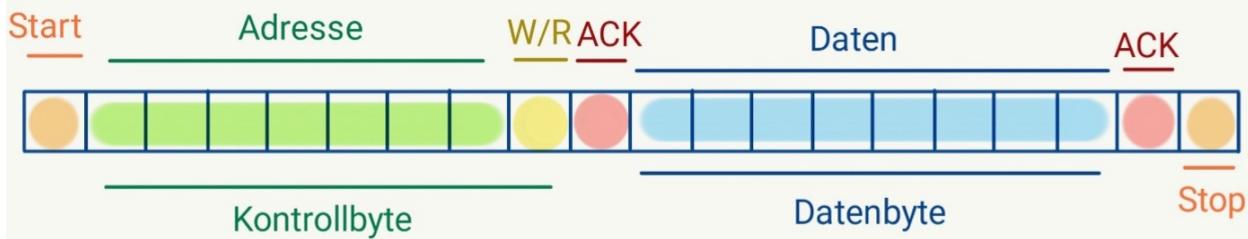


Abbildung 8: I2C Adressierung

## 4.9 UART Funktionsweise

Der Arduino, beziehungsweise der ESP, werden über die serielle Verbindung UART (Universal Asynchronous Receiver and Transmitter) vom Computer angesprochen. Dabei stellt diese Verbindung eine asynchrone Datenübertragung mit fester Baudrate, Datenbits, Paritätsbits und Stopbits dar. Es sind zwei Datenleitungen vorhanden, wobei eine für das Senden (TX) und die andere für das Empfangen (Rx) von Daten zuständig ist. Die Start- und Stopbits geben an, wann gesendet werden soll und wann dies nicht mehr geschehen soll. Sobald der Sender ein Startbit erhält, beginnt die Übertragung der Daten, aber beim Empfangen des Stopbit wird diese unterbrochen. Es wird dadurch keine Clock benötigt, da die Baud Rate dies regelt, allerdings ist zu beachten, dass diese maximal 10% von jeder des anderen Geräts abweichen darf. Die Baud Rate gibt an, wie viele Symbole pro Sekunde übertragen werden. Dabei kann ein Symbol ein einzelnes Bit oder eine Gruppe von Bits sein. Die Baudrate gibt also an in welchem Abstand jeweils ein Symbol gesendet wird ( $1s/Baudrate$ ) oder eben im Umkehrschluss auch, wie viele Symbole pro Sekunde gesendet werden ( $Baudrate/1s$ ).

**Beispiel Protokoll für 8-Bit Packet** Es ist ein Startbit vorhanden (LOW), dann noch 8 Datenbits (inklusive einem Paritybit, das Fehler identifizieren kann) und ein oder zwei Stopbits (HIGH). Zur Kontrolle, ob es eine gerade oder ungerade Anzahl von Highbits gibt, wird ein Paritybit mitgesendet, das angibt, ob die Anzahl dieser gerade oder ungerade ist. Es gibt klarerweise keine hundertprozentige Sicherheit, aber man kann sich mehr auf die Nutzbarkeit der Daten verlassen. Bei der UART-Kommunikation bedeutet ein Paritybit von 0 (odd), dass die Summe der Highbits eine ungerade Zahl ist. Ein Paritybit von 1 (even) hingegen zeigt an, dass die Summe der Highbits eine gerade Zahl ist. Üblicherweise werden zwei Stopbits am Ende einer Übertragung gesendet, um eine verlängerte Ruhephase zu gewährleisten. Dies führt zu einer stabileren Synchronität und Erhöhung der Robustheit gegen Signalstörungen. Bei UART wird

das LSB (least significant bit) immer ganz rechts in den Daten gefunden. Sollte das Paritybit nicht stimmen, dann wird vom Empfänger an den Sender eine Anfrage mit der Bitte um erneutes Senden gesendet, da der Datensatz nicht stimmt.

<https://edistechlab.com/wie-funktioniert-uart/?v=fa868488740a>

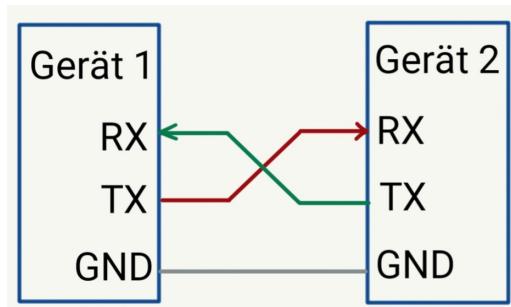


Abbildung 9: UART erklärendes Diagramm

## 4.10 WiFi



Abbildung 10: WiFi Logo

[https://commons.wikimedia.org/wiki/File:WiFi\\_Logo.svg#/media/Datei:WiFi\\_Logo.svg](https://commons.wikimedia.org/wiki/File:WiFi_Logo.svg#/media/Datei:WiFi_Logo.svg)

WLAN (Wireless Local Area Network) ist eine Technologie, die Geräte drahtlos in einem lokalen Netzwerk verbindet. Es wird häufig verwendet, um einen Zugang zum Internet herzustellen. Mobile Geräte wie Smartphones, Tablets und Laptops werden meistens über WLAN mit dem Internet oder untereinander verbunden. WIFI (Wireless Fidelity) ist eigentlich ein Markenname, der oft fälschlicherweise als Synonym für WLAN verwendet wird. Innerhalb eines WLANs können Datenpakete gesendet und empfangen werden. Der Vorteil dieser drahtlosen Verbindung liegt darin, dass die Geräte keine physische Verbindung per Datenkabel benötigen. Sie können innerhalb einer bestimmten Reichweite miteinander kommunizieren. Die Reichweite eines WLANs kann durch verschiedene Faktoren beeinflusst werden. Beispielsweise können physische Hindernisse

wie Wände oder Möbel das Signal abschwächen. Auch andere drahtlose Geräte oder Netzwerke können Interferenzen verursachen und die Signalqualität beeinträchtigen.

# 5 Grundlegende Systemkonzepte

## 5.1 Gesamtsystemkonzept

### 5.1.1 Systembeschreibung

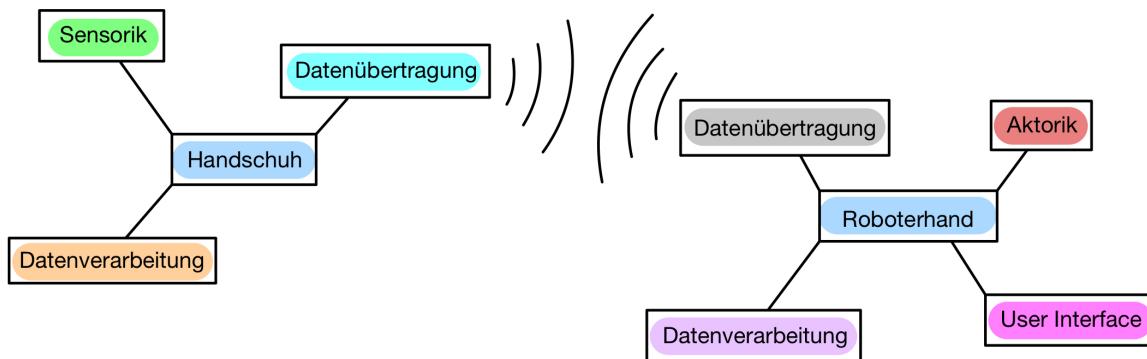


Abbildung 11: Blockschaltbild des Gesamtsystems

Das Gesamtsystem ist in Abbildung 11 grafisch veranschaulicht. Die linke Hälfte zeigt das Konzept des Handschuhs (Eingabe) und die rechte Hälfte die Roboterhand (Ausgabe). Die grundlegende Konzeptionierung der Bewegungserfassung besteht darin, dass mithilfe von Sensoren die Fingerbeugung aller fünf Finger gemessen wird. Anschließend werden die Sensordaten ausgewertet und verarbeitet. Die resultierenden Informationen werden anschließend über eine drahtlose Verbindung an das zweite Subsystem, nämlich der Roboterhand, übertragen. Dort wird das Gesendete empfangen, weiterverarbeitet und interpretiert. Die Aktorik setzt die digitalen Anweisungen in mechanische Bewegungen um, wodurch die Roboterhand gesteuert wird. Im Ausgabesystem ist auch ein User-Interface enthalten. Die diversen Funktionen werden in späteren Punkten näher erläutert.

## 5.2 Eingabesubsystem

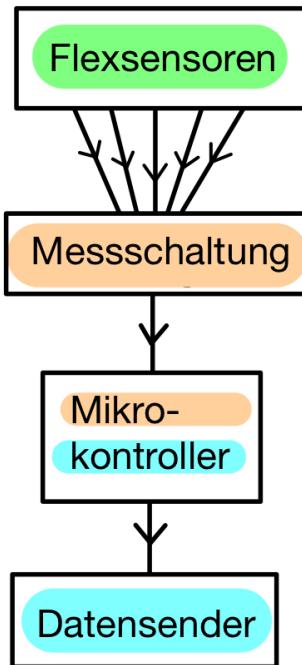


Abbildung 12: Blockschaltbild des Eingabesubsystems

Der Handschuh ermöglicht dem Benutzer die Roboterhand zu steuern. Dies geschieht durch Flexsensoren, die an den Fingern des Handschuhs angebracht sind. Wird ein Finger gebeugt, so ändert sich der Widerstandswert des Sensors. Diese Änderungen werden von einem Mikrokontroller erfasst, der per Kabel über UART die Werte kontinuierlich ausliest. Anschließend wird die kabellose Übertragung über den Funkstandard ESP-NOW realisiert. Das Senden an die Roboterhand wird vorbereitet. Damit der Handschuh funktioniert, bedarf es einer Spannungsversorgung. Diese wird in Form einer Batterie oder eines Akkus vorgesehen sein, funktioniert aber allenfalls über ein USB-C Kabel. Für einen möglichen stationären Betrieb kann der Handschuh auch mit einem Netzteil betrieben werden. Die Passform des Handschuhs soll möglichst komfortabel sein, um eine möglichst lange Benützung zu ermöglichen. Dies setzt eine gut überlegte Integration der Elektronik voraus. Die Flexsensoren sollen fest auf dem Handschuh befestigt sein. Die Widerstandswerte sind ansonsten bei jedem Mal biegen unterschiedlich, wenn die Flexsensoren nicht an einer Stelle am Handschuh bleiben, sondern sich verschieben. Diese Montage wird bei diesem Projekt auch bestmöglich beachtet. Der Handschuh ist so gebaut, dass er auch von Personen ohne elektronische Ausbildung und ohne Vorwissen bedient werden kann. Das liegt daran, dass nur die Finger gebeugt oder gestreckt werden müssen. Jede Handbewegung kann also gemacht werden, die Werte

werden per Software automatisch richtig verarbeitet.

### 5.3 Ausgabesubsystem

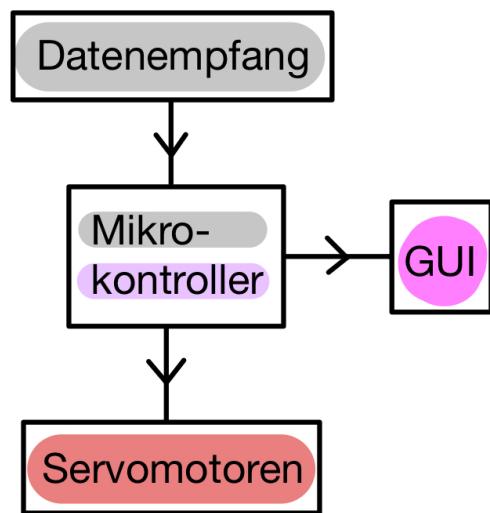


Abbildung 13: Blockschaltbild des Ausgabesubsystems

Sind die Daten des Eingabesubsystems empfangen, müssen diese in ein geeignetes Format für die Steuerung der Aktorik, also den Servomotoren, umgewandelt werden. Dies geschieht mit dem Mikrokontroller. Angebunden an das Ausgabesubsystem, ist ein Graphical-User-Interface, welches in Unterabschnitt 8.3 näher erklärt wird.

---

## 6 Mechanische Realisierung

### 6.1 Handschuh **Amir**

#### 6.1.1 Platzhalter für diverse Unterpunkte von Amir

## 6.2 Roboterhand **Amir**

### 6.2.1 Der Grundgedanke

Der Grundgedanke der Mechanik in der Roboterhand ist es alle drei Glieder eines Fingers, darunter das Fingergrundglied, das Fingermittelglied und das Fingerendglied mit ausschließlich der Bewegung eines einzelnen Servomotors auf oder zu zuklappen. Umsetzbar ist dies durch ein Zugsystem, welches für ein Aufklappen und ein Zuklappen des Fingers ermöglicht. Gezogen wird durch den Finger der Roboterhand ein oder mehrere Schnüre, ähnlich wie bei einer Marionette. Diese Schnüre bringen den Finger je nach Verlängern oder Verkürzen zum Bewegen. Ebenso ist je nach Zug wählbar in welcher Stellung der Finger sich dann befindet. Befestigt wird dies an einem mit dem Servomotor gesteuerten Zugmechanismus.

#### 6.2.1.1 Fertigung und CAD

Der Entwurf eines Modells ist mit ständiger Weiterentwicklung verbunden, sind in solch einem Modell mechanische Komponenten vorhanden nimmt die Fertigungsschleife ersichtlich mehr Komplexität an und erfordert die Möglichkeit in kürzester Zeit flexibel Anpassungen und Optimierungen am Entwurf ausgiebig zu testen. Speziell während der Entwicklung der mechanischen Komponente einer Roboterhand ist auf viele Faktoren zu achten, es ergeben sich ständig neue Erkenntnisse sowie auch Problematiken, welche eine Lösung erfordern. Sei es mangelnde Stabilität unter hoher Belastung, zu viel Reibung oder gängige Konstruktionsfehler.

Dank dem neuen Fertigungsverfahren des 3D-Drucks ist es möglich Versuchsaufbauten in einem CAD-Programm zu entwerfen, diese folglich zu drucken und dementsprechend auf ihr Verhalten zu testen.

Ein CAD-Programm ist eine Software, welche zur Erstellung von zweidimensionalen oder auch dreidimensionalen Komponenten Modellen dient, CAD steht für Computer-Aided Design(engl.) und bedeutet übersetzt „Computerunterstütztes Entwerfen“. Neben dem reinen Entwerfen von Modellen haben solche Programme auch die Eigenschaft, das Modell je nach Material auf Stabilität, maximal Belastung oder sonstige Verhaltensmuster zu testen.

Das 3D-Verfahren arbeitet mit den exportierten Modell-Dateien eines CAD-Programms und setzt diese in ein greifbares Modell um.

Dabei gibt es darunter mehrere Fertigungsmethoden, die von uns gewählte ist die gängigste, FDM (Fused Deposition Modeling), dieses Verfahren arbeitet hauptsächlich mit Kunststoffen. Dabei zeichnet ein Extruder mit eingeführtem Filament Sichtweise ein Modell auf eine Druckplatte und formt damit ein reales Objekt. Kunststoffe, die für dieses Fertigungsverfahren verwendet werden ist PLA, ABS, PETG und weitere.

Die Vorteile des FDM-Drucks sind, dass die Druckgeräte sowie das Filament billig zu erhalten sind. Aber auch die das Verfahren einfach und nicht besonders komplex ist, sowie auch schnell.

In der initialen Entwicklungsphase des Projekts „Bionic-Hand“ kommen zwei private 3D, Drucker zum Einsatz, welche das FDM-Verfahren nutzen. Der sich daraus erschließende Vorteil ist eine effektive und besonders effektive Prototypenherstellung. Von großer Bedeutung ist die

Möglichkeit, jederzeit ein Modell in den Drucker zu laden und damit innerhalb weniger Stunden neues Versuchsobjekt zu erhalten.

Im Laufe der ersten Versuchsaufbauten war die Druckqualität nicht sehr relevant, denn der Fokus lag auf der Zeitersparnis und hauptsächlich auf dem Gewinn von Erkenntnissen, sowie auf dem Verständnis der in der CAD-Software entworfenen Mechanismen.

(Quelle: Zum zitieren und Quellenverzeichnis: <https://filamentworld.de/3d-druckverfahren/>)

### 6.2.2 Versuchsaufbauten

In diesem Versuchsaufbau war es das Ziel mit einem simplen Testmodell Erkenntnisse über den Mechanismus zu erhalten und dadurch mehr aus ihm zu lernen. Dafür wurde in der CAD Software Fusion360 ein erstes Testmodell entworfen.

#### 6.2.2.1 Aufbau

Das Modell besteht aus den drei Gliedern eines Fingers, einer Halterung für einen Servomotor, sowie aus den im Inneren vorhandenen Führungen.

Zwischen jedem Fingerglied befindet sich 1cm Abstand, verbunden sind diese durch eine 3mm dicke Fläche. Die Fläche hat den Zweck sich bei Zug an der Schnur zu biegen, durch die Führungen wird nur eine Biegung an der „Fläche“ ermöglicht. Befestigt wird die Schnur an einer Halterung am Servomotor und wir dadurch bei Rotation des Servos entsprechend angezogen. Die Fingerglieder haben eine nach außen gewölbter Form und sind in Richtung der Fläche verjüngt, dies ermöglicht dem Finger sich auf noch engeren Raum zusammenzuziehen.

#### 6.2.2.2 Versuchsresultat

Das Versuchsresultat war nahrhaft und bot einen breiten Einblick in den Mechanismus. Dabei gab es zahlreiche Erkenntnisse über Reibung, Stabilität und mechanische Abnutzung sowie aber auch Belastung.

Einer der Erkenntnisse war es, dass die Schnur eine gewisse Zugfestigkeit benötigt. Getestet wurden dabei mehrere Schnüre unter anderem auch Nylonschnüre. Als besonders stabil haben sich dann Angelschnüre bewiesen. Festgestellt wurde ebenso, dass ein mechanischer Widerstand benötigt wird, welcher den Finger wieder in seine Ursprungsposition: ausgestreckt – versetzt. Anfangs war dafür die freie Fläche vorgesehen, diese hat sich jedoch über die Zeit abgenutzt und hat den Finger nicht in die Ursprungsposition befördert. Eine mögliche Lösung während Gummikordeln, elastische Schnüre, welche am „Fingerrücken“ platziert werden und sich bei Biegen des Fingers spannen. Desto kürzer die Zugschnur im Finger, desto länger die Gummikordel.

### 6.2.3 Erste Testhand

#### 6.2.3.1 Konzept Einführung

Die erste Testhand zielt darauf ab ein Modell welches durchgängig erweitert werden kann zu schaffen. Mit Rücksicht auf etwaige Testergebnisse des Versuchsresultats (siehe Absatz 6.2.2.2) soll ein Modell geschaffen werden, welches den im Versuch aufgetretenen Problematiken entgegenwirkt. Unter anderem der nicht vorhandene Rückzug, eine Schnur für den Zug mit ausreichender Reißfestigkeit sowie eine anwendungsgemäße Konstruktion des Fingers.

Eine konkrete Lösung, um für mehr Sicherheit im Zusammenhang mit dem Zugsystem zu sorgen wurde für diesen Aufbau eine Angelschnur mit einem Durchmesser von XXmm gewählt, diese hat eine Kraft von XX/m. Aufgrund der XXX ist die Schnur ebenso schnittfest.

Im Versuch (Verweis) hat sich erwiesen, dass eine Fläche aus PLA welche die Fingerglieder verbindet nicht genügend Kraft aufbringt, um den Finger wieder in seine Ursprungsposition zu bewegen. Demnach wird eine Kraft benötigt, welche ständig dem Zug der Angelschnur entgegenwirkt und den Finger entsprechend austreckt.

In diesem Modell wird daher ein Elastomer, ein elastisches Material in Form eines Bandes mit ausreichender Kraft gewählt.

#### 6.2.3.2 Realisierung und Zusammenbau

Das Konzept wird anhand einer Testhand realisiert, diese setzt sich im Wesentlichen aus zwei Komponenten zusammen: dem Motorblock, welcher für die Antriebsmechanik zuständig ist, und der Roboterhand, welche die eigentliche die Bewegungen der Motoren in der Bionic Hand umsetzen.

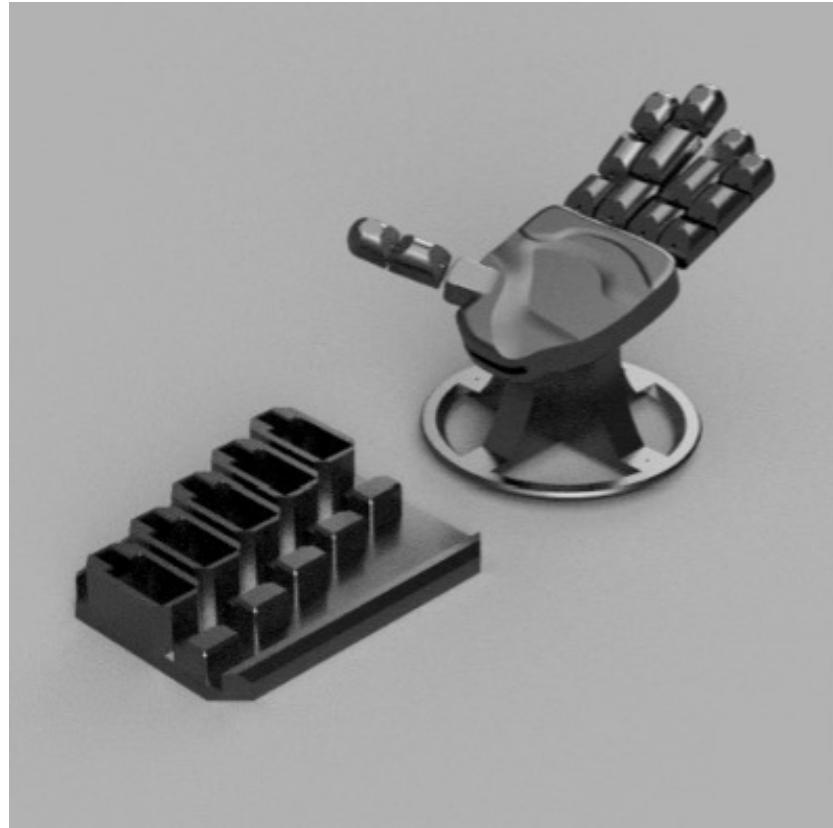


Abbildung 14: ergonerg

Das komplexeste Modell ist die Roboterhand, die aus mehreren Elementen besteht, darunter die Handfläche sowie die einzelnen Finger, die sich jeweils in drei Glieder unterteilen lassen. Der Mechanismus in jedem Finger erfüllt dieselbe Funktion: Durch ziehen an der Schnur verkürzt sich ihre Länge, was dazu führt, dass sich der jeweilige Finger zusammenzieht. Im Detail betrachtet beschränkt sich der Mechanismus des Fingers nicht nur auf das bloße Ziehen an ihm. Der Finger unterteilt in weitere Elemente. Er besteht aus drei Gliedern, von denen jedes eine Abschrägung an beiden Kanten, die zum nächsten Glied ausgerichtet sind, aufweist. Durch die Verjüngung der Kanten, kann beim sogenannten Falten des Fingers dafür gesorgt werden, dass sich der Finger auf kleineren Raum zusammenziehen oder auch falten kann.

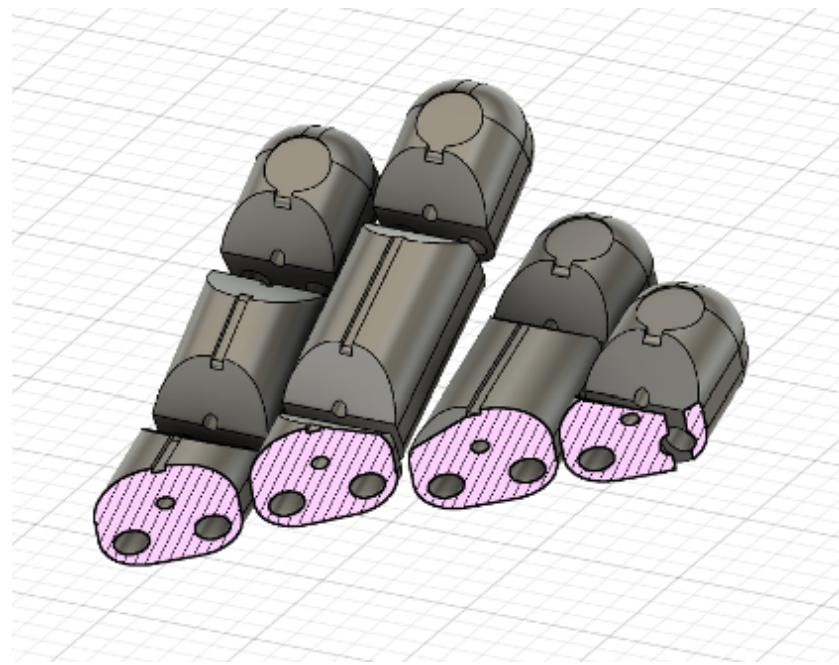


Abbildung 15: ergonerg

Wie in dieser Grafik ersichtlich wird hier der Querschnitt aller Finger dargestellt, dabei ist es nicht von Relevanz welches der drei Fingerglieder hier genommen wird. Alle Glieder, das unterste, das mittlere sowie das oberste Glied haben eine gemeinsame Eigenschaft: Durch jeden Finger verlaufen drei Tunnel. Der oberste Tunnel hat den Zweck einen Durchgang für die Angelschnur zu schaffen, an dieser wird gezogen und bewirkt das Falten des Fingers.

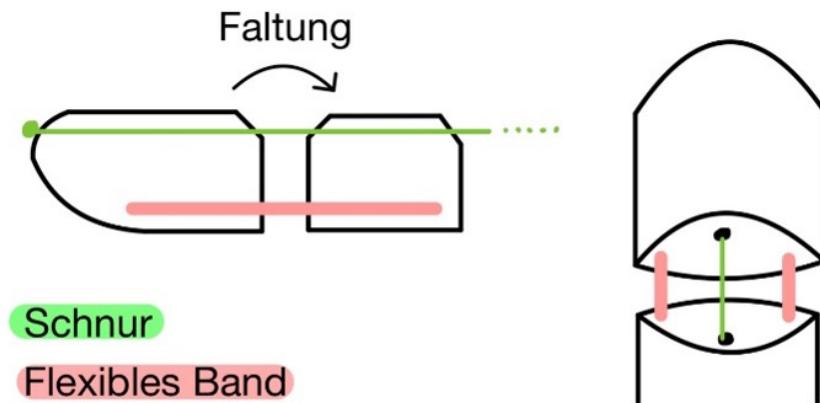


Abbildung 16: Fingerskizze

In den beiden unteren Tunneln oder auch Kanälen, welche einen ersichtlich größeren Durchmesser von XX mm besitzen verläuft das Elastomer, ein elastisches Band. Durch Falten des Fingers, welcher durch Zug an der Angelschnur ausgelöst wird, verlängert sich die Strecke auf, die sich das elastische Band spannen muss. Dies löst gegengleich zum Zug eine gegenwirkende Spannung aus. Anhand der durch das Band entstandenen Zugkraft, behält der Finger seine Position, er wird somit gesichert, um weiter in Richtung Zug der Angelschnur zu kippen und wiederum von der Angelschnur, um in Richtung des Bandes zu kippen. Neben der Funktion des Rückzuges verbinden die beiden Bänder alle aneinander liegenden Fingerglieder sowie auch das unterste Glied mit dem Handflächenelement. Dabei spielt auch die Position der Tunnel für das Band eine wesentliche Rolle, denn diese verhindern durch ihre Parallele Anordnung, an den unteren äußeren Seiten des Gliedes, dass sich die Glieder nach rechts oder links neigen. Ebenso befindet sich an dem inneren Teil des Gliedes, welches die Fingerkuppe nachstellt, beim Fingerabdruck eine runde Einkerbung sowie ein rechteckiger Kanal. Diese Eigenschaft des Modells hält die Option eines optionalen Kraftsensors frei. Mit so einem Sensor könnte, bei Integration der Funktion auch die Druckkraft gemessen werden.

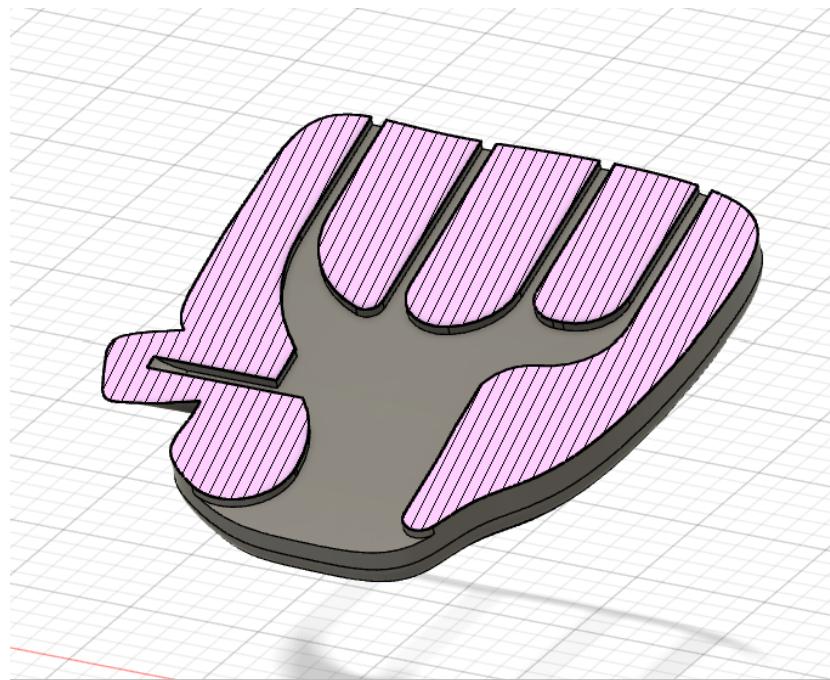


Abbildung 17: ergonerg

Verbunden sind alle Finger mit dem Element der Handfläche, dieses Element fungiert lediglich als Verbindungskomponente zwischen allen Fingern und dem Daumen. Im Querschnitt der Handfläche lässt sich erkennen, dass diese von innen einen Hohlraum beinhaltet. Dieser Hohlraum hat jeweils einen Kanal zu jedem der Finger, sowie Daumen. Dort soll später die Angelschnur durchgeführt werden. Nahe dem Handgelenk befindet sich eine große Öffnung, diese ist für alle fünf Schnüre gedacht und hat ausreichend Größe für die Schnüre und evtl. auch Kabel für die optionalen Kraftsensoren.

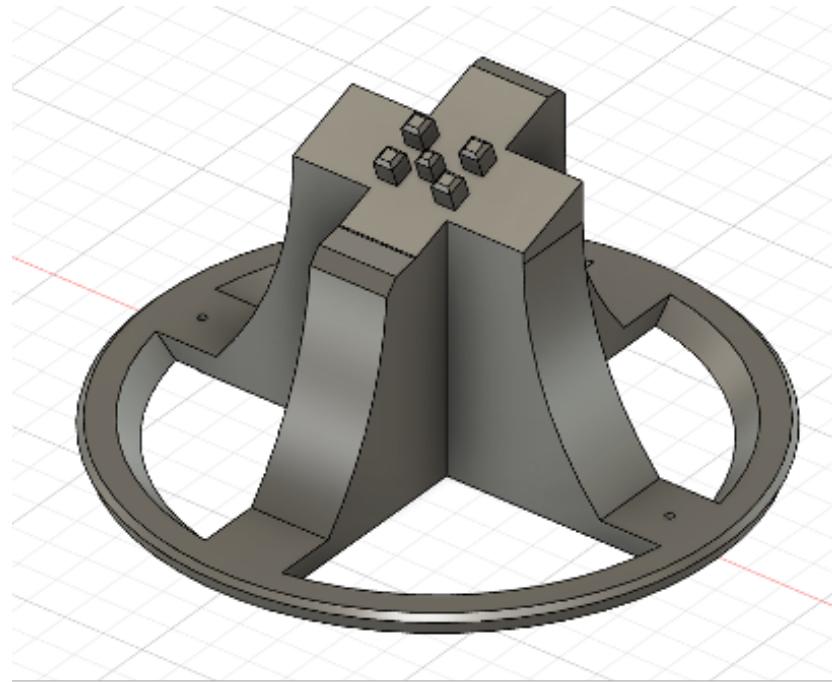


Abbildung 18: ergonerg

Weiters wurde eine Halterung für die Roboterhand modelliert, diese soll mit der Roboterhand verbunden werden. Ihr Zweck ist, die Hand in eine erhöhte und leicht geneigte Position zu bringen. Durch das Neigen der Hand werden die fünf Angelschnüre bereits in Richtung Antriebsmechanik gerichtet, auch wird bewirkt, dass die Hand auf ihrer Höhe nicht unter dem Drehkopf des Servomotors positioniert wird. Für die Montage der gesamten Roboterhand auf der Halterung, wird eine Aufsteckvorrichtung verwendet. Auf dem Gerüst sind fünf Pins mit einer Abschrägung und diese können in die Roboterhand mit Einkerbungen aufgesteckt werden.

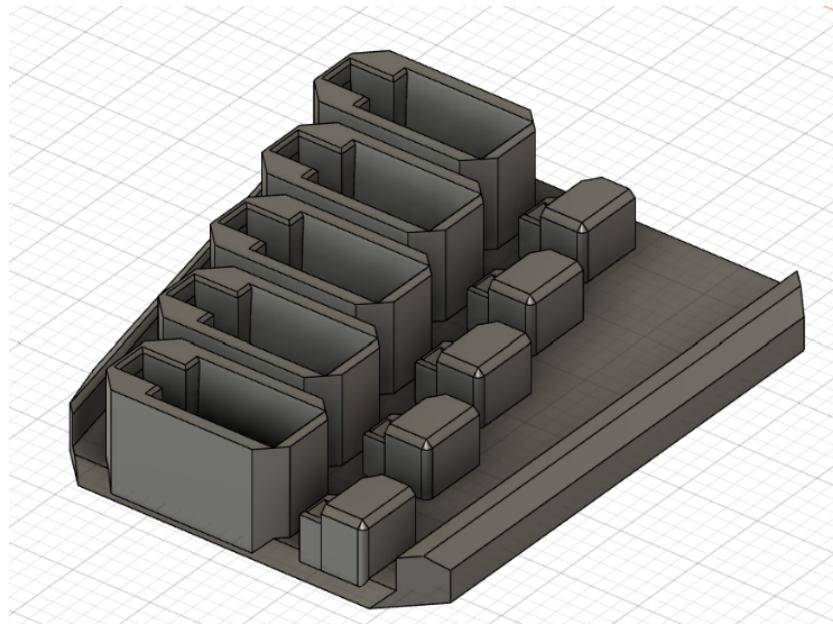


Abbildung 19: ergonerg

Letztlich beinhaltet das Modell der Roboterhand ein Element welches hier als Motorblock bezeichnet wird, dieses Element trägt diesen Namen da es alle für die Antriebsmechanik relevanten Komponenten beinhaltet.

Das Modell setzt sich aus einer Grundplatte, der fünf Halterungen für die Motoren und jeweils einer passend angeordneten Umlenkung zusammen.

Die Halterungen besitzen eine Einkerbung in die der Servomotor eingesteckt werden soll, parallel dazu befindet sich eine 90° Umlenkung welche dazu dient, dass falls seitlich von unten die Schnur eingeführt wird diese nach oben zum Servomotor geführt wird.

### Zusammenbau

Wie bereits erwähnt soll dieser Prototyp mit dem FDM-Druckverfahren gefertigt werden. Für den ersten Versuch wurden dabei ausschließlich die Glieder des Zeigefingers für den Druck gewählt. Die drei Fingerglieder, wurden mit einem Band und der Schnur befestigt. Der erste Versuch, indem ausschließlich der Faltmechanismus und sein Verhalten bei Zug am Finger in Erfahrung gebracht werden hat weitlaufende Erkenntnisse geliefert. Diese haben zu einem frühzeitigen Abbruch weiterer Tests geführt. Die sich dabei erschlossenen Problematiken und Erkenntnisse werden im folgenden Kapitel (6.2.2.2 Fazit und Ergebnisse) erläutert.

#### 6.2.3.3 Fazit und Ergebnisse

Der Grundstein für eine bewegliche Hand, welche Finger mit einem Faltmechanismus inkludiert wurde mit diesem ersten Entwurf bereits gelegt. Dennoch haben sich in der Realität anhand eines

einzelnen Versuchs entsprechend viele unbekannte Problematiken ergeben die zu einem Verwurf des Konzepts geführt haben.

Die entstandenen Problematiken:

- Da die einzelnen Fingerglieder, bei ihren Verbindungen aneinander aufliegen und sich beim Falten mehr oder weniger aneinander abrollen entsteht zusätzlich mehr Reibung. Dies führt dazu, dass für den Mechanismus zusätzlich mehr Zugkraft benötigt wird.
- Demnach ist die Verbindung für immer jeweils zwei Glieder über zwei parallel versetzte Bänder nicht die effizienteste Lösung oder gar die zuverlässigste. Der gewünschte Effekt, dass sich der Finger einklappt, beziehungsweise faltet ist geschaffen, jedoch hat der Finger keine Stabilität. Würde man ihn nach hinten biegen wollen, obwohl er bereits zu 50% eingefaltet ist, hätte man ein Spiel, mit dem es möglich wäre ihn zu formen. Beim Anhängen einer Last direkt an der Fingerkuppe oder auf das mittlere Fingerglied, müsste somit ausschließlich die Zugschnur dafür sorgen, dass der Finger weiterhin seine Form beibehält. Sinnhafter wäre es somit die Fingerglieder so zu modellieren, dass sich der Finger nicht nach Außen falten oder in diesem Fall besser gesagt biegen lässt, sondern ausschließlich nach Innen. Weiters ist auch bei der Verbindung anzumerken, dass der Finger sich auch nach links oder rechts biegen kann. Diese Funktion ist jedoch nicht erwünscht und muss mechanisch blockiert werden, nur die Verwendung der Bänder ist zu schwach um dies zu verhindern.

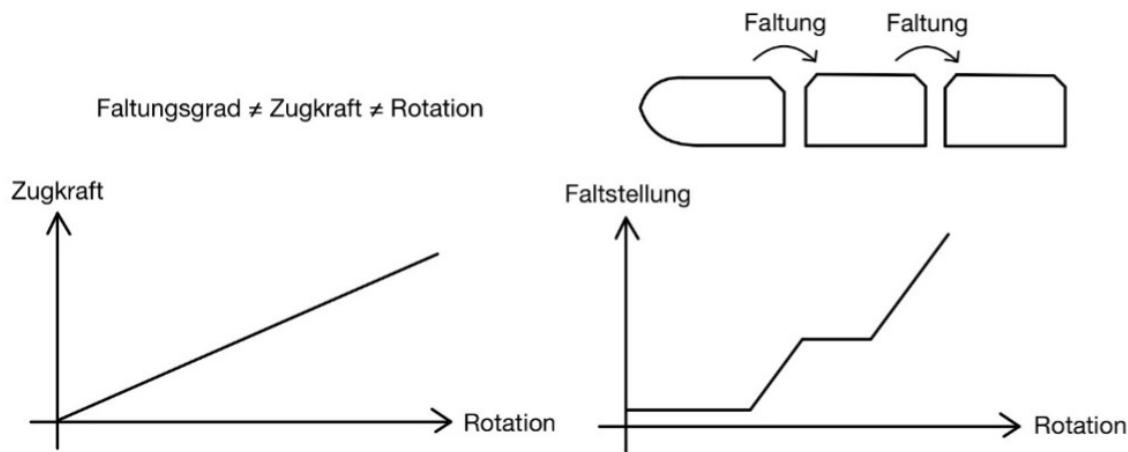


Abbildung 20: Fingerskizze

- Eine Fehlkonstruktion war wie bereits erwähnt die Fläche zwischen beiden Fingern, die zu einem zu mehr Reibung führt aber auch den einzelnen Glieder aufgrund ihrer flachen Fläche und kantigen Fläche nicht ermöglicht sich aneinander abzurollen. Dies führt dazu, dass sich der Finger bei ausreichend Zugkraft einklappt und nicht gleichmäßig abrollt. Wobei

dies von großer Bedeutung ist, da somit nicht mit linear ansteigender Zugkraft für eine gleichmäßige Faltung gesorgt werden kann.

- Ebenso führt auch der Durchgang, welcher für die Zugschnur ist zu einem Problem. An jedem Ausgang des Tunnels reibt die Schnur stark an den Rändern und dies führt zu mehr Reibung, sowie einem Schaden am Finger. Somit muss zusätzlich mehr Kraft angewendet werden.

## Fazit

Wie im Kapitel (6.2.2.2 Realisierung und Zusammenbau) erwähnt wurde aufgrund dieser Erkenntnisse der weitere Zusammenbau der Hand vorzeitig beendet und das Modell der Hand erstmalig verworfen.

Dennoch ist dieser Versuch äußerst lehrreich für die Entwicklung eines weiteren Prototypen, denn Reibung, Stabilität und die dadurch benötigte Kraft sind nun weitere Einflussfaktoren, auf diese in weiteren Designs mehr geachtet werden muss.

Auch hat sich gezeigt, dass der Faltmechanismus umsetzbar ist, jedoch ein Konstrukt erfordert, welches dem Finger ermöglicht sich in einem gleichmäßigen Verhältnis zur Zugkraft zu Falten.

### 6.2.4 Zweite Testhand (Inmoov)

#### 6.2.4.1 Einführung

Im Versuch der „ersten Testhand“ haben sich zahlreiche Problematiken und Erkenntnisse ergeben, demnach erfordert die Weiterentwicklung der Hand, die Unterstützung externer Fachkenntnisse. Nach Recherche im Internet, hat sich ergeben, dass ein französischer Bildhauer Gael Levin bereits 2012 für einen damaligen Kunden eine Handprothese entworfen hat und damit den Grundstein für sein für sein heutiges Projekt InMoov gelegt hat. InMoov ist der erste Open-Source 3D-Druckbare humanoide Roboter in 1:1 Größe und ist gedacht für Forschung an Universitäten, in Laboratorien und auch für private Zwecke.

Ein großer Vorteil des Projekts ist es, das bereits viele Erkenntnisse und Wissen in dieses eingeflossen sind. Ziel ist es dieses 3D druckbare Hand selbst zu realisieren, je nach Anforderung zu modifizieren und anhand dessen eine weitere Versuchshand zu schaffen.

#### 6.2.4.2 Mechanismus

Der Mechanismus der zweiten Versuchshand InMoov ist äußerst umfangreich und weist meist bessere Lösungen für Features der ersten Versuchshand aus Unterunterabschnitt 6.2.3 auf.

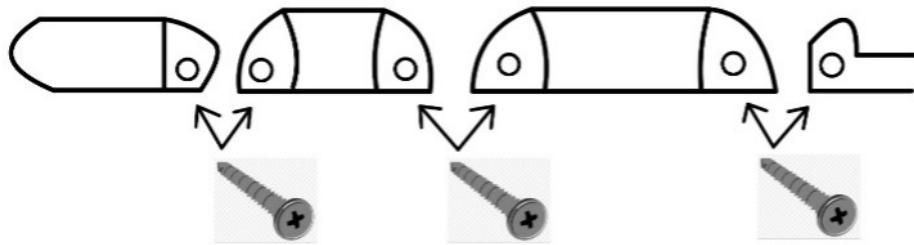


Abbildung 21: Fingerskizze

Die Problematik, dass wie beim vorherigen Modell die beiden Glieder aneinander reiben und sich nicht parallel zur Zugkraft kontinuierlich umrollen ist hier nicht vorhanden, da die Konstruktion von Grund auf andere Eigenschaften aufweist. Denn die jeweiligen Glieder werden hier nicht mit Bändern verbunden, sondern mit einer Schraube, welche die beiden Glieder, die ineinander gehen verbinden. Durch diese Verbindung sind die Finger zu einem stabil – können nicht nach links und rechts kippen und sich ausschließlich vom ausgestreckten Zustand nach innen Falten. Auch wird mechanisch ein Falten in die andere Richtung, verhindert.

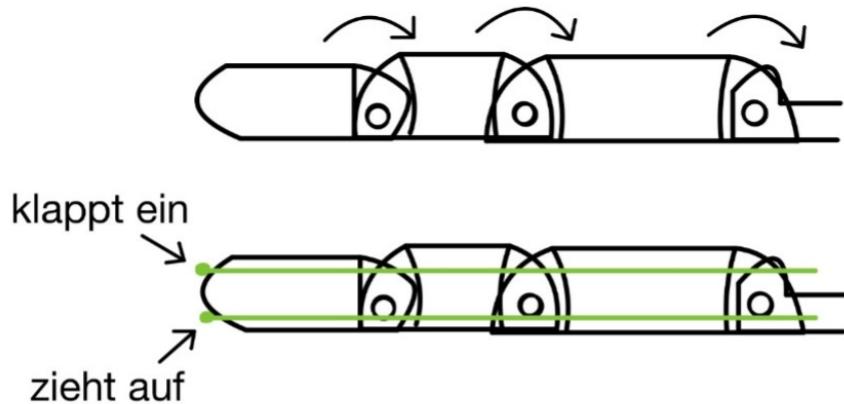


Abbildung 22: Fingerskizze

Am tatsächlichen Faltmechanismus hat sich bis auf einer Anpassung am Elastomer nicht viel geändert, denn der Rückzug erfolgt nicht mehr traditionell per elastischem Band. Diese Umsetzung des Faltmechanismus ist jedoch von großer Bedeutung, dar nur der Rückzug sowie der Faltung von ein und demselben Motor ausgelöst wird. Im Detail funktioniert der Mechanismus wie mit Hilfe der Grafik ersichtlich wie folgt: Am Servomotor wird eine runde Vorrichtung montiert, diese

hat an der Außenkante eine beidseitige Verjüngung, an dieser soll die Schnur entlanglaufen. Nun verlaufen beide Schnüre, die für den Ein- sowie Auszug verantwortlichen Schnüre, jeweils auf einer Seite, an dieser Runden Vorrichtung und sind an dem Ende dieser montiert. Dreht sich nun der Motor um  $90^\circ$  gibt er mehr der Schnur(1) her und verkürzt die Schnur(2). Bei diesem Vorgang müssen jedoch beide Schnüre so gespannt sein, dass keine der beiden nachgibt und sich bei Bewegung des Motors lockert.

Das Modell der Handfläche, an dem alle Finger und weitere Komponenten zentral zusammenkommen setzt sich selbst aus insgesamt vier tatsächlichen Elementen zusammen und zwei Hilfselementen. Das Hauptelement ist in dem Fall wie in der Grafik ersichtlich (1), dies ist das Größte und Zentrale Element, durch dieses Element verlaufen 15 Kanäle, die mittleren fünf dienen optionaler Sensorik. Die oberen fünf Kanäle sind für die Schnüre, welche dem Einfalten dienen, die oberen dienen der Schnüre für den Rückzug.

Um die Hand noch dynamischer zu gestalten besitzen der Ringfinger sowie der Kleinefinger ein zusätzliches Element (2) sowie (3), welches sich bei Zug auf wenige Grad mit faltet. Der Daumen hat eine ähnliche Funktion, Element (4) in der Grafik.

Als Hilfselement werden hier die Stifte (5) und (6) welche zur Verbindung zwischen Element (1) sowie den weiteren Elementen dienen.

Ein ebenso neuer und optionaler Mechanismus ist die des Handgelenks, dieser ermöglicht es der Hand um ihre eigene Achse zu rotieren. Umgesetzt wird dies mit einem Servomotor, welcher mit einer Übersetzung anhand zweier Zahnräder die Hand rotiert.

Montiert wird dieser Handgelenksmechanismus auf ein Unterarmkonstrukt, dieses besteht aus mehreren einzelnen Teilen um den Druck auf mit einem kleinen Drucker zu ermöglichen. Grundsätzlich sind alle Modelle des Projekts InMoov auf einem Drucker mit einer Druckfläche von  $12x12x12\text{cm}$  ausgelegt.

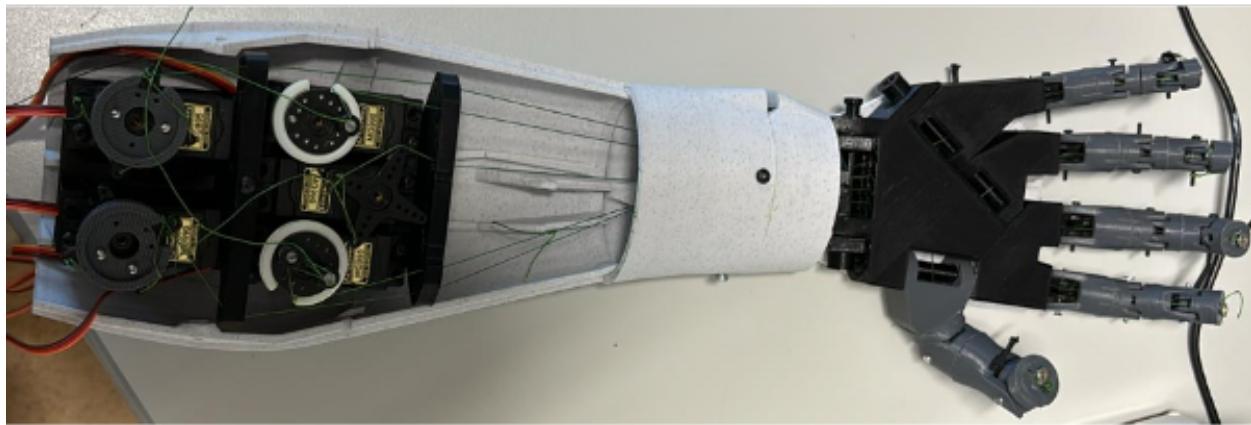


Abbildung 23: Roboterhand zweite Version

Im Unterarm befindet sich an der Innenseite eine Umlenkung, die die Schnüre über den

Rotationsmechanismus des Handgelenks lenkt. Dies verhindert vor allem eine Reibung zwischen der Schnüre und dem Servomotor für das Handgelenk, welcher in der idealen Strecke zwischen Schnur und Zielmotor steht.

Weiters werden daraufhin die Schnüre zum Motorblock der Antriebsmechanik für den Faltmechanismus der Finger geführt. In diesem Block befinden sich fünf Motoren, für die Finger. Dort wird der bereits beschriebene Mechanismus der beiden Schnüre umgesetzt.

Wurde ein Modell in einer CAD-Software wie beispielsweise Fusion 360 entworfen und soll nun in ein reales greifbares Objekt umgewandelt werden, so benötigt man eine sogenannte Slicer-Software. Die Bezeichnung „Slicer“ stammt dabei vom englischen Wort Slice (engl.) und bedeutet so viel wie Scheibe. Dies stammt der Tatsache ab, dass diese Software das 3D-Modell in einzelne Scheiben mit einer Sichthöhe geringer als 1mm teilt, denn wie bereits in Absatz 6.2.1.1 erklärt, geht das 3D-Druck Verfahren FDM so beim Schaffen eines realen Objekts vor.

Gewählt wird die Open-Source Software Ultimaker Cura, in der Version 5.6.0, in diesem kann man zahlreiche Optionen treffen: Die Sichthöhe der einzelnen Scheiben, in Fachsprache „Layers“, die Druckgeschwindigkeit, die Füllungsdichte, ob Stützstrukturen für den Druck vorhanden sein sollen oder nicht. Durch das Regeln dieser Optionen wird die Druckqualität beeinflusst, drückt der Drucker zu schnell so leidet die Qualität und es können Fehler im Druck auftauchen, ist die Schichthöhe zu hoch ist der Druck äußerst ungenau und weist eine unschöne Textur an seiner Oberfläche auf.

Auch können Faktoren, wie die Drucktemperatur oder das Filament gewählt werden. Je nach Filament gibt es bereits im Vorhinein konfigurierte Optionen.

Sind nun alle gewünschten Werte angepasst, kann „gesliced“ werden dabei wird nun von der Software das Objekt in einzelne Scheiben geteilt und ein g-code File erstellt, welches dem Drucker daraufhin Anweisungen gibt.

Ein Druck dauert in der Regel mehrere Stunden, dies variiert jedoch je nach Drucker – neuere schaffen das selbe Ergebnis in 20% der eigentlichen Druckzeit.

Nach einiger Zeit ist der Druck fertig und kann von der Druckplatte gelöst werden, vorhandene Stützstrukturen können im Bestfall einfach vom Druck gelöst werden.

Für den Zusammenbau der InMoov Hand werden neben dem Druck auch weitere Elemente benötigt, die nicht aus dem 3D-Drucker stammen. Als Auflistung die zusätzlich benötigten Komponenten:

- 16 Stück M3x20mm Schrauben
- 16 Stück M3-Mutter
- Superkleber
- Angelschnur: 3 Meter

### 6.2.4.3 Fazit und Ergebnisse

Der Versuchsaufbau hat nur kurzfristig und teilweise funktioniert. Für einen kurzen Moment konnten alle Finger, sowie der Daumen geöffnet und geschlossen werden. Der Faltmechanismus hat sich zwar bewiesen, jedoch mit einigen Unstimmigkeiten.

Während dem Testen sind zwei Punkte besonders auffällig gewesen: Zu einem die Stabilität mit und ohne Belastung, sowie die Spannung der jeweils beiden Schnüre. Unter die Stabilität fallen in diesem Fall, das gesamte Gerüst vom einzelnen Finger bis zum Motorblock, die Antriebsmechanik und ihre Halterung für die Umwicklung. Sowie die Schnüre, welche einige Probleme und demnach auch Schäden verursacht haben.

### **Problematik-1: Ünstimmigkeiten bei dem Umwickeln der jeweils beiden Schnüre”**

Wie in Absatz 6.2.4.2 erläutert, sollen sich beide Schnüre um eine runde Halterung wickeln und dabei je nach Drehung den verbundenen Finger ein -oder auffalten. Damit dieser Mechanismus funktioniert müssen beide Schnüre ständig unter Spannung stehen, da sonst die Gegenwirkung des Systems nicht vorhanden ist und der Finger Überklappen kann.

Es gab mehrere Lösungsversuche, zu einem den Servomotor in 5° Schritten von 0 bis 180° Grad approximieren und dabei während jedem Schritt die Schnüre nach zu spannen, jedoch blieb auch dies erfolglos.

### **Problematik-2: Instabilität der Halterungen auf dem Motor”**

Die Angelschnur hat neben ihren Vorteilen auch einen trivialen Nachteil, denn sie ist sehr dünn und daraus folglich unter Spannung äußerst schneidend. Stand die Schnur unter zu hoher Spannung hat dies dazu geführt, dass diese sich selbst aufgrund der zu hohen Reibung durch die Halterung am Motor geschnitten hat und ihn dabei meist auch horizontal zerteilt. Nach Erkenntnis dieses Problems gab es drei grundlegende Lösungsversuch:

- **Lösungsversuch 1:**

Hier wurde versucht das gedruckte Objekt selbst härter und dicker zu drucken. Umgesetzt wurde das durch Erhöhen der Fülldichte von 25% auf 65%, dadurch ist das innere Modell von Innen dichter ist. Weiters wurde auch die Wandstärke und -dicke erhöht.

Das Ergebnis war ernüchternd, bereits nach neuem Anspannen führte es bereits wieder zum selben Problem und die Halterung hielt nicht stand.

- **Lösungsversuch 2:**

Nach genauerem Hinblick wurde klar, es liegt am Druckverfahren, denn wie bereits erklärt drückt der Drucker in einzelnen Schichten. Das Problem: Die Schnur liegt parallel zu den einzelnen Schichten und genau zwischen diesen Schichten liegen die Schwachpunkte des Modells. Somit wurde die Entscheidung getroffen das Modell so zu drucken, dass die Schnur und die einzelnen Schichten nicht parallel zueinander liegen, bewirkt wurde dies indem die Halterung in einem 45°Winkel gedruckt wurde.

Der zweite Lösungsversuch ist somit erfolgreich und die Schnur schneidet oder halbiert nicht mehr die Halterung. Jedoch nicht perfekt, denn die Schnur reibt an der gerippten Oberfläche der Halterung. In diesem Fall muss ein anderes Material als PLA und damit auch ein anderes 3D-Druck Verfahren verwendet werden. Ergeben hat sich der SLS (engl. Selective Laser Sintering). Dies ist ebenso eine additive Fertigungstechnologie, dabei verschmilzt ein Laser selektiv Pulverpartikel und das schichtweise um daraus dann das gewünschte 3D-Objekt zu fertigen. Finalerweise entstanden

dann für das Projekt Halterungen aus dem Material Nylon mit Glasfaserverstärkung.

### **Problematik-3:**

Zwar sind Angelschnüre sehr stabil und stark, tendieren jedoch unter ständiger Reibung trotz ihrer acht fachen Flechtung mit der Zeit zu reißen. Die Strecke von Fingerkoppe, bis zum Motor beträgt dabei je nach Finger zwischen 20-30cm, auf dieser Strecke reibt sich die Schnur an mehreren Stellen des Modells auf. Da dieses Problem erst mit der Zeit auftritt und somit mit der Hand dennoch ausgiebig getestet werden kann bis zum nächsten Riss wurde dieses Problem im Versuchsaufbau der zweiten Hand noch nicht behandelt.

### **Problematik-4:**

Ein weiteres Problem ist die Befestigung der Schnur am Fingerende, der Fingerkoppe. Diese hat zwei Bohrungen bzw. Löcher, zwischen diesen werden beide Schnüre befestigt. Das Problem, da diese Schnüre so klein sind und dauerhaft unter Spannung stehen reißt die Fläche zwischen den Löchern meist ein und die Schnüre können nicht mehr befestigt werden.

Das Modell löst einige Probleme und schafft neue Ansätze. Der Faltmechanismus wurde grundlegend erweitert, der Rückzug wird nun nicht mehr per Elastomer, sondern mit einer zweiten Schnur bewirkt. Die einzelnen Fingerglieder, sind tatsächlich mechanisch durch eine Schraube miteinander verbunden.

All dies führt zu präziseren und genaueren Bewegungen, der Faltmechanismus erfolgt annähernd linear zur Zugkraft. Trotzdem hat sich herausgestellt, wie brüchig das Modell ist, besonders an den Fingern. Reibung ist weiterhin ein großes Thema und wird durch die zweite Schnur im Faltmechanismus noch trivialer.

Ziel ist es im dritten Versuchsmodell die Problematik mit der Reibung zu verringern und daraus folglich auch ein stabileres Modell zu ermöglichen.

## 6.2.5 Dritte Testhand (Projekt-Silikonhand)

### 6.2.5.1 Das Konzept

Nachdem die vorherigen Versuche erwiesen, haben welche Problematiken sich aus einem festen Modell aus PLA ergeben, stand ein großes Umdenken im Raum. Auf Vorschlag einer Betreuungskraft stellte sich heraus, dass das Modell elastischer sein muss und ein hartes Material nicht immer die Beste Lösung ist. Der Vorschlag war Silikon in das Modell der Hand zu integrieren, ein ähnliches Projekt habe so bereits eine Hand realisiert. Ähnlich wie bei dem Modell aus Unterunterabschnitt 6.2.3 bestanden dort die einzelnen Glieder aus einzelnen Teilen, zwar aus Silikon, waren jedoch auch per Elastomer verbunden. Die Handfläche bestand bei dem Modell aus dem Vorschlag dennoch aus harten Kunststoff.

Für das Projekt "Bionic-Hand" gab sich eine neue Ebene und Vielfalt der Möglichkeiten. Die neue Idee ist es die gesamte Hand vollständig aus Silikon zu entwickeln und somit Problemen wie der Reibung und Brüchen unter starker Last aus dem Weg zu gehen.

Genauer lässt sich das neue Konzept wie folgt beschreiben: Die gesamte Hand besteht aus

dem Material Silikon, seien es die einzelnen Finger, der Daumen und die Handfläche. Daraus ergeben sich mehrere Vorteile. Zu einem ein Grip (engl.), aufgrund des Silikons hat die Hand beim Greifen mehr Griff und Objekte können gehalten werden, ohne aus der Hand zu rutschen. Der nächste Vorteil ist, dass die Hand anpassbarer wird und sich aufgrund ihrer elastischen Struktur genau um das Objekt stülpen kann, dies ist eine große Verbesserung im Vergleich zu der festen Hand aus hartem Kunststoff.

Ein Hand aus Silikon springt von selbst wieder in ihre Ursprungsposition zurück, dies liegt an den Eigenschaften des Materials, dadurch wird sich ein externer Elastomer erspart, der Rückzug erledigt sich von selbst. Die Fingerglieder der Hand sollen ähnlich wie bei den vorherigen Modellen eine Verjüngung zueinander haben und durch Zug an der Schnur der Finger sich Falten. Reibung dürfte nach Überlegungen nicht mehr existieren, da die Schnur bei Bewegung das Material eher bewegt und nicht wirklich reibt. Befestigt wird dies dann an einer Vorrichtung am Handgelenk.

### 6.2.5.2 Gussform

Eine Herausforderung von diesem Versuchsmodell ist das Verfahren des Silikongusses, um das Verfahren des Silikon gießen zu verstehen wird eine Gussform in der CAD-Software entworfen.

Als Testobjekt wurde ein einfacher Finger, welcher einen Faltmechanismus ermöglicht entworfen.

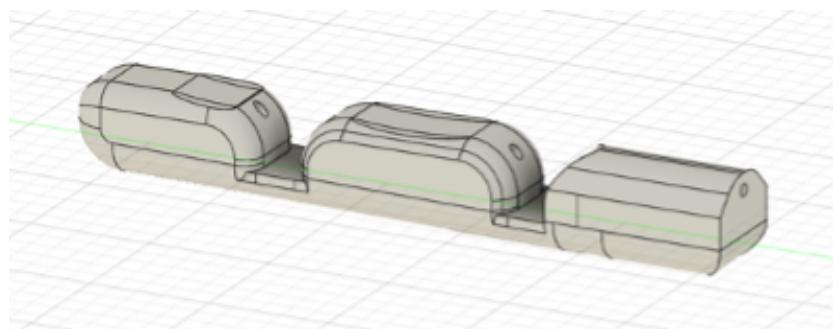


Abbildung 24: Silikon-Testfinger

Im oberen Teil des Fingers verläuft durch jedes einzelne Glied ein Kanal, für die Zugschnur, die Verbindungen zwischen den einzelnen Gliedern wird durch eine Fläche geschaffen. Bei Zug faltet sich dann der Finger an den dünneren Punkten des Objekts.

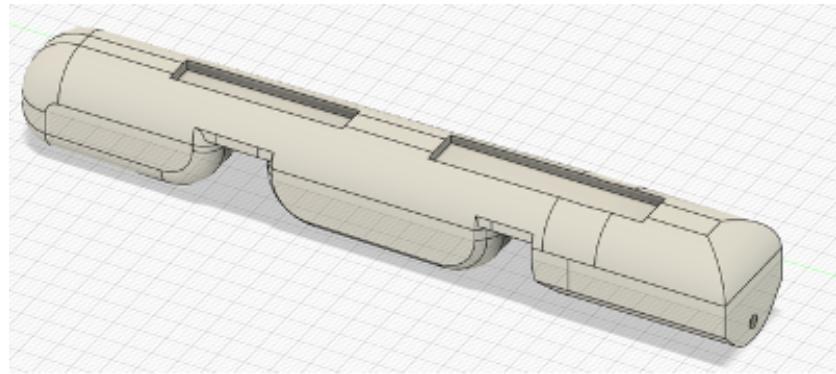


Abbildung 25: Silikon-Testfinger

Um diese Fläche optional nochmals im Nachhinein zu verdicken und dadurch zu beeinflussen, wie stark der Rückzug ist wurden zwei Einkerbungen in das Modell des Fingers implementiert. Durch das Aufgießen mit Silikon dieser rechteckigen Einkerbungen an den Biegungspunkten, kann dies bewirkt werden.

Der ganze Finger wird in eine Gussform gegossen, diese besteht aus zwei Teilen um den Finger leicht aus der Form lösen zu können, ohne dabei die beiden Objekte zu beschädigen.

Verbunden sind beide Teile der Form mit einem Steckmechanismus, wie in der Grafik (x.x.x) ersichtlich. Am oberen Teil der Gussform befindet sich ein Loch in dieses wird das flüssige Silikon später gegossen, das Loch wurde extra groß dimensioniert damit gleichzeitig auch die Luft im Innenraum entweichen kann.

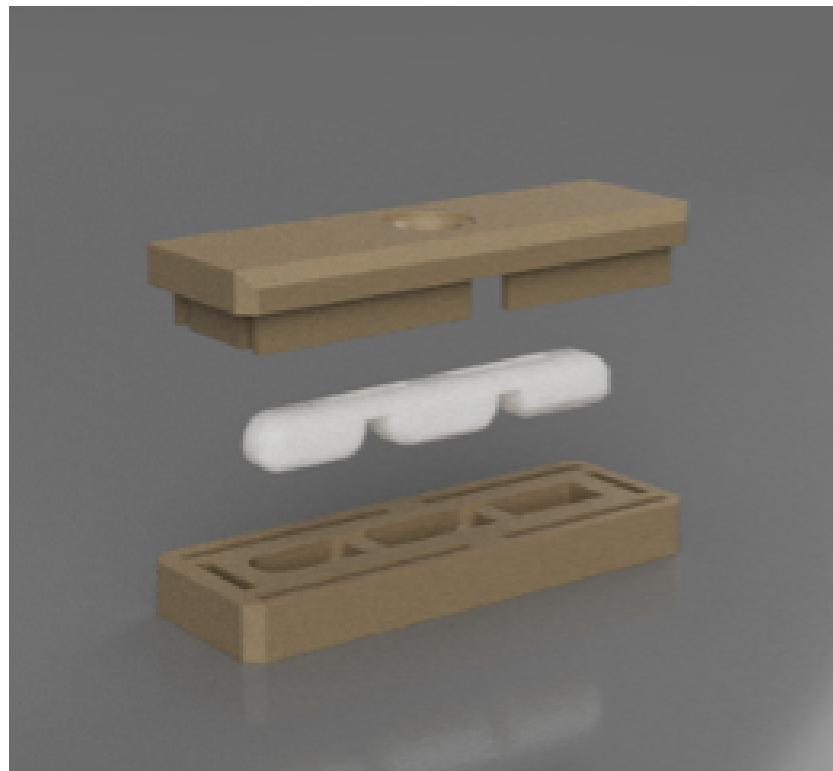


Abbildung 26: Silikon-Testfinger-Gussform

Quelle: <https://silikon-profis.de/2-Komponenten-Massen#:~:text=2%2D%20Komponenten%20Silikonkautschuk>

Das hier verwendete flüssig Silikon besteht aus zwei Komponenten welche als Teil A und Teil B bezeichnet werden. Teil A ist die Hauptkomponente des Silikons und Teil B ist die Härtekomponente, zusammen vermischt führt dies zu einem festen elastischen Material.

Das Ergebnis nach dem ersten Guss: Während dem Entwurf des Gussform wurde nicht daran gedacht, dass das Loch für den Einguss auf derselben Höhe wie die höchste Fläche oder Decke des Innenraums sein muss. Durch diesen Fehler hat sich ab dieser Höhe ein Luftraum gebildet, welcher nicht mit Silikon gefüllt wurde.

**Verbesserung:**

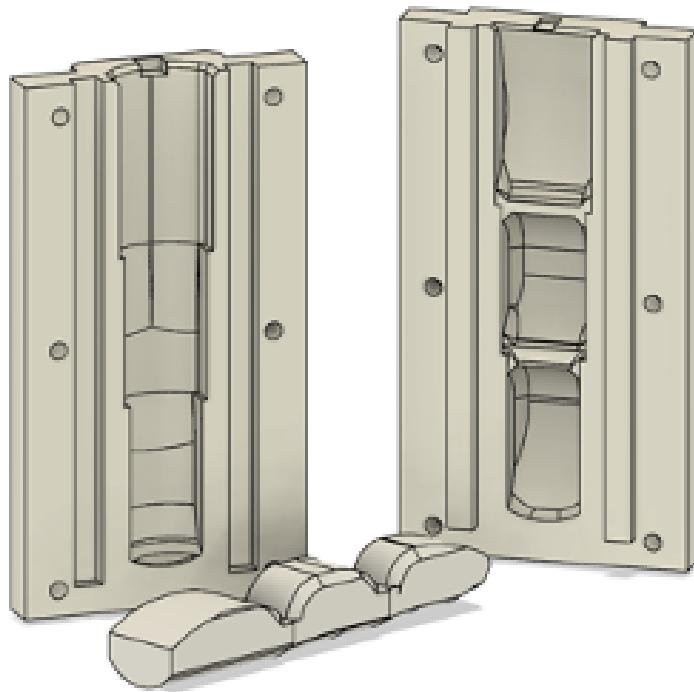


Abbildung 27: Silikon-Testfinger-Verbesserung

Dies wurde durch eine Form, in der der Finger aufrechtstehend gegossen werden soll verbessert. Das Loch ist somit am Anfang des Fingers und die Luft kann vollständig entweichen. Eine kleine Anpassung ist, dass neben dem Steckmechanismus auch eine Verschraubungsmöglichkeit geschaffen wurde um die beiden Teile der Gussform stärker aneinander zu pressen, dies verhindert das Austreten von Silikon während dem Guss.

Gezeigt hat sich nach dem Guss, dass das fertige Objekt aus Silikon, sich auch in andere Richtungen biegen lässt als ausschließlich in die Faltrichtung. Um alle anderen Richtung außer die gewünschte zu verhindern, benötigt der Finger einen Stabilisator von Innen, ähnlich wie der Finger des Menschen, welche einen Knochen hat.

### 6.2.5.3 Gelenke

Um zu verhindern, dass sich der Finger nach links, rechts, oder nach hinten biegt also nicht in die Richtung, in die er sich Falten soll wird im Inneren des Fingers etwas benötigt, dass nur das Biegen in die Faltrichtung erlaubt. Somit ein mechanisches Gelenk welches durch den gesamten Finger geht.

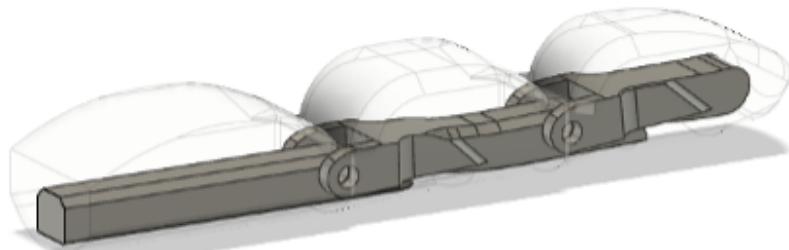


Abbildung 28: Fingergelenke

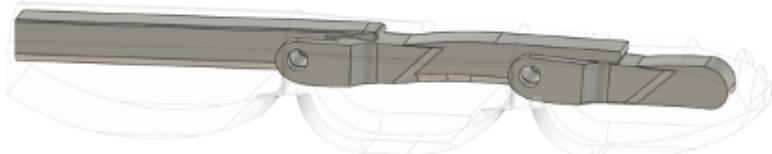


Abbildung 29: Fingergelenke

Das Gelenk besteht aus drei Komponenten, verbunden sind diese durch eine Schraube, exakt an dem Punkt an dem sich der Finger beim Falten biegt. Durch eine mechanische Blockade kann sich das Gelenk ausschließlich in Faltrichtung falten.

Mit einer Erweiterung wird das gesamte Modell in der Gussform befestigt und positioniert, nach dem Guss findet sich dann das Gelenk eingegossen im Finger.

#### **Ergebnis:**

Während des Gusses ist Silikon ausgetreten, obwohl bereits im Vorhinein vorgesorgt wurde, damit dies nicht passiert – wie die Verschraub Möglichkeiten, mit denen beide Teile aneinander gepresst werden sollen oder die Steckvorrichtung. Gelöst konnte dies mit Klebebändern, die zum Abdichten verwendet wurden.

Trotz der Komplikationen, hat sich der Versuch als erfolgreich herausgestellt, denn der Finger funktioniert in Zusammenspiel mit dem Gelenk wie erwartet.



Abbildung 30: Gelenke in echt

#### 6.2.5.4 Entwurf einer neuen Gussform

Nachdem sich gezeigt hat, dass ein einzelner Finger umsetzbar ist der nächste Schritt eine gesamte Hand zu entwerfen. Für diese wird somit ein vollständiges Gelenksgerüst für die gesamte Hand benötigt, ein Modell einer Hand sowie eine dafür entworfene Gussform.

Das Modell der Hand – Grundsätzlich ist Ziel die gesamte Hand aus Silikon zu verwirklichen und diese aus Silikon zu gießen. Der Faltmechanismus funktioniert dabei wie in Kapitel (6.2.4.2 Finger), mit der zusätzlichen Eigenschaft, dass die Finger früher in der Handfläche beginnen, damit sich diese besser um das gewünschte Objekt "Falten" kann.



Abbildung 31: 3D-Modell Silikonhand

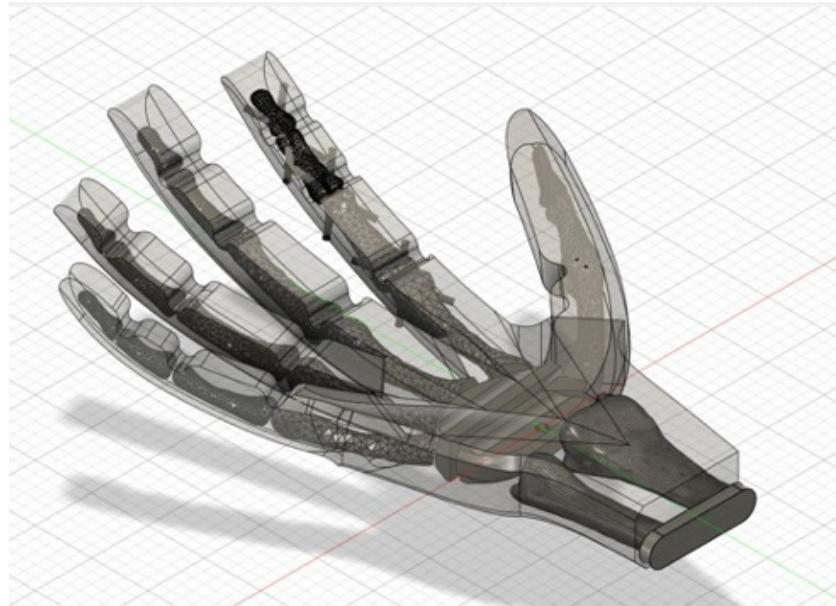


Abbildung 32: 3D-Modell Silikonhand

Ohne nun bereits auf das Gerüst einzugehen ist in dieser Grafik zu sehen, dass die Handfläche zwischen den Fingern kein Material hat. Dies schafft auch die optionale Möglichkeit die Finger mit einem weiteren Mechanismus seitlich zu bewegen.

In der Grafik ist ebenso die sich von den anderen Fingern unterscheidende Anordnung des Daumens, dieser liegt seitlich und soll beim Falten sich ähnlich wie der echte Daumen des Menschen verhalten. Das Gerüst hat dabei eine triviale Neuerung, denn es wurde sich an den tatsächlichen Knochen eines Menschen orientiert, damit soll für mehr Ähnlichkeit zur Anatomie der menschlichen Hand gesorgt werden.



Abbildung 33: Skelett

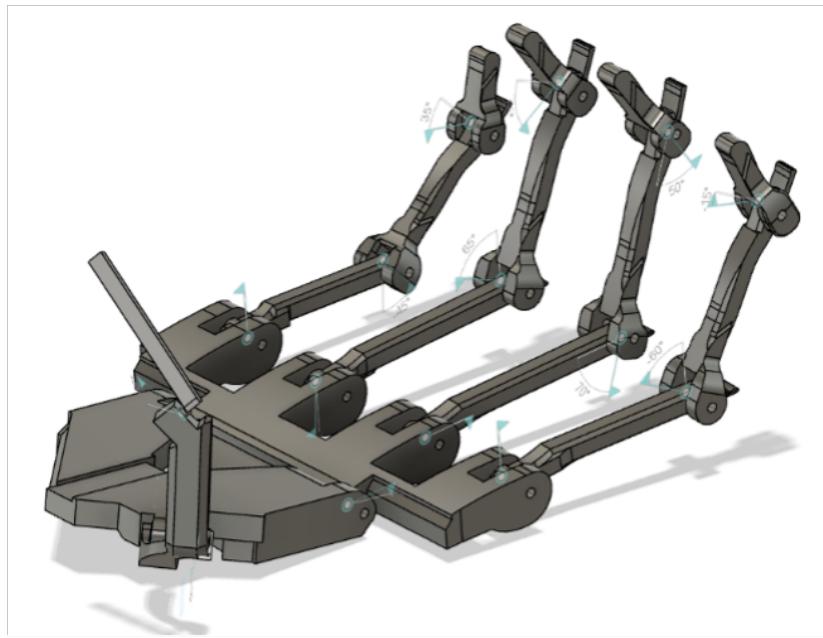


Abbildung 34: Skelett

Im Unterschied zu einer vorherigen Version des Gerüsts, ist zu erkennen, welche Vorteile die neue gegenüber ihr hat. Zu einem ist diese viel flexibler und hat keine große und starre Handwurzel. Ein weiter Unterschied ist die Verbindung der einzelnen Knochen in jedem Finger. Anders wie

bei der alten Version werden hier keine Schrauben zu der Verbindung einzelner Gelenksteile, sondern Gummikordeln verwendet.



Abbildung 35: Fingerglieder

Diese neue Form der Verbindung war notwendig um die einzelnen Elemente der Knochen platzsparend und effektiv zu verbinden. Zwar wurde kurzzeitig eine Verbindung anhand einer Schraube getestet, dies zeigte jedoch, dass das mit der Größe der Knochen und dem Raum im Finger nicht umsetzbar wäre. Zwar blockiert dieser Mechanismus nicht Bewegungen in alle anderen Richtungen außer die Faltrichtung, schafft jedoch mehr Flexibilität für den Finger – dennoch wurde jeder Finger so modelliert und an den Krümmungsstellen so verstärkt, dass er sich hauptsächlich in die Faltrichtung biegt.

#### 6.2.5.5 Gussform und Gussversuch

Für den Guss der gesamten Hand wird ähnlich wie bei dem Finger aus Versuch von Kapitel (x.x.x), eine Gussform benötigt, diese Gussform hat jedoch ein größeres Spektrum an Anforderungen. Mitunter muss berücksichtigt werden, dass das SSkeletäuch bekannt als das Gelenksgerüst präzise in der Mitte eines Fingers platziert werden muss.

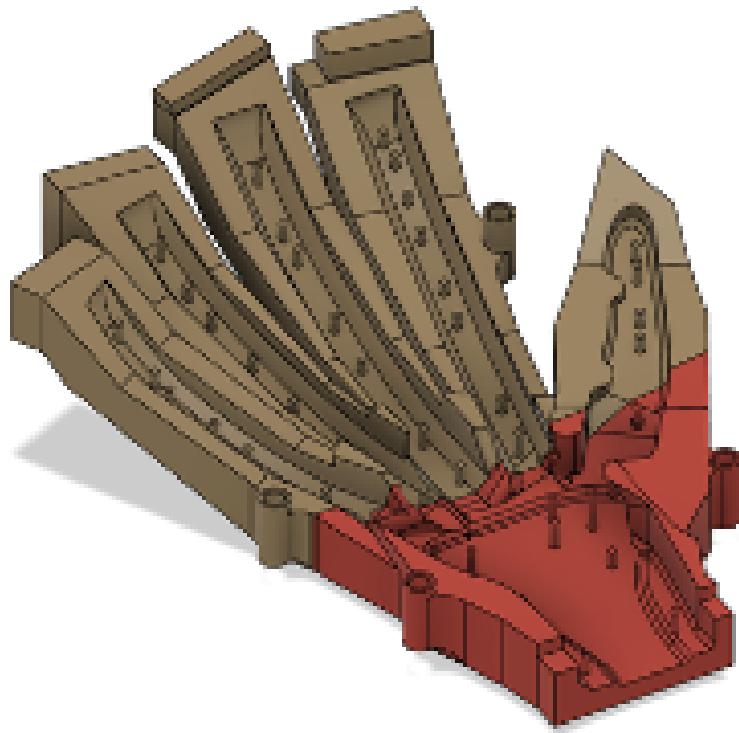


Abbildung 36: Gussform

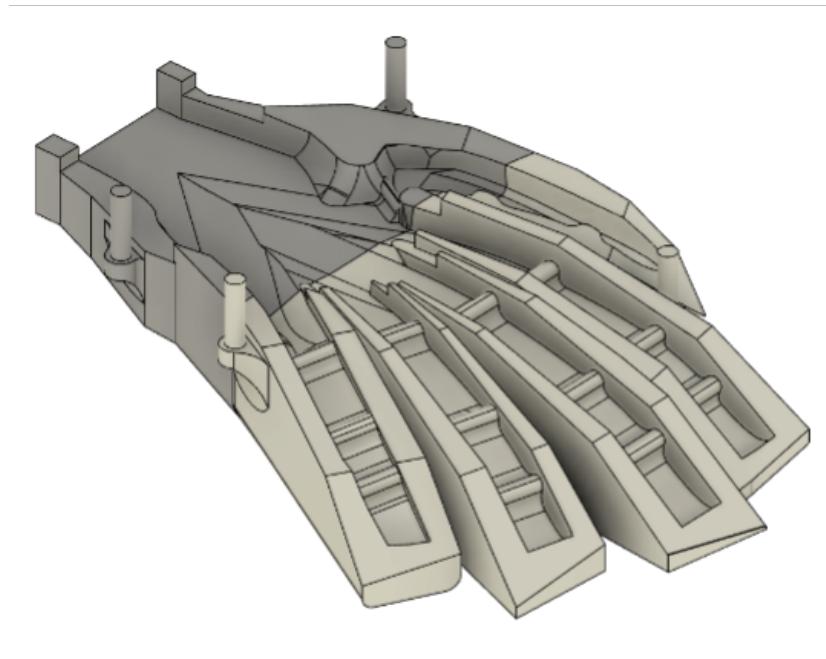


Abbildung 37: Gussform

Zu sehen sind in den beiden Grafiken, vier Teile der Gussform, dies ist darauf zurückzuführen, dass die Druckfläche des Druckers sonst überschritten werden würde. Daher muss jeweils der untere und der obere Teil der Form halbiert werden. Zu der unteren Hälfte der Gussform in der linken Grafik gehört das rote und braune Element und zu der oberen gehören wie in der rechten Grafik sichtbar das beige und graue Element. Befestigt werden die beiden Teile jeweils mit vier Stiften in Form einer Steckvorrichtung. Sichtbar ist ebenso zwischen dem unteren und dem oberen Element der Gussform eine weitere Steckvorrichtung, durch diese werden beide Hälften der Form verbunden.

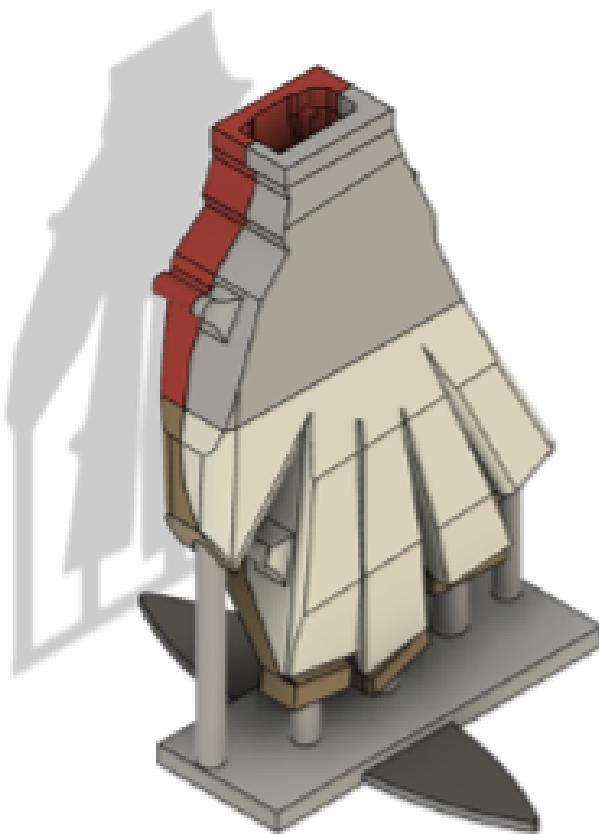


Abbildung 38: Gussform

Wie in dieser Veranschaulichung ersichtlich wird in aufrechtem Zustand der Gussform das Silikon in das obere Loch eingegossen. Wie in Absatz 6.2.5.2 an einem Versuch zu sehen ist, ist es von großer Bedeutung, dass das Eingussloch an der höchsten Stelle des Innenraums liegt. Um sicherzustellen, dass die Hand stets aufrecht bleibt und während dem Härteprozess des Silikons nicht gesichert in ihrer Position bleibt wurde eine Halterung entworfen, auf diese kann die Form gesteckt werden. Das Loch wurde möglichst groß modelliert, damit die im Innenraum verbleibende Luft der Gussform ausreichend entweichen kann.

### Der Guss:

Der Gussprozess ist nicht einwandfrei verlaufen, mitunter lag dies hauptsächlich an den Lücken zwischen den einzelnen Elementen. Aufgrund des FDM-Druckverfahren sind zwischen den einzelnen Flächen entweder Rillen oder versetzte Schichten und dies führt dazu, dass diese nicht exakt aufeinander Aufliegen. Dort fließt das Silikon dann meist aus der Gussform aus. Um diesem Problem entgegenzuwirken, ohne das Modell aus Kostengründen neu drucken zu müssen gab es folgende Lösungsversuche:

- Ein Versuch war es mit einem Dichtenden Klebeband, diese Freiräume von außen abzudichten. Dies hat kaum zu einer Verbesserung beigetragen. Das Band hat dem Druck vom Silikon meist nicht standgehalten und hatte keine ideale Haftung an den äußeren Wänden der Gussform.
- Weiters wurde versucht mit einer Dichtmasse abzudichten, doch selbst diese konnte das Modell nicht vollständig abdichten. Zwar wurde es besser, jedoch besonders an komplexeren Bereichen, wie bei dem Daumen gab es keine bemerkbare Verbesserung.

Letztlich konnte kein nutzbares Ergebnis mit der Gussform erzielt werden. Die Versuche das Modell abzudichten war nur teilweise erfolgreich. Wie an dieser Abbildung ersichtlich konnten die Finger zwar gegossen werden, jedoch konnte das Silikon nicht die „Handfläche“ halten ohne auszurinnen.



Abbildung 39: Silikonhand Fehlversuch

Jedoch gab es einen Notlösungsversuch um das vorhandene Material zu retten, dafür wurden

die Finger abgeschnitten bis zu dem Punkt, an dem der Guss nicht gelungen ist, daraufhin vermessen und ein Modell wurde modelliert. Dieses ersetzt die Handfläche aus Silikon mit einer 3D Gedruckten und auf diese können die bestehenden Silikonfinger auf eine Einkerbung für diese gesteckt werden. In der Abbildung ist der Unterschied, zwischen Silikon und Kunststoff anhand der Farben zu sehen, dunkelgrau steht für das 3D-Gedruckte Modell und die beige Farbe für das Silikon. Zusehen sind auch fünf Kanäle, durch diese soll die Angelschnur gehen und an den Fingern ziehen.

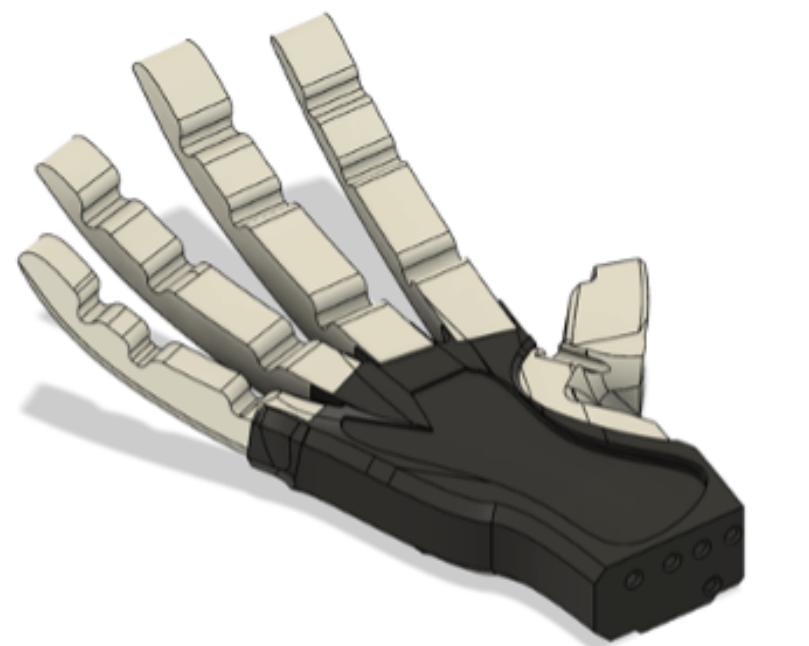


Abbildung 40: Silikonhand Fehlversuch 2

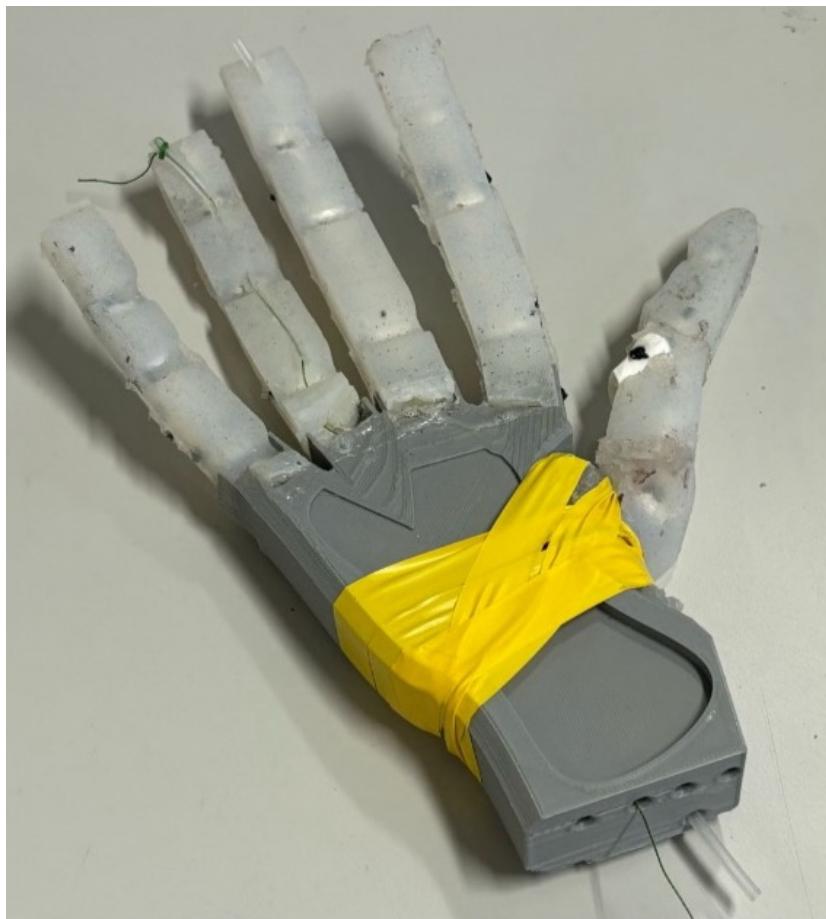


Abbildung 41: Silikonhand Fehlversuch 3

In der Praxis hat sich dann herausgestellt, dass die Finger sehr weich sind und daher ihr eigenes Gewicht kaum halten können. Der hat Faltmechanismus funktioniert, jedoch nicht ausreichend und in auch nicht in gewünschtem Maße. Lösen könnte man die Instabilität der einzelnen Fingern mit härterem Silikon. Grundsätzlich wurde moderat hartes Silikongewählt mit dem Härtefaktor 25A, das doppelte somit 50A oder mehr wären notwendig um die Finger stabil zu halten.

Die Problematik mit dem Guss, dass das Modell sich nicht in einem gießen lässt und die Erkenntnis beim Notmodell, dass die Finger nicht stabil bleiben, hat gezeigt dass dieses Konzept für unser Projekt nicht tauglich ist. Aufgrund dieser Ergebnisse wird das Konzept vorerst verworfen, dennoch soll Silikon weiterhin ein Bestandteil der zukünftigen Modelle bleiben besonders aufgrund des Griffes, der damit gesichert werden kann.

### 6.2.6 Vierte Hand (Hybrid-Konzept)

#### 6.2.6.1 Erfahrung und Konzepterläuterung

Aus den Versuchen der letzten Kapitel konnte einiges an Erfahrung gesammelt werden. Nun ist klar worauf bei dem Modell einer Hand geachtet werden muss.

Wie in Kapitel Unterunterabschnitt 6.2.3 hat sich gezeigt, dass die einzelnen Fingerglieder sich kontinuierlich um die anderen Falten müssen. Auch hat sich im Kapitel „zweite Testhand“ gezeigt, wie wichtig es ist für ein Modell zu sorgen, dass einer hohen Last standhält und nicht brüchig ist. In den letzten Versuchen hat sich gezeigt, dass die Zugschnur an sämtlichen Stellen auf Reibung trifft.

Entdeckt wurde eine erweiterte Version der Hand aus Unterunterabschnitt 6.2.4 von Gael Levin. Diese hat sämtliche Optimierungen, sei es die zu bemängelnde Stabilität der vorherigen Version oder auch der problematische Mechanismus der Zugschnüre. Im neuen Modell der i2Hand“, sind die Fingerglieder besonders an ihren Schwachstellen verstärkt worden, somit kann verhindert werden, dass die Verbindungen der einzelnen Glieder bei hoher Last oder gar der Montage brechen. Die Verbindungen haben vorgesehene Einkerbungen für den Kopf der Schraube oder eine Mutter, somit steht an den Seiten der Finger nichts ab.

Der Faltmechanismus besteht nicht mehr aus zwei Schnüre, sondern reduziert sich ausschließlich auf die Zugschnur, diese hat eine eigene Führung mit abgerundeten Kanten um die Reibung zu reduzieren. Der Rückzug wird durch eine Zugfeder verursacht, diese erstreckt sich durch den gesamten Finger und ist am Anfang und seinem Ende montiert.

Doch nicht alles war ideal an dem Modell der neuen Hand, den der Motorblock wurde nicht optimiert und nutzt weiterhin den Mechanismus, in dem sich die Zugschnur um einen Aufsatz auf dem Motor wickelt. Daher muss eigens ein neuer Mechanismus entworfen werden in diesem soll die Drehbewegung des Servomotors in eine lineare Zugbewegung übersetzt werden.

### 6.2.6.2 Erster Entwurf

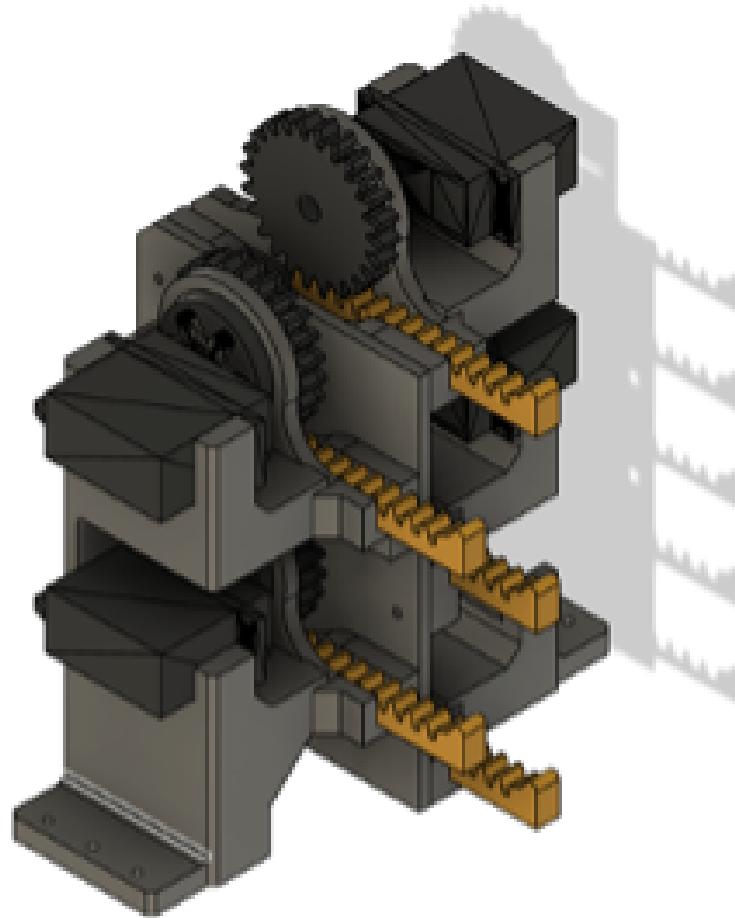


Abbildung 42: Motorblock v2

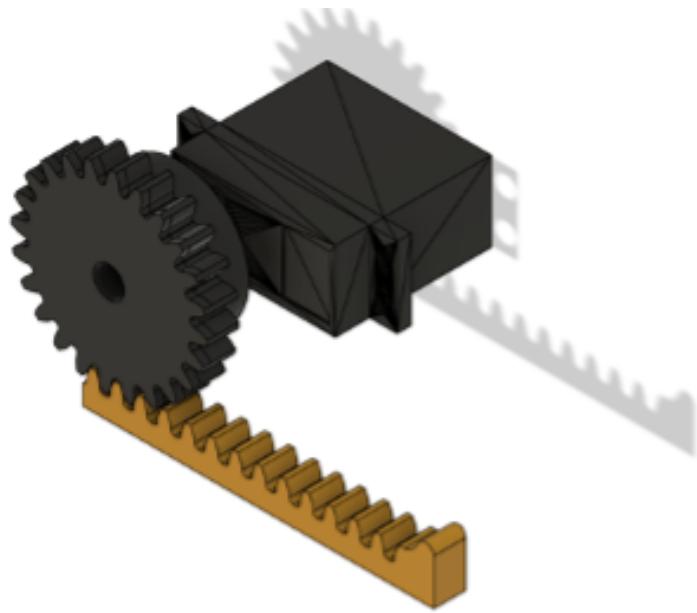


Abbildung 43: Motorblock v2 Mechanismus

In der Abbildung 43 ist ein Gerüst zu sehen, dieses in Kombination mit den fünf Motoren, den zugehörigen Zahnstangen und Zahnrädern als Motorblock bezeichnet. Auf jeden Motor wird ein Zahnräder befestigt, dreht sich dieses Zahnräder bewegt es die in der Fassung platzierte Zahnstange. Der Mechanismus nennt sich "Rack and Pinion" (engl.) und wurde so dimensioniert um mit einer Drehung von 180 Grad die Zahnstange um 60mm zu bewegen.

Nach dem Druck und einer ersten Inbetriebnahme hat sich die Funktionalität des Mechanismus erwiesen. Der Motor dreht sich und wandelt dies in eine lineare Zugbewegung um, auf der Zahnstange wird dann die Angelschnur montiert und somit kann an dieser kontrolliert gezogen werden.

#### 6.2.6.3 Verbesserungen – zweiter Entwurf

Neben der Funktionalität hat sich gezeigt, dass das Modell in einer deutlich kleineren Form und raumeffizienter entworfen werden kann. Dafür spricht, dass sich die Zahnstange nur um 30mm Bewegen muss um den Finger ausreichend zu Falten. Dadurch konnte der Durchmesser des Zahnrades um ein Vielfaches verkleinert werden und dies bewirkt, dass die Halterung auch kleiner ausfällt.

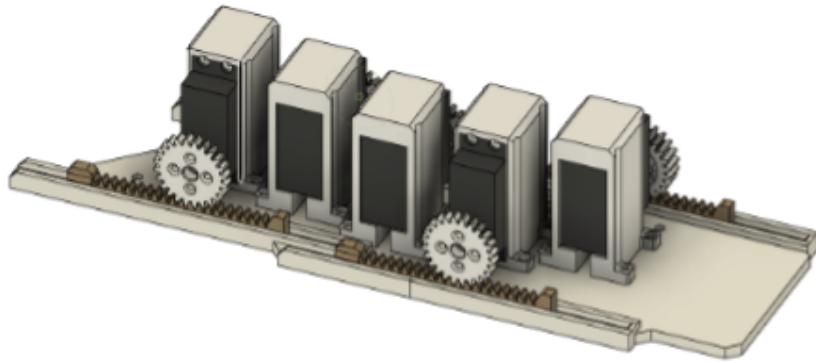


Abbildung 44: Zahnstangenmechanismus Unterarm

In diesem Modell ist zu sehen, dass alle Servomotoren auf einer Höhe liegen. Diese werden auf einer Servohalterung (2) montiert, welche dann auf der Platte (1) montiert wird, aufgrund der Toleranzen des 3D-Drucks können die Halterungen mit Langlöchern adjustiert werden. Das Modell wurde mit der Rücksicht entworfen, Schrauben und Muttern im Modell zu versenken.

#### 6.2.6.4 Realisierung und Aufbau

#### 6.2.6.5 Versuche...

---

## 7 Hardware Realisierung

### 7.1 Eingabesubsystem

#### 7.1.1 Grundlegende Voraussetzungen Laci

Die Hardware des Eingabesubsystems muss einige Kriterien erfüllen, um schlussendlich voll funktionfähig in das Gesamtsystem eingebaut werden zu können. Zunächst ist die Größe der Schaltung, als mit der wichtigste Punkt zu nennen. Da es bei diesem Projekt nicht nur um die Funktionalität, sondern auch um ein ergonomisches Benutzererlebnis geht, sollte die entgültige Platine auf dem Handschuhrücken nicht größer als 4cm x 4cm sein. Beim Design der Elektronik muss durch die Größenrestriktion natürlich darauf geachtet, dass durch diese keine Einbußen in Bezug auf die korrekte und sichere Funktion des Projektteils entstehen. Übermäßige Wärmeentwicklung ist ebenfalls zu vermeiden, da diese auf lange Zeit unangenehm für den Endnutzer ist. Die Versorgung der Schaltung sollte möglichst über einen kleinen und portablen Akku geschehen, um dem Benutzer das bestmögliche Erlebnis zu bereiten. Dieses Feature ist allerdings optional und ist daher nicht garantiert zur Verwendung bereit. Über einen USB Anschluss soll der Mikrokontroller programmierbar sein und die Schaltung auch für Test -und Wartungszwecke versorgt werden können. Das Maximalgewicht darf 500g nicht übersteigen.

#### 7.1.2 Überlegungen, Simulationen und Berechnungen Laci

##### 7.1.2.1 Bewegungserfassung der Fingerbeugung

Die Bewegungen des Benutzers müssen gemessen werden können. Das bedeutet, dass eine Form von Messschaltung notwendig ist um die Beugung der Finger interpretieren zu können. Dies könnte man durch das Messen des Beugungswinkels realisieren. Allerdings hat jeder Finger drei Gelenke, wodurch man diese auch bei der Roboterhand individuell steuern müsste. Die zweite Möglichkeit wäre, durch eine visuelle Aufnahme die Bewegung des Handschuhs und dadurch des Benutzers aufzuzeichnen. Da dies allerdings nur in dafür vorgesehenen, mit Kameras ausgestatteten, Räumen funktionieren würde, ist dies für uns auch keine sinnvolle Möglichkeit. Schließlich haben wir uns für die Erfassung der Fingerbewegungen mittels Flexsensoren entschieden. Diese ändern den Widerstand je nach der aktuell vorherrschenden Beugung. Bei dieser Art der Bewegungserfassung muss man nicht jedes Fingergelenk einzeln steuern und braucht auch keine externen Kameras. Somit ist bei dieser Methode der Datenerfassung ein sehr flexibler Verwendungsbereich des Handschuhs gewährleistet. Um die Änderungen der Widerstandsstreifen messen und verarbeiten zu können ist nun eine Schaltung notwendig. Diese muss Wertdifferenzen erkennen und in ein geeignetes Format zur Weiterverarbeitung mit einem Mikrokontroller umwandeln können.

##### 7.1.2.2 Auslesen der Sensoren

Das Auslesen der Flexsensoren kann durch einen einfachen Spannungsteiler erfolgen. Dabei ist die Genauigkeit allerdings nicht optimal und ist daher nicht für unsere Anwendung geeignet.

## 7.1 Eingabesubsystem

Als Lösung für dieses Problem haben wir an eine OPV-Messschaltung gedacht. Diese soll mit einem Shuntwiderstand die Spannungsdifferenz messen, die sich bei einer Veränderung des flexiblen Widerstands ergibt. Durch eine geeignete Verstärkung des OPVs kann diese mit einer Referenzspannung verglichen werden.

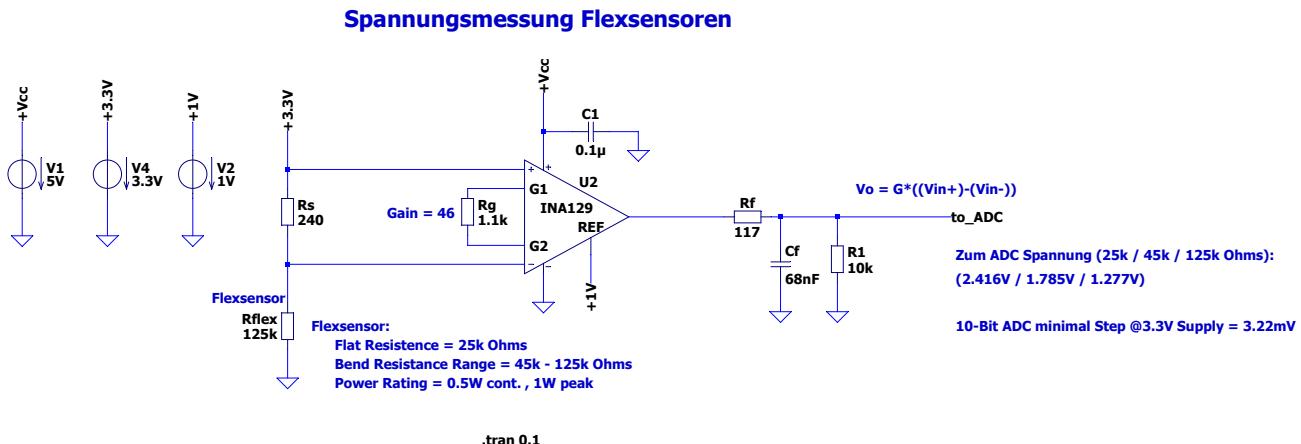


Abbildung 45: LTspice Simulation der Flexesnor - Messschaltung

Die Flexsensoren ( $R_{flex}$ ) beziehen ihre Versorgung über einen Shunt-Widerstand ( $R_s$ ). Je nach Belastung, ändert sich der Spannungsabfall an diesem (Je größer die Beugung des Sensors, desto kleiner ist der Spannungsabfall). Die Spannungsdifferenz am Shunt-Widerstand wird von einem Operationsverstärker verstärkt. Bei der Auswahl des OPVs sind einige Punkte zu beachten, um eine korrekte und genaue Erfassung der Fingerbeugung zu gewährleisten.

Folgende Kriterien müssen folglich bei der Wahl des Operationsverstärkers beachtet werden:

- Ausgangspegel bei gewählter Versorgungsspannung:

Zunächst wurde das Kriterium der Versorgungsspannung betrachtet. Da wir maximal 5V Gleichspannung in der gesamten Schaltung verwenden wollen, muss der OPV mit dieser geringen Spannung immer noch verstärken. Da am positiven Verstärkereingang eine maximale Spannung von 3.3V anliegt, muss dies bei Verstärkern mit einer geeigneten Supply Range auch mit nur 5V Versorgungsspannung gewährleistet sein.

- Referenzspannung:

Ein weiteres Kriterium ist das vorhanden sein eines Referenzspannungsanschlusses. Da der OPV keine Rail-to-Rail Technologie besitzt, muss der Ausgangsspannungspegel auf ein gewisses Minimum angehoben werden. In unserem Fall ist dies +1V. Würde diese Referenzspannung nicht vorhanden sein, so würde der OPV falsche Werte erzeugen, da dieser erst ab einer verstärkten Spannung am Ausgang von ca. 750mV korrekt funktioniert.

Aufgrund dieser Kriterien und der Notwendigkeit von Genauigkeit und geringer Störungseigenschaften, viel die Wahl des Operationsverstärkers auf den INA129 instrumentation amplifier.

Der Shunt-Widerstand wurde nicht berechnet. Dieser wurde einfach durch probieren in der Simulation bestimmt.

Ein Tiefpassfilter ( $R_f$  und  $C_f$ ) ist hinter den Ausgang des OPVs geschaltet, um mögliche Spannungsstörungen (Ripple), zusätzlich zu dem ohnehin schon sehr störungssarmen Ausgangssignal des INA129, herauszufiltern. Der zu GND geschaltete Widerstand ( $R_1$ ), entlastet den Eingang des folgenden ADCs. Der Tiefpassfilter wurde folgendermaßen dimensioniert.

### 7.1.2.3 Berechnung des Tiefpassfilters

$$f_g = 20\text{kHz} \quad \tau = \frac{1}{\omega_g} = \frac{1}{2\pi * 20\text{kHz}}$$

$$f_g = \frac{\omega_g}{2\pi} \quad \tau = R * C$$

$$C = 68\text{nF} \quad R = \frac{\tau}{68\text{nF}} = 117\Omega$$

### 7.1.2.4 Berechnung der OPV Verstärkung und Dimensionierung des Shuntwiderstands

$$\text{Verstärkungsgleichung laut Datenblatt: } G = 1 + \frac{49.4k\Omega}{R_g}$$

$$\text{Gain gewählt mit 46.} \quad R_g = 1.1k\Omega$$

Die Verstärkung wurde so gewählt, dass diese mit dem Shunt-Widerstand für den ADC optimal geeignet ist. Der Shunt-Widerstand wurde mithilfe der Simulation in Abbildung 45 gewählt.

Der Shuntwiderstand wurde nicht wirklich berechnet. Dieser wurde durch probieren in der Simulation ermittelt. Eine Berechnung des Shunts wäre nicht wirklich Zielführend gewesen, da diese normalerweise bei Schaltungen mit hohen Strömen verwendet werden. Da sich die Flexsensoren allerdings in einem Widerstandsbereich von  $25\Omega$  -  $125\Omega$  befinden, benötigen diese nicht viel Strom, wodurch schon zu erwarten war, dass ein relativ hoher Wert benötigt wird. Schlussendlich wurden  $240\Omega$  gewählt, da dieser Widerstand bei sowohl voller, als auch geringer Biegung der Flexsensoren, eine gute Spannungsdifferenz für die Verstärkung mit dem OPV liefert.

### 7.1.2.5 Umwandlung der Differenzwerte in ein geeignetes Format

Um nun die analogen Ausgangswerte des Operationsverstärkers nach der Verstärkung der

## 7.1 Eingabesubsystem

---

Spannungsdifferenzen am Flexsensor für den Mikrokontroller möglichst effizient und brauchbar zu machen, ist eine Umwandlung in ein digitales Signal notwendig. Dies Funktion wird mit einem ADC umgesetzt. Bei der Wahl dieses Logikbauteils, sind, wie beim OPV, einige Kriterien zu beachten um die korrekte Funktion der Schaltung weiterhin zu gewährleisten.

Folgende drei Kriterien sind maßgeblich bei der Wahl der Analog-Digital-Wandlers zu beachten:

- Genauigkeit, Auflösung und Aussteuerbereich:

$$\text{Aussteuerbereich} = 0 - 3.3V$$

bei 10Bit ADC:  $LSB = 3.22mV$

Wegen der 1V Referenzspannung des OPVs ist der Ausgangspegel 1V - 3.3V

$$ADC \text{Ausgangsstufen} = \frac{2.3V}{3.22mV} = 713$$

Der reale Ausgangspegel des OPVs liegt, wie bei der Simulation in Absatz 7.1.2.2 ermittelt, zwischen 1.277V bis 2.416V. Das bedeutet, dass eine Auflösung von 10Bit und ein Austeuerbereich von 0V - 3.3V ausreichend ist, um den kompletten Wertebereich sehr genau abzudecken. Zusätzlich haben wir uns noch dazu entschieden alle Logikbauteile die eine Kommunikation mit dem Mikrokontroller erfordern mit dem I2C Bussystem anzuschließen. Daher muss der Analog-Digital-Wandler diese Art der Kommunikation ebenfalls unterstützen. Aufgrund dieser Auswahlkriterien ist die Wahl des Bauteils auf den MAX11611 gefallen.

### 7.1.2.6 Vervielfachung der Schaltung für alle Flexsensoren

Um die zuvor beschriebene Schaltung nun nicht für jeden Flexsensor einzeln bauen zu müssen, wäre eine Art Schalter vorteilhaft. Dieser soll in Sekundenbruchteilen zwischen allen Sensoren durchschalten. Das bedeutet also, dass zwischen dem Shunt-Widerstand und  $R_{flex}$  in der Simulation dieses Bauteil platziert werden muss.

Für diesen Zweck ist ein Multiplexer bestens geeignet. Folgende Kriterien muss dieser erfüllen.

- Versorgung und Kanäle:

Die Versorgung muss an den Rest der Schaltung angepasst sein, das bedeutet, dass entweder 3.3V oder 5V in Frage kommen. Bei einer Anzahl von einem Flexsensor pro Finger, also fünf, muss der Multiplexer mindestens 5 Kanäle aufweisen, wobei mehr Kanäle für mögliche zukünftige Erweiterungen kein Problem sind. All diese Eingänge müssen auf einen Ausgang geschalten werden.

- Ansteuerung:

Die Ansteuerung muss mit einem Mikrokontroller möglich sein. Hier bleibt also die Wahl zwischen analogen und digitalen Anschlüssen, oder ein I2C Anschluss um mit dem Rest der Schaltung kompatibel zu bleiben.

Folglich viel die Wahl auf den MUX508IDR. Dieser ist ein 8:1 Channel Multiplexer, der über 5V versorgt werden kann und über drei Analoganschlüsse für die Auswahl des Kanals verfügt.

### 7.1.2.7 Bewegungserfassung der Handgelenksdrehung

Um die Drehung des Handgelenks zu erfassen ist ein anderer Sensor als ein Flexsensor notwendig. Dieser neue Sensor muss die Funktion eines Gyroskops haben und folglich die Positionen von X, Y -und Z-Achse übermitteln. Dieser Übermittlung muss per I2C-Bus erfolgen, um die Kompatibilität mit der restlichen Schaltung zu ermöglichen.

Ausgewählt wurde der Sensor MPU-6000, da dieser schon eingebaute ADCs hat, um die Achswerte vor der Übertragung zu digitalisieren.

### 7.1.2.8 Mikrokontroller

Bei der Auswahl des Mikrocontrollers wurden sehr viele Aspekte beachtet. Dieser ist das Herzstück der Schaltung und ermöglicht allen Komponenten zusammen zu funktionieren und diese auch zu steuern.

Folgende Kriterien müssen von dem verwendeten Mikrocontroller folglich erfüllt werden:

- Performancerelevante Ressourcen:

Zu beachten ist hierbei vor allem der vorhandene Flash-Speicher, die CPU und der On-Chip Memory. Hierbei gilt grundsätzlich natürlich je mehr, desto besser. Das gleiche gilt ebenfalls für den Flash-Speicher.

- Versorgung und Anschlüsse:

Um mit der Schaltung kompatibel zu sein, muss der Mikrocontroller mit 3.3V oder 5V versorgt werden können. Zusätzlich sollte der Chip möglichst wenig Leistung brauchen. Es sollten mindestens zehn I/O-Anschlüsse vorhanden sein.

- Unterstützte Bussysteme:

Da wir bei der Schaltung einheitlich auf das I2C Bussystem setzen, muss zumindest dieses von dem gewählten Mikrocontroller unterstützt werden. Als zweite Pflichtunterstützung gilt die UART-Kommunikation. Die Funktion und Notwendigkeit dieser, wird in Absatz 7.1.4.1 erläutert.

## 7.1 Eingabesubsystem

---

- Möglichkeiten der drahtlosen Übertragung:

Da die Flexsensorwerte drahtlos übertragen werden müssen, muss der Mikrokontroller eine Form dieser Übertragung unterstützen. Vorzüglichweise ist die Antenne für die Übertragung schon vorhanden, damit weitere Schaltungsteile nicht notwendig sind. Hier kämen zum Beispiel Bluetooth oder Wifi in Frage.

- Programmierbarkeit:

Der Chip muss mit einer schon verfügbaren Entwicklungsumgebung programmierbar sein. Wichtig ist in diesem Bezug vor allem die Debugmöglichkeit, da bei einigen Mikrokontrollern ein extra Debugtool um viel Geld erworben werden muss. Eine Programmierung im Terminal kommt ebenfalls nicht in Frage.

- Größe und Formfaktor:

Schlussendlich dürfen sich alle Kriterien jedoch nicht zu sehr auf die Größe des Mikrocontrollers auswirken. Diese sollte natürlich so klein wie möglich sein und trotzdem Bauteile wie eine Antenne aufweisen.

Nach beachtung aller Kriterien haben wir uns für einen ESP32 Mikrokontroller entschieden. Hierbei blieb allerdings die Wahl zwischen dem reinen Chip und dem Modul, bei dem die Antenne und andere Funktionalitäten, die andernfalls selbst gebaut werden müssten, schon integriert sind. Nach Abwägungen von Größe und Performance, haben wir uns für das ESP32-WROOM-32E-N16 Modul entschieden. Dieses hat eine integrierte Antenne, reichlich Performance und viel Flash-Speicher. Der Formfaktor ist bei allen diesen Funktionalitäten immer noch im Rahmen.

### 7.1.2.9 Akkuversorgung

Da es nicht praktikabel ist die Schaltung des Handschuhs dauerhaft mit einem Kabel zu versorgen, soll dies schlussendlich durch einen Akku oder eine Batterie erfolgen. Entschieden haben wir uns für eine Lithium-Polymer-Akku (LiPo), da diese trotz geringer Größe verglichen mit anderen Akkuarten eine hohe Kapazität besitzen.

Die notwendige Kapazität für eine bestimmte Betriebsdauer wurde folgendermaßen berechnet:  
**Berechnung Akkukapazität hier einfügen**

Um den LiPo-Akku nicht jedes mal extern aufladen zu müssen, haben wir uns eine Schaltung überlegt, die dies auch mithilfe des schon vorhandenen USB-C Anschlusses für das Programmieren des Mikrocontrollers verwendet wird. Dazu ist allerdings eine relativ komplizierte Schaltung notwendig, weswegen bei vielen Produkten, die LiPo-Akkus verwenden, extra Ladegeräte gekauft werden müssen. Dies wollten wir vermeiden und haben dementsprechend viel Zeit in die Reserche für eine funktionierende LiPo-Kontroller-Schaltung investiert.

Zunächst ist jedoch wichtig zu wissen, welchen Akku wir überhaupt erwenden. Entschieden

haben wir uns für den **LiPo-Akku**.

### 7.1.3 Versuchsaufbauten und Messungen **Laci**

In den folgenden Punkten, werden die praktischen Tests der, in Unterunterabschnitt 7.1.2 beschriebenen Konzepte und Überlegungen, erläutert.

#### 7.1.3.1 Messschaltung der Flexsensoren

Zunächst wurde die Schaltung aus Absatz 7.1.2.2 auf einem Steckbrett aufgebaut. Das Ziel der Messung ist es, den Widerstand bei jeder Stellung eines Flexsensors zu wissen. Dies ist wichtig, um die Daten der Sensoren korrekt auszulesen und an die Roboterhand zu senden. Werden die Widerstandswerte nicht korrekt gemessen, so bewegt sich die Roboterhand nicht entsprechend nach den Bewegungen des Benutzers.

Die zu messende Größe ist die Ausgangsspannung des OPVs, die sich je nach Widerstand der Last verändert. Die Last wird in unserem Versuchsaufbau von drei Widerständen simuliert, da wir die Flexsensoren zu dem Zeitpunkt der Messung noch nicht zu Verfügung hatten. Wir haben die Lastwiderstände mit 27k, 68k und 118k gewählt, da der Widerstandsbereich der Sensoren bei 25k – 125k liegt. In Absatz 7.1.3.2 werden die Messergebnisse dargestellt.

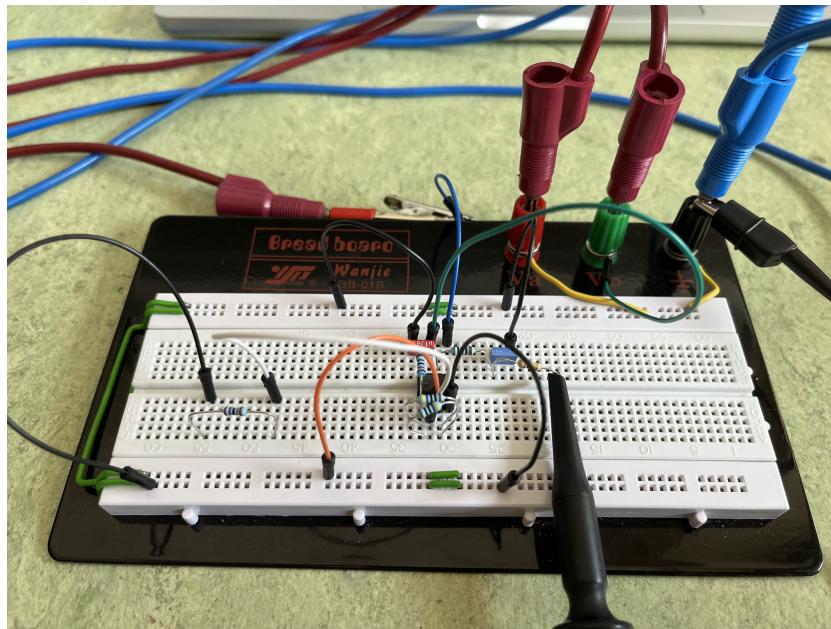


Abbildung 46: Steckbrettaufbau der Flexesnor - Messschaltung

## 7.1 Eingabesubsystem

---



Abbildung 47: Netzteileinstellungen

+3.3V für die Versorgung des Flexsensors (Last), simuliert mit drei normalen Widerständen.

+1V als Referenzspannung für den INA-129.

+5V zur Versorgung des INA129.

### 7.1.3.2 Messergebnisse



Abbildung 48: Oszilloskopbild mit  $R_L = 27k\Omega$

Das obige Oszilloskopbild zeigt die, mit einem Tastkopf, gemessene Ausgangsspannung des Operationsverstärkers, bei einem Flexsensorwiderstand von  $27k\Omega$ . Die gemessene Spannung ist rot eingekreist und beträgt 2.28V. Da bei der Simulation in LTspice mit einem Widerstand von  $25k\Omega$  eine Ausgangsspannung von 2.416V resultierte, ist anzunehmen, dass 2.28V für den gewählten Widerstand angemessen sind. Daraus kann man schließen, dass die Schaltung für diesen Widerstandswert des Flexsensors korrekt funktioniert.

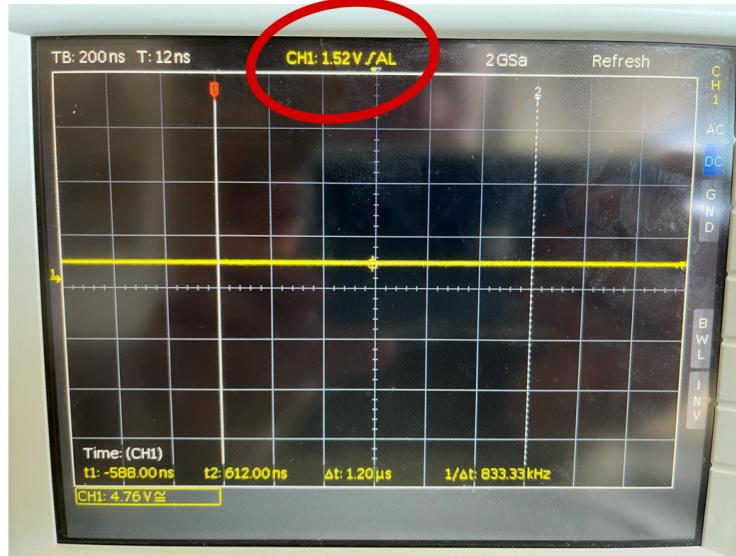


Abbildung 49: Oszilloskopbild mit  $R_L = 67\text{k}\Omega$

Nun wurde die gleiche Messung erneut durchgeführt, allerdings mit einem Flexsensorwiderstand von  $67\text{k}\Omega$ . Wir zu erwarten, ist die Ausgangsspannung des OPVs gesunken, da der Spannungsabfall am Shuntwiderstand nun geringer ausfällt. Zu erwarten war ein Spannungswert von rund 1.5V. Dieser Pegel wurde mit 1.52V fast genau getroffen, wodurch die funktionsfähigkeit der Schaltung weiter sichergestellt wurde.

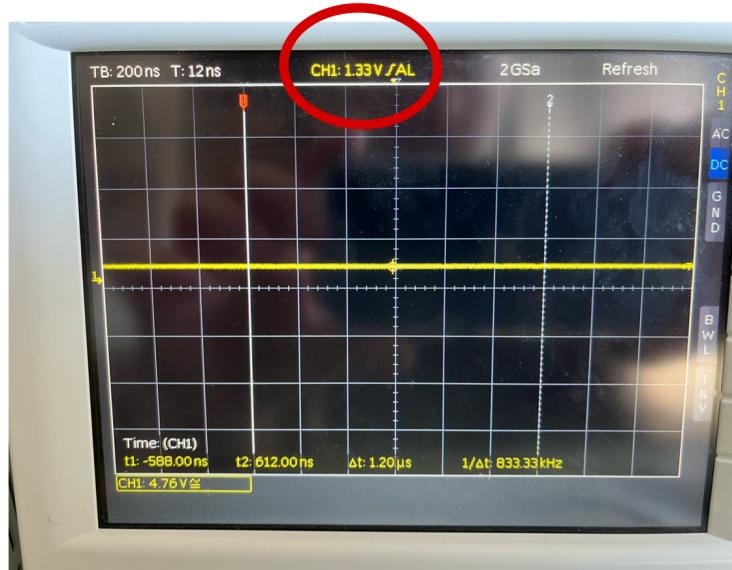


Abbildung 50: Oszilloskopbild mit  $R_L = 118k\Omega$

Schlussendlich wurde noch ein Flexsensorwiderstand von  $118k\Omega$  simuliert, um den nahezu Maximalwert des Sensors zu simulieren. Zu erwarten waren rund 1.3V. Die Messung bestätigt die Überlegungen, wodurch die Funktionalität dieser Schaltungskomponente bewiesen wurde.

Zusätzlich ist nun die Notwendigkeit einer Referenzspannung von +1V bewiesen worden, da ohne dieser ein theoretischer Spannungspegel von nur 500mV am Ausgang des INA-129 aufgetreten wäre. Dies ist praktisch allerdings nicht möglich, da der OPV über keine Rail-to-Rail Technologie verfügt, wodurch er nur eine minimale Ausgangsspannung von rund 800mV ausgeben kann.

### 7.1.4 Schaltungsdesign Laci

Nach den generellen Überlegungen und Versuchsaufbauten, die zu der Entwicklung der Schaltung des Eingabesubsystems beigetragen haben, wird in diesem Punkt das genaue Schaltungsdesign erläutert.

#### 7.1.4.1 Externe Anschlüsse

- Anschluss zur Programmierung des Mikrocontrollers:

Für die Programmierung des Mikrocontrollers wird ein USB Anschluss benötigt. Dieser sollte möglichst kompatibel mit den neuesten Computern sein, weswegen wir uns für USB-C-Typ2.0 entschieden haben. Um das Serial Signal der USB Schnittstelle für den Mikrokontroller lesbar zu machen, muss dieses für die UART-Kommunikation umgewandelt werden. Hierzu muss der Mikrokontroller diese auch unterstützen. Mit einer Serial-UART-Bridge, wird das Signal umgewandelt. Der Chip wird von +3.3V versorgt. Von der USB-Buchse werden die beiden Datenleitungen D+ und D- mit verbunden. Anschließend wird das umgewandelte Signal über die UART-Leitungen an den ESP32 Mikrokontroller übertragen. Wichtig zu beachten ist hierbei, dass die UART-Leitungen ausgekreuzt sein müssen. Die Funktion der Anschlüsse RTS und DTR wird in Absatz 7.1.2.8 erläutert.

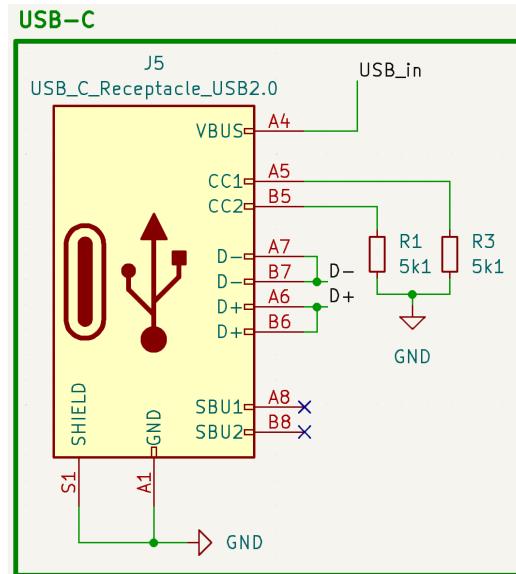


Abbildung 51: USB-C-Buchse

- Anschluss der Flexsensoren:

Die Flexsensoren könnten natürlich einfach angelötet werden, jedoch ist die einfache Wartung bei einem fehlerhaften Sensor ebenfalls zu berücksichtigen. Aufgrund dessen wird eine 6-Pin-JST-Buchse als Anschluss verwendet. Alle GND-Pins werden auf einen zusammengefasst, um möglichst viel Platz zu Sparen.

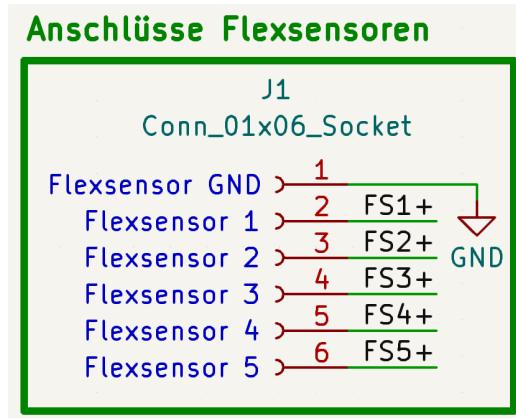


Abbildung 52: Flexsensor Connector

#### 7.1.4.2 Mikrokontroller

Der ESP32-Chip wird mit +3.3V versorgt. Wichtig zu beachten ist dadurch, dass ein Logic-HIGH somit auch 3.3V und nicht 5V ist! Die Versorgung wird mit einem Kondensator stabilisiert. Für alle Busleitungen, also I2C und UART, wurden Widerstände in Serie hinzugefügt, um die Kommunikationsleitungen vor Spannungsspitzen oder Überspannung zu schützen. Die Funktion wäre auch ohne diese gegeben. Für die Datenleitungen SDA und SCL des I2c Busses sind zwei  $10k\Omega$  PullUp-Widerstände vorgesehen. Zusätzlich wird ebenfalls der Anschluss IO16 mit einem PullUp-Widerstand auf 3.3V gezogen, da dies so vom Datenblatt vorgegeben wird. Die Funktionen der einzelnen Anschlüsse werden in den folgenden Punkten gemeinsam mit den damit verbundenen Bauteilen näher erläutert.

## 7.1 Eingabesubsystem

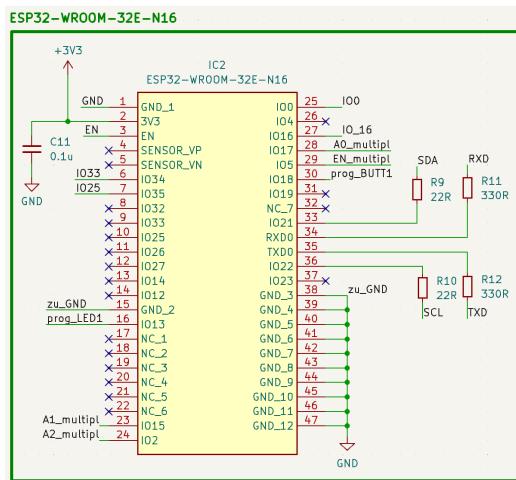


Abbildung 53: ESP32 Mikrokontroller

### 7.1.4.3 Mikrokontroller Buttons

In der Schaltung sind drei Taster verbaut. Diese sind alle mit dem Mikrokontroller verbunden und erfüllen verschiedene Funktionen.

- Upload Button:

Dieser Button ist mit dem IO0 Anschluss des ESP32 verbunden und muss bei dem Hochladen von Code kurzzeitig gedrückt werden, um den Mikrokontroller in den Upload-Modus zu versetzen. Der Pin IO0 ist standardmäßig für diese Funktion vorgesehen und sollte für nichts anderes verwendet werden.

- Reset Button:

Dieser Button ist mit dem EN (Enable) Anschluss des ESP32 verbunden. Die Funktion dieses Tasters ist es, den ESP32 jederzeit zurücksetzen zu können falls dieser abstürzt oder ein anderweitiges Problem auftritt, durch das dieser nicht mehr korrekt funktioniert.

- Programmable Button:

Die Funktion dieses Buttons kann frei durch den programmierten Code gewählt werden.

### 7.1.4.4 Status LEDs

Die beiden Leuchtdioden sind als Statusanzeige gedacht. Eine POWER LED, die immer leuchtet wenn die Schaltung mit +5V versorgt wird. Die Funktion der anderen LED ist, sowie bei einem Button, frei wählbar und ist deswegen mit Pin IO13 des ESP32 verbunden.

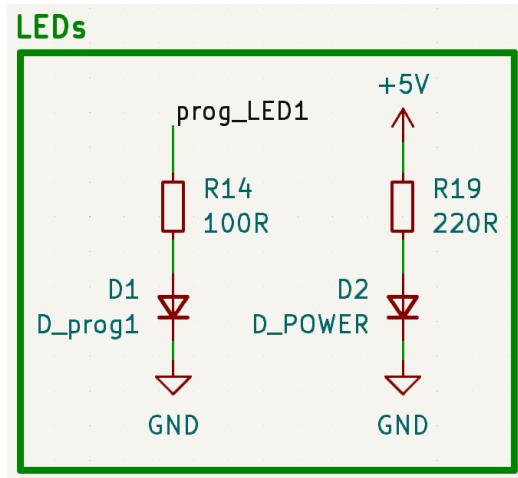


Abbildung 54: Status LEDs

#### 7.1.4.5 Multiplexer

Der Multiplexer schaltet zwischen allen Flexsensoren durch und vermeidet somit die Messschaltung fünf mal bauen zu müssen. Die Verbindung zum ESP32 erfolgt über drei digitale Addresspins und einen Enable Pin. Je nachdem welche Bitkombination übermittelt wird, ändert sich die interne Schalterposition und somit der gerade aktive Kanal zur Messung eines Flexsensors.

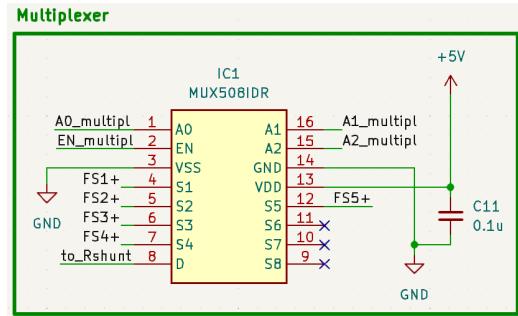


Abbildung 55: Multiplexer

#### 7.1.4.6 Operationsverstärker

Für das Auslesen der Flexsensoren, wird eine Schaltung, wie in Absatz 7.1.2.2 beschrieben, benötigt. Da am Shuntwiderstand nur sehr wenig Spannungsabfall auftritt und daher die Spannungsdifferenz zwischen positivem und negativem Verstärkereingang sehr klein ist, muss das Signal verstärkt werden. Hierfür wird der INA129 Instrumentenverstärker verwendet. Dieser wird mit 5V versorgt, was für einen OPV eine relativ geringe Versorgungsspannung ist. Da die maximale Eingangsspannung der Verstärkereingänge allerdings nie mehr als 3.3V beträgt, ist dies kein Problem.

## 7.1 Eingabesubsystem

---

Die +1V Spannungsreferenz ist notwendig, da der ausgewählte Operationsverstärker nicht über Rail-to-Rail Technologie besitzt. Die kleinstmögliche Spannungsdifferenz am Eingang des OPV, wenn der Flexsensor seinen maximalen Widerstand von  $125\text{k}\Omega$  erreicht, beträgt nur 6.3mV. Aufgrund der fehlenden Rail-to-Rail Fähigkeit, muss die Ausgangsspannung des OPV deshalb auf mindestens 800mV angehoben werden, um eine korrekte Funktion des OPVs zu gewährleisten. Deshalb wird eine Referenzspannung von 1V verwendet.

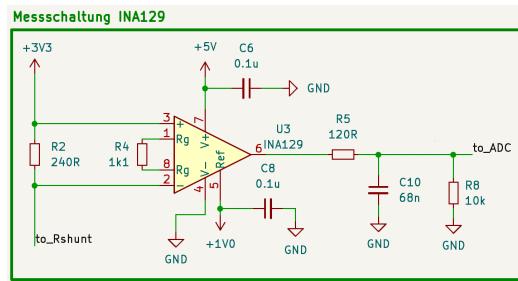


Abbildung 56: Operationsverstärker - Schaltung

### 7.1.4.7 Analog-Digital-Wandler

Der Analog-Digital-Wandler ist dafür zuständig, die analogen Werte, vom Ausgang des Operationsverstärkers kommend, in digitale Signale umzuwandeln. Der OPV und der ADC sind über das Label toADC verbunden. Da der Chip über 3.3V versorgt wird, befindet sich der mögliche Aussteuerbereich zwischen 0V und 3.3V. Wie im Absatz 7.1.2.5 berechnet, hat der ADC in unserer Schaltung ein LSB von 3.22mV. Da wir wissen, dass die kleinste Spannungsdifferenz am Shuntwiderstand 6.3mV ist und diese auch noch mit dem Faktor 46 verstärkt wird, kann festgestellt werden, dass der ADC mehr als genau genug für unsere Anwendung ist. Dies ist ein Vorteil, da bei der Programmierung anschließend nicht zwischen einzelnen ADC-Stufen unterschieden werden muss, sondern immer mehrere LSBs Unterschied auftritt. Die aktualisierten Werte, werden über die beiden I2C Leitungen an den Mikrokontroller übertragen.

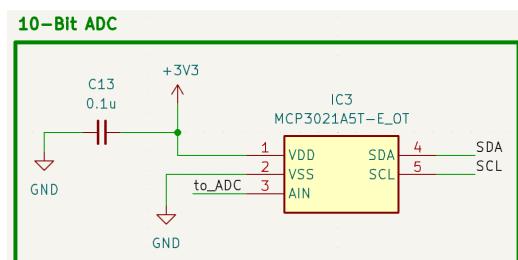


Abbildung 57: ADC - Schaltung

### 7.1.4.8 Gyroskop-Sensor

Der Gyroskopsensor ist dafür zuständig die Drehung des Handschuhs in X, Y -und Z Richtung zu übernehmen. Die Versorgung basiert auf 3.3V und die Datenübertragung erneut mittels I2C-Bussystem. Der Unterschied bei diesem Chip ist allerdings, dass mehrere ADCs integriert sind, um die Positionsdaten der Achsen schon digitalisiert an den Mikrokontroller zu übergeben. AD0 wird dabei verwendet um die Adresse des Mikrochips festzulegen. Diese wird später benötigt um mit dem Sensor Daten austauschen zu können.

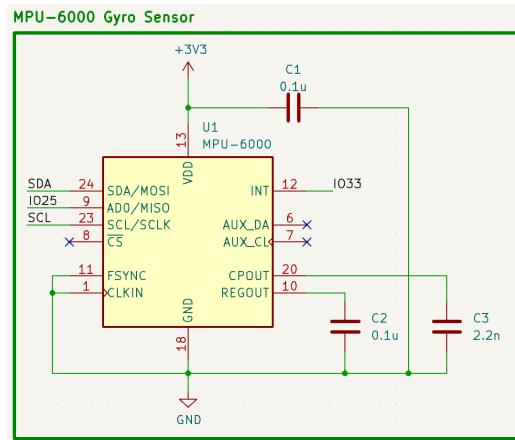


Abbildung 58: Gyroskopsensor - Schaltung

Am Pin CLKIN hätte man die Möglichkeit einen externen Referenztaktgeber anzuschließen. Bei unserer Anwendung ist der integrierte Taktgeber allerdings völlig ausreichend, weswegen der Anschluss mit GND verbunden und damit deaktiviert wurde. Der CS Pin wird nur für die SPI Kommunikation benötigt, weshalb dieser ebenfalls deaktiviert wurde. Das gleiche gilt für den Anschluss FSYNC. Die beiden AUX-Anschlüsse könnte man verwenden, um mit anderen I2C fähigen Sensoren direkt zu kommunizieren. Da dies allerdings unser einziger Sensorchip mit dieser Fähigkeit ist, bleiben die beiden Pins auch nicht verbunden. Die Kondensatoren sind vom Hersteller im Datenblatt vorgesehen und gewährleisten einen stabilen Betrieb.

### 7.1.5 Platinendesign Laci

#### 7.1.5.1 Erste Testplatine

Die erste Testplatine wurde in der schuleigenen Werkstatt, mithilfe einer Platinenfräse, gefertigt. Da das Design auch zu dem frühen Zeitpunkt des Projekts schon relativ kompliziert war, gab es bei der Fertigung einige Probleme. Aufgrund schon vorhandener Erfahrung, wurden die Bauteile mit möglichst großem Fotoprint (1206) gewählt, um ein einfaches Löten per Hand zu ermöglichen. Da es ICs aber leider nur in einer, für das Bauteil festgelegten, Größe zu kaufen gibt, mussten

## 7.1 Eingabesubsystem

---

Leiterbahnen mit einer Breite von nur 0.3mm verwendet werden. Dies führte beim Löten leider zum Ablösen dieser, weshalb einige Jumper-Kabel notwendig waren, um die kaputten Leiterbahnen zu ersetzen. Die Installation dieser Ersatzkabel, hat einige Zeit in Anspruch genommen, wodurch der Entwicklungsprozess der Hardware kurzzeitig verzögert wurde.

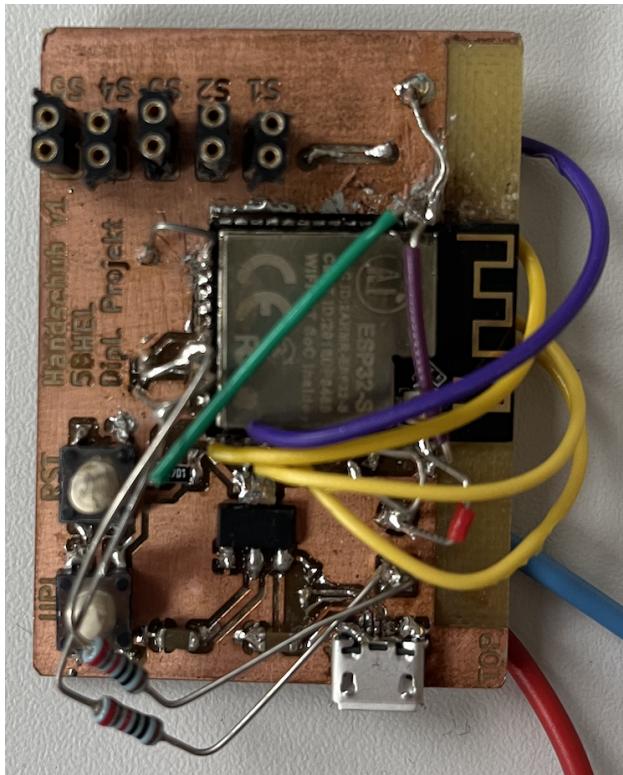


Abbildung 59: Platinenoberseite - erste Testplatine - Eingabesubsystem

Zu sehen ist die Oberseite der ersten Testplatine des Eingabesubsystems. In der oberen linken Ecke des Bilds, sind die Anschlüsse für die 5 Flexsensoren zu sehen. Diese werden für Testzwecke mit Jumper-Kabeln verbunden, um ein schnelles Auswechseln der Sensoren und einsetzen von fixen Referenzwidersänden zu ermöglichen. Bei der Fertigung der Platne sind einige Leiterbahnen sehr dünn ausgefallen, weshalb einige Kabel zur erneuten Verbindung von Lötpads gebraucht wurden. Zu dem frühen Entwicklungszeitpunkt, wurde noch eine Micro-USB-Buchse verwendet, da diese größere Pins hat und das Löten somit erleichtert.

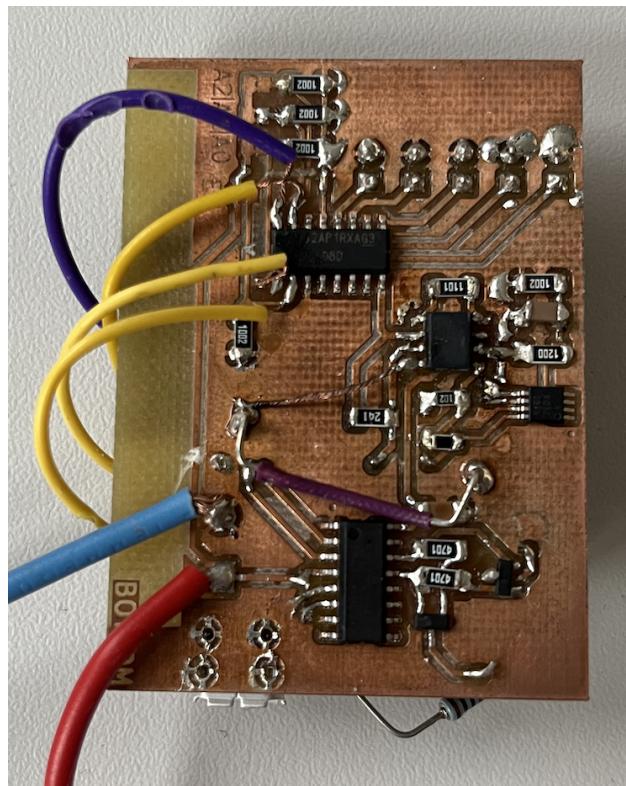


Abbildung 60: Platinenunterseite - erste Testplatine - Eingabesubsystem

An der Unterseite, kann der Grund für die Fehler in der Fertigung erkannt werden. Da die verwendeten ICs mit einer maximalen Leiterbahnbreite von 0.3mm verbunden werden müssen, kam es, vor allem bei der Serial-UART-Bridge in der oberen linken Ecke des Bilds, zu Leiterbahnablösungen. Das blaue und rote Kabel stellt die Versorgung der Platine dar.

### 7.1.5.2 Zweite Version der Platine

Bei dem Design der zweiten Platinenversion, wurden schon einige Anforderungen für die Integration in das Gesamtsystem berücksichtigt. Zu nennen sind hier beispielsweise die Abmessungen, die Konnektivität und die visuelle Darstellung von Parametern für den Benutzer.

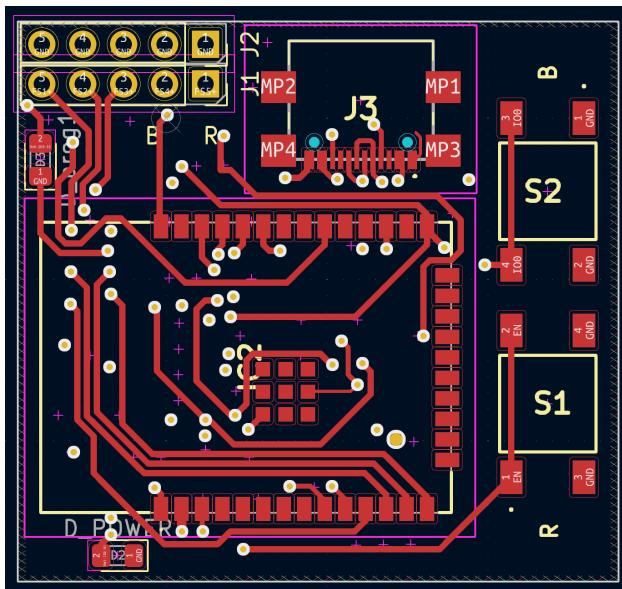


Abbildung 61: Toplayer des zweiten Platinenlayouts

In Abbildung 61, ist der Toplayer des zweiten Platinenlayouts in KiCad (siehe Unterabschnitt 15.3) zu sehen. Der Großteil des Platzes wird vom Mikrokontroller, dem ESP32, eingenommen. An diesen sind alle anderen Komponenten angeschlossen. Wie in Absatz 7.1.4.3 erläutert, sind zwei Buttons zur Steuerung des ESPs vorgesehen. Mit einem *R* ist der Reset-Button markiert mit einem *B* der Upload-Button.

In der linken oberen Ecke der Abbildung 61, sind die Anschlüsse für die Flexsensoren zu sehen. Rechts daneben ist die USB-C-Buchse platziert. Außer den genannten Bauteilen, sind nur mehr die beiden Leuchtdioden, zur Visualisierung für den Benutzer, auf dem Toplayer angebracht.

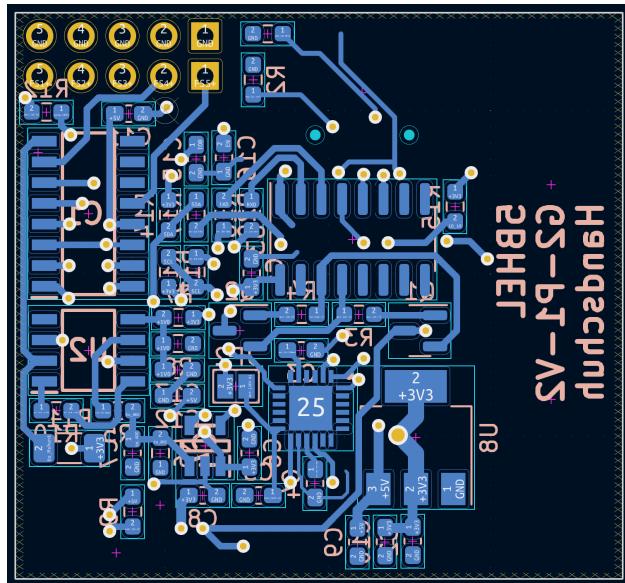


Abbildung 62: Bottomlayer des zweiten Platinenlayouts

In Abbildung 62, ist der Bottomlayer der zweiten Platinenversion zu sehen. Hier sind alle Elemente der Messschaltung (siehe Absatz 7.1.2.2) verbaut. Dies beinhaltet den Multiplexer, Operationsverstärker, ADC, Gyroskopsensor, Spannungswandler und die Serial-UART Bridge. Alle Bauteile wurden möglichst Platzsparend platziert, damit die Dimensionen der Platine gering gehalten werden können. Die Abmessungen betragen ca. 3.5cm x 3.5cm.

Alle Leiterbahnen wurden so breit wie möglich gemacht, um mögliche Defekte ausschließen zu können. Dies gilt für die Fertigung, als auch für mechanische Abnützungen bei der Verwendung.

Der größte Fehler bei diesem Platinendesign ist, dass in der Zone des Funkmoduls vom Mikrokontroller, Leiterbahnen und Bauteile platziert wurden. Dies beeinträchtigt die Übertragung allerdings nur teilweise, indem diese nur mehr bis zu 10 Meter weit funktioniert. Dies ist allerdings zum Testen kein Problem und wird beim nächsten Platinenlayout behoben.

## 7.1 Eingabesubsystem

---

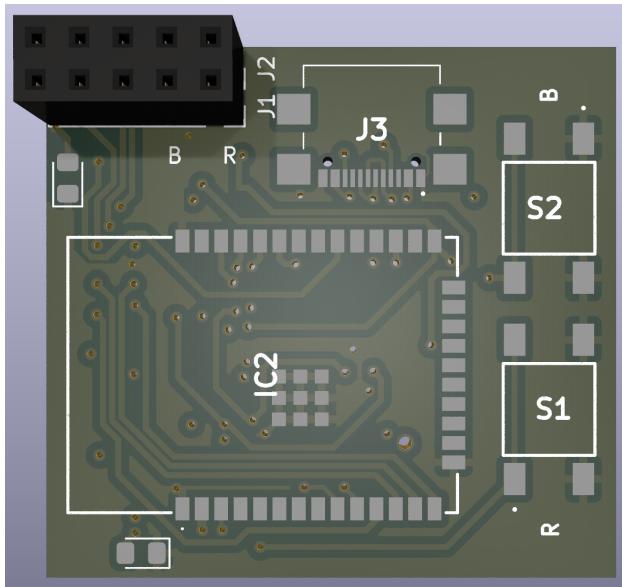


Abbildung 63: Topside des zweiten Platinenmodells

In Abbildung 63, ist ein 3D-Modell des Toplayers abgebildet. Dies ist eine exakte Representation der gefertigten Platine, die anschließend bestückt worden ist.

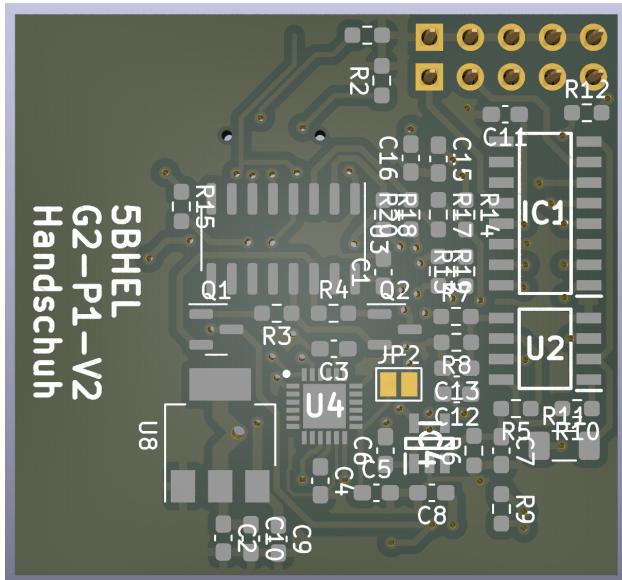


Abbildung 64: Bottomside des zweiten Platinenmodells

In Abbildung 64, ist das 3D-Modell des Bottomlayers abgebildet. Dies ist ebenfalls eine exakte Representation der gefertigten Platine und muss noch bestückt werden.



### 7.2 Roboterhand

#### 7.2.1 Grundlegende Voraussetzungen

Die Roboterhand ist die Ausgabe des Gesamtsystems und wird vom Eingabesubsystem gesteuert. Jeder Finger wird von einem Motor bewegt. Die ankommenden Daten müssen interpretiert und anschließend in ein geeignetes Format zur Ansteuerung der Motoren umgewandelt werden. Zusätzlich zu dieser grundlegenden Funktionalität der Ansteuerung, müssen auch noch einige weitere Aspekte beachtet werden. Hier sind zum Beispiel die möglichst geringe Größe der Schaltung und die trotzdem ausgiebigen Funktionen zu nennen. Dazu zählen visuelle Indikatoren für diverse Parameter, Schalter und Taster, Anschlüsse und Erweiterungsmöglichkeiten. Die detaillierten Entwicklungs -und Designschritte, werden in den folgenden Punkten näher erläutert.

#### 7.2.2 Überlegungen, Simulationen und Berechnungen **Laci**

##### 7.2.2.1 Aktorik zur Fingerbewegung

Bei der Aktorik zur Bewegung der Roboterfinger, gibt es, wie bei Sensorik zur Erfassung der Bewegungen, ebenfalls unzählige Methoden und Konzepte zur Auswahl. Bei schon etablierten Herstellern von bionischen Händen werden fast ausschließlich lineare DC Motoren verwendet. Diese sind sehr hochpräzise, sehr gut gefertigt und haben für ihre Größe sehr viel Kraft. Nachteile sind allerdings die hohen Preise und die vergleichweise kompliziertere Ansteuerung. Außerdem sollten diese Motoren in präzise gefertigten Produkten Anwendung finden, die möglichst wenig Reibung aufweisen und gelagerte Gelenke haben. Da wir in der Schule die Möglichkeit zur Fertigung eines solchen Produkts nicht haben und dies ebenfalls zu teuer wäre, haben wir uns entschieden diese Art der Aktorik, obwohl es die beste Wahl wäre, nicht zu verwenden. Stattdessen haben wir uns für die Verwendung von Servo Motoren entschieden, die wesentlich billiger und einfacher anzusteuern sind und trotzdem mehr als genügend Kraft haben. Der Nachteil bei diesen Motoren ist allerdings die Größe, wodurch unser Endprodukt ziemlich sicher etwas größer als gehofft ausfallen wird.

##### 7.2.2.2 Datenempfang

Um den Datentransfer möglichst einfach zu gestalten und keine Kompatibilitätsprobleme zu haben, entschieden wir uns den gleichen Mikrokontroller wie beim Eingabesubsystem zu verwenden. Dies ermöglicht eine relativ einfache Datenübertragung ohne zusätzliche Antennen. Werden Daten empfangen, so wird die Software am Chip aktiv, die in ?? näher beschrieben wird. Der Entschluss zur Verwendung dieser identischen Systeme, wurde relativ rasch getroffen, da wir genug andere Schwierigkeiten haben werden und die Funkübertragung nicht dazu zählen soll. Genauere Designanforderungen, zum Layout der integrierten Antenne, werden in Unterabschnitt 7.2.5 erklärt.

##### 7.2.2.3 Ansteuerung der Motoren

Nachdem die Daten in Software verarbeitet wurden, müssen nun die Servomotoren angesteuert

werden. Da diese über ein PWM-Signal (siehe Unterabschnitt 4.7) angesteuert werden, stellt sich die Frage ob man dies direkt über den Mikrokontroller realisiert, oder mithilfe einer PWM-Controller-Schaltung. Da wir eine Mindestanzahl von fünf Servomotoren haben und auch für zukünftige Erweiterungen bereit sein möchten, haben wir uns für die Ansteuerung via Servo-Controller entschieden.

Bei der Ansteuerung der Motoren ohne Controller, ist das Problem, dass für jeden Servomotor ein eigener Pin des Mikrocontrollers benötigt wird. Dies ist nicht gerade vorteilhaft, weil dadurch zwangsläufig andere Funktionen des Mikrocontrollers, die nur mit bestimmten Pins funktionsfähig sind, außer Kraft gesetzt werden. Um dieses Problem zu vermeiden verwenden wir einen PWM-Controller-Chip, dessen Auswahl im folgenden Punkt näher erläutert wird.

### PWM-Controller-Chip

Bei der Auswahl des Mikrochips war die Ansteuerungsmöglichkeit das größte Auswahlkriterium. Da wir bei der Schaltung des Eingabesubsystems auf die I2C-Kommunikation setzen, wollten wir dem treu bleiben und haben das gleiche bei dem gesamten Schaltungsdesign des Ausgabesubsystems ebenfalls beachtet. Das bedeutet, dass nur Mikrochips in Frage kommen, die diese Art des Übertragungsprotokolls unterstützen. Als nächstes ist die Anzahl der PWM Ausgangskanäle zu beachten und wie diese und der Baustein selbst adressiert werden können. Ebenfalls zu beachten war, dass der Chip mit 5V Versorgungsspannung zurecht kommen muss.

Die Wahl fiel schlussendlich auf den PCA9685, welcher 16 PWM-Kanäle hat und über alle anderen, soeben beschriebenen, Anforderungen verfügt. Durch die hohe Anzahl von Signal-Ausgängen bleiben wir ebenfalls für zukünftige Erweiterungen kompatibel.

#### 7.2.2.4 Positions -und Kraftmessung der Roboterfinger

Da mit der Roboterhand Objekte gegreift werden sollen, ohne diese zu beschädigen oder zu zerquetschen, mussten wir uns eine Schaltung überlegen, mit der man einen, mit der Griffkraft steigenden, Parameter messen kann. Hierfür könnte man den Winkel jedes Fingers messen, wodurch allerdings extra Sensoren notwendig wären. Außerdem ändert sich der Gelenkwinkel der Finger nicht mehr, sobald die Roboterhand etwas festhält und nur mehr stärker zudrückt. Diese Methode der Griffkraftmessung haben wir deswegen verworfen.

Stattdessen haben wir uns dazu entschieden den Strom zu messen, den jeder einzelne Servomotor beim Schließen des jeweiligen Fingers benötigt. Dies ermöglicht uns jeden Motor einzeln zu regulieren und gegebenenfalls bei einer Fehlfunktion abzuschalten.

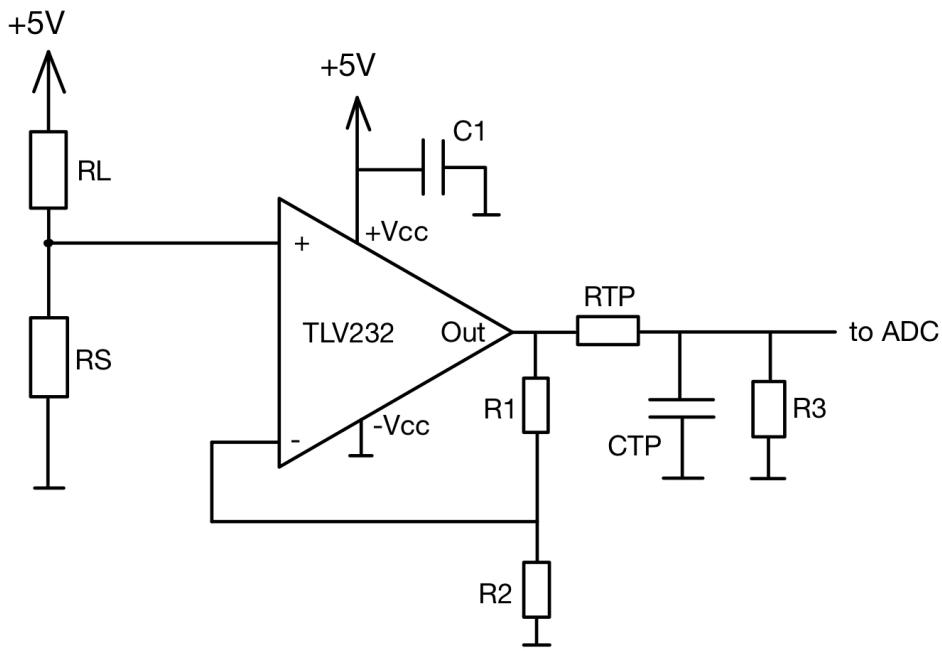


Abbildung 65: Schaltbild der Strommessschaltung

Diese Messung basiert auf dem gleichen Prinzip wie die der Flexsensorstellung in Absatz 7.1.2.2. Mithilfe eines Shunt-Widerstands wird ein Spannungsabfall gemessen, der anschließend über eine Messschaltung verstärkt und in ein Digitalsignal umgewandelt wird.

Je mehr Strom der Servomotor benötigt, um die gewünschte Position zu erreichen und somit dem Widerstand des Objekts in der Hand entgegen zu wirken, desto größer ist der Spannungsabfall am Shunt-Widerstand  $R_S$ . Der Motor wird in der obigen Abbildung mit  $R_L$  dargestellt, da eine höher Strombedarf dem senken dieses Widerstands in der Simulation gleicht.

Wird nun ein Spannungsabfall gemessen, so wird dieser durch den Operationsverstärker verstärkt. Die Verstärkung wird bei dieser Schaltung von den Widerständen  $R_1$  und  $R_2$  bestimmt. In Serie kommt anschließend ein Tiefpassfilter, um, sowie bei der Schaltung des Eingabesubsystems, ungewünschte Störungen herauszufiltern. Der Widerstand  $R_3$  entlässt den Eingang des folgenden Bauelements, nämlich einem ADC.

Folgendes Kriterium wurde bei der Wahl des Operationsverstärkers beachtet:

- Ausgangsspeicher bei gewählter Versorgungsspannung:

Zunächst wurde das Kriterium der Versorgungsspannung betrachtet. Da wir maximal 5V Gleichspannung in der gesamten Schaltung verwenden wollen, muss der OPV mit dieser geringen Spannung immer noch verstärken. Da am positiven Verstärkereingang eine maximale Spannung von 5V anliegt, muss dies bei Verstärkern mit einer geeigneten

Supply Range auch mit nur 5V Versorgungsspannung gewährleistet sein. Folglich wird die Rail-to-Rail Technologie (siehe Unterabschnitt 4.4) tragend, um keine Referenzspannung zu benötigen.

Da wir nicht bis ganz an die Grenzen des Aussteuerbereichs kommen werden (wird anhand der Berechnungen verständlich), ist der TLV232 wie für diese Anwendung gemacht.

## Dimensionierung der Schaltung

Den theoretischen Widerstand der Servomotoren, haben wir uns, anhand der im Datenblatt angegebenen Stromwerte bei 5V Versorgungsspannung, mit dem ohmschen Gesetz ausgerechnet. Dies hilft uns unter anderem auch in LTspice, da wir so verschiedene Lasten simulieren können.

$$R_{Idle} = 500\Omega$$

$$R_{noLoad} = 29.41\Omega$$

$$R_{max.Load} = 4.16\Omega$$

Der OPV, wurde zusammen mit dem Shunt-Widerstand so dimensioniert, dass am Ausgang, bei voller Belastung des Motors, annähernd 5V auftreten. Dabei sollten ca. 100mV - 200mV Sicherheitsabstand gelassen werden, da die Rail-to-Rail Technologie auch ihre Grenzen hat und nicht unendlich genau an den Versorgungspegel ausgeben kann. Dies lässt die Physik einfach nicht zu.

Es wurde ein relativ hochohmiger Shunt-Widerstand mit  $R_s = 165m\Omega$  gewählt, um die Leistung, die über den Widerstand verloren geht, möglichst gering zu halten.

Mit den folgenden Formeln wurde anschließend die notwendige Verstärkung des OPVs berechnet, um auf den gewünschten Ausgangsspannungsbereich zu kommen.

$$G = 1 + \frac{R_1}{R_2}$$

$$V_{out} = G * (V_{in+} - V_{in-}) + V_{ref}$$

Die entgültigen Werte lauten wie folgt:  $R_1 = 100k\Omega$   
 $R_2 = 6.2k\Omega$

Mit diesen Widerstandswerten ergibt sich eine Verstärkung von  $G = 17.13$ .

Da nun sichergestellt ist, dass der Ausgangsspannungspegel des OPVs bei der gewählten Verstärkung, dem gewählten Shunt-Widerstand und der maximalen Belastung des Servos 5V nicht übersteigt, kann ein Analog-Digital-Wandler ausgewählt werden, um das analoge Signal zu digitalisieren. Da wir bei der Wahl von ADCs durch das Schaltungsdesign des Eingabesubsystems schon Erfahrung gesammelt haben, viel das Aussuchen des jetzigen Wandlers nicht schwer. Entschieden haben wir uns für den identen Baustein wie bei der Sensorschaltung, jedoch mit dem Unterschied, dass dieser nun einen Aussteuerbereich von 0V - 5V hat, anstatt von 0V - 3.3V. Mit den folgenden Berechnungen wurde die Auswahl nochmals überprüft und die Stufenweite bestimmt.

Auflösung = 10 Bit, bei 0V - 5V Aussteuerbereich

$$LSB = \frac{5}{1024} = 4.88mV$$

Durch diese hohe Auflösung, kann die Griffkraft der einzelnen Finger sehr genau geregelt werden, wodurch ein feines Greifen von Objekten ermöglicht wird.

### 7.2.3 Versuchsaufbauten und Messungen Laci

#### 7.2.3.1 Messschaltung der Griffkraftkontrolle

Das Ziel der Messung ist es, herauszufinden wieviel Strom ein Servomotor, abhängig von dem gerade erzeugten Drehmoment, braucht. Die gemessenen Daten sollen uns bei Feinjustierung der Griffkraftregelung der Roboterhand behilflich sein.

Der folgende Steckbrettaufbau repräsentiert die oben gezeichnete Schaltung.

Die Grundidee ist, dass sich der Spannungsabfall am Shunt Widerstand je nach Auslastung des Servos ändert und somit der Stromverbrauch gemessen werden kann. Anhand der vom OPV ausgegebenen Werte, kann folglich in Software die Ansteuerung programmiert werden.

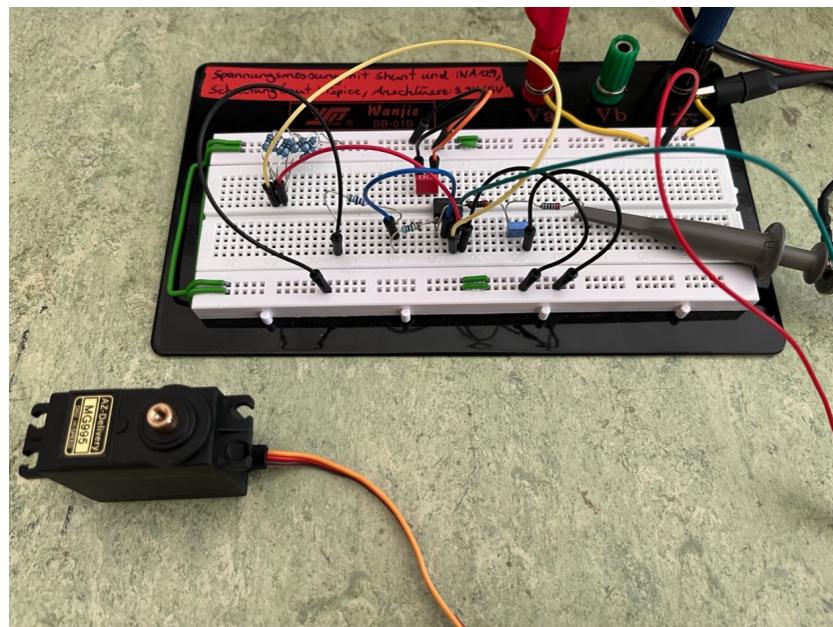


Abbildung 66: Steckbrettaufbau der Messschaltung zur Strommessung

### Erklärung des Testablaufs

Die Schaltung wird mit +5V versorgt. Der OPV „TLV232“ kann, dank RailToRail Technologie, den

Ausgang bis fast auf die Versorgungsspannung anheben. Durch die Simulation von verschiedenen Lasten am Servo mit einer Zugkraftwaage, fällt am Shunt Widerstand eine andere Spannung ab. Dadurch variiert die Ausgangsspannung des OPVs und im Programm kann ausgewertet werden, wie stark ein Finger der Roboterhand belastet wird. Je nachdem, kann dann geregelt werden. Wichtig zu beachten ist allerdings, dass diese Messung nicht die wahre Verwendung der Servos widerspiegelt. Für eine wahrheitsgetreue Ausmessung der Servobelastung, muss diese an der Hand (den einzelnen Fingern) durchgeführt werden und nicht in einem Testaufbau. Im folgenden Bild wird der gesamte Testaufbau gezeigt.

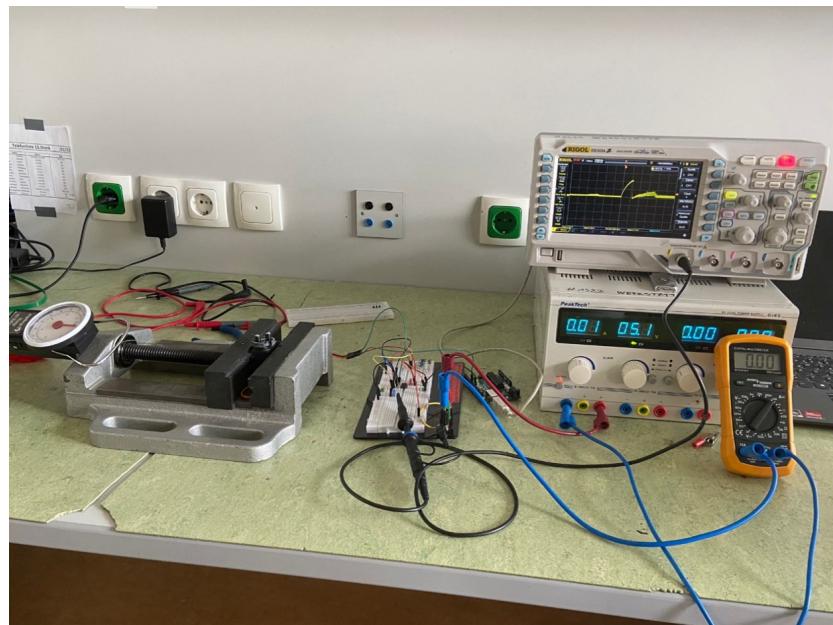


Abbildung 67: Testaufbau der Strommessung eines Servomotors

### Messergebnisse

Die gemessenen Werte wurden in einer Tabelle dargestellt und anschließend grafisch in Diagrammen visualisiert.

## 7.2 Roboterhand

---

Servomotoren (Ausmessung)						
Servomotor						
Winkel (°)	Strom (A)	max. Spannung v. Einsch. (V)	Spannung n. Einsch. (V)	Zugkraft (kg)	Zugkraft (N)	Delay (ms)
5	0,01	1,88		0,00	0,00	30
10	0,12	1,88		0,00	0,00	30
15	0,14	1,72		0,00	0,00	30
20	0,17	2,04		0,78	7,65	30
25	0,25	1,86		1,10	10,79	30
30	0,29	1,98		1,90	18,63	30
35	0,42	2,06		2,40	23,54	30
40	0,58	2,02		3,00	29,42	30
45	0,70	2,06		3,75	36,77	30
50	0,93	1,98		4,15	40,70	30
55	1,25	2,06		4,50	44,13	30
60	1,36	2,20		4,75	46,58	30
65	1,39	2,30		4,90	48,05	30
70	1,38	2,18		4,90	48,05	30
75	1,37	2,10		4,80	47,07	30
80	1,38	2,10		4,85	47,56	30
85	1,39	1,74		4,85	47,56	30
90	1,36	2,10		4,60	45,11	30
95	1,37	1,70		4,60	45,11	30
100	1,38	2,06		4,75	46,58	30
105	1,39	2,08		4,76	46,68	30
110	1,42	2,09		4,82	47,27	30
115	1,43	2,11		4,90	48,05	30
120	1,45	2,10		4,91	48,15	30

Abbildung 68: Messtabelle Strommessung

In der linkesten Spalte, ist die Servostellung, in Grad, vom Nullpunkt des Servos gemessen, abgebildet. Gleich daneben, wurde der Strom aufgetragen, der mit einem Multimeter gemessen wurde.

Da bei der Messschaltung ein Fehler aufgetreten ist, sind die Spalten *max.Spannung v. Einsch.(V)* und *Spannung n. Einsch.(V)* nicht zu beachten!

Die Zugkraft des Servomotors ist in Kilogramm und in Newton abgebildet. Rechts daneben ist noch eine Delay-Spalte zu sehen, die angibt wie lange der Servo gebraucht hat, um sich dem gewählten Winkel anzunähern. Am Anfang wurde es dem Motor selbst überlassen wie schnell er sich bewegt, jedoch führte dies zu unerwünschten Ungenauigkeiten, weshalb wir diesen Fehlerfaktor der Messung durch das Delay ausgeschlossen haben.

Ebenfalls zu erkennen ist, dass sich die Zugkraft des Motors ab der halben Vollausssteuerung des Servodrehwinkels nicht mehr ändert. Dies liegt daran, dass der 3D gedruckte Aufsatz, auf dem die Schnur zur Verbindung des Motors und der Zugkraftwaage angebracht ist, ab ca. 45N Zugkraft nachgegeben hat und in Folge auch kaputt gegangen ist. Dieser Materialfehler hat uns geholfen später im Projekt neue Aufsätze zu designen, die dem Zug des Servos standhalten.

Die Messergebnisse wurden im folgenden Diagramm visualisiert.

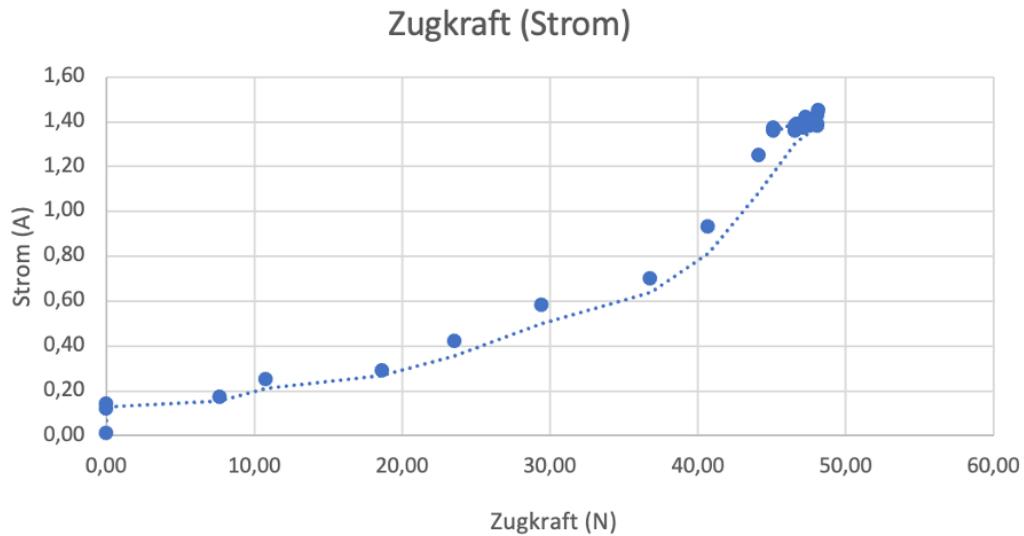


Abbildung 69: Diagramm der Strommessung in Abhängigkeit der Zugkraft

Der nicht-lineare Anstieg der Stromwerte in Abhängigkeit der Zugkraft, ist weitestgehend auf die Ungnauigkeit des Servomotors und die leichten Varianzen im Testaufbau zurückzuführen. Erwartet, beziehungsweise erhofft, haben wir uns einen möglichst linearen Ansitz, jedoch hat dies keinen wirklichen Einfluss auf die Funktion des Gesamtsystems, wenn der Kurvenverlauf in der Software berücksichtigt wird.

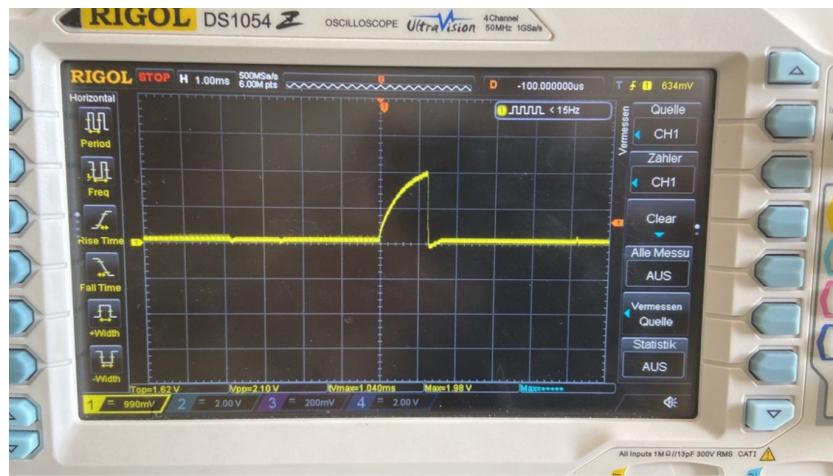


Abbildung 70: Oszilloskopbild - Stromanlaufverhalten eines Servomotors

An dem Oszilloskopbild ist zu erkennen, wie sich der Strom bei der Ansteuerung eines Servomotors verhält. Es ist deutlich sichtbar, dass der Strom nicht linear steigt, sondern sich dem Spitzenwert immer weiter annähert, bis die gewünschte Position erreicht ist. Wird kein Objekt gegriffen, so

existiert kein mechanischer Widerstand gegen den der Servo ankämpfen muss, wodurch sich der Strom wieder auf ein Minimum senkt. Würde der Roboterfinger und somit der Motor einem Widerstand entgegenwirken, so bliebe der Strom auf einem relativ konstanten Wert stehen bleiben. Dies ist der Fall, da ein Servo einen Gleichstrommotor eingebaut hat und dieser nicht nur für die Bewegung, sondern auch für das Entgegnwirken einer mechanischen Last Strom benötigt.

### 7.2.4 Schaltungsdesign **Laci**

Nach den diversen Überlegungen, Konzepten und Berechnungen, wird in diesem Punkt das genauere Schaltungsdesign erläutert. Es wird erklärt worauf bei dem Schaltungsentwurf geachtet werden musste und mit welchen Besonderheiten, beziehungsweise Problemen, wir zu tun hatten.

#### 7.2.4.1 Externe Anschlüsse

Wie auch schon bei der Schaltung des Eingabesubsystems, setzen wir auch beim Ausgabesubsystems auf einen USB-C Anschluss. Dieser dient zur Programmierung des Mikrocontrollers. Versorgt werden kann die Schaltung aufgrund des hohen Strombedarfs der Servomotoren allerdings nicht. Vorteilhaft ist dies allerdings auch, da die Platine gleichzeitig mit einem Netzteil verbunden sein kann und keine Gefahr besteht zwei Versorgungen gleichzeitig anzuschließen. **Anschluss zur Programmierung des Mikrocontrollers**

Für die Programmierung des Mikrokontroller ist ein USB-Anschluss notwendig. Da das Projekt mit den neuesten Computern kompatibel sein soll, haben wir uns, genauso wie beim Eingabesubsystem, für einen USB-C-Typ2.0 entschieden. Das Serial-Signal des angeschlossenen Computers, wird über die beiden Leitungen  $D+$  und  $D-$  übertragen. Die Anschlüsse  $CC1$  und  $CC2$  werden nicht benötigt, weshalb diese mit über einen Widerstand mit  $GND$  verbunden sind.  $SBU1$  und  $SBU2$  werden ebenfalls nicht benötigt.

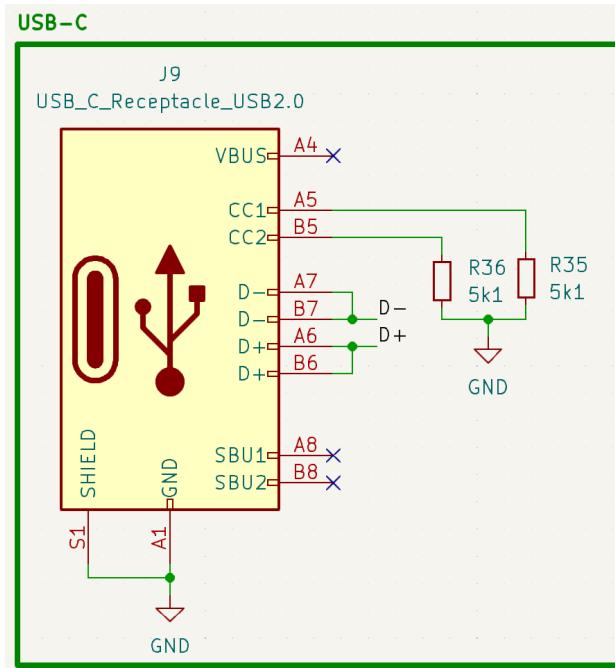


Abbildung 71: USB-C-Buchse

### Anschluss zur Versorgung der Schaltung

Da die Schaltung nicht über die USB-C-Buchse versorgt werden kann, muss dies über ein Netzteil geschehen. Für diesen Zweck haben wir einen Connector mit zwei Pins vorgesehen. Ein Pin für die positive Versorgung und der andere für den *GND – Anschluss* des Netzteils. Zusätzlich dazu, wurde noch ein Jumper verbaut, der es uns erlaubt die Schaltung von der Versorgung zu trennen ohne jedes mal das Netzteil abdrehen zu müssen. Bei der nächsten Schaltungsversion wird die Versorgung über einen Barrel Jack erfolgen.

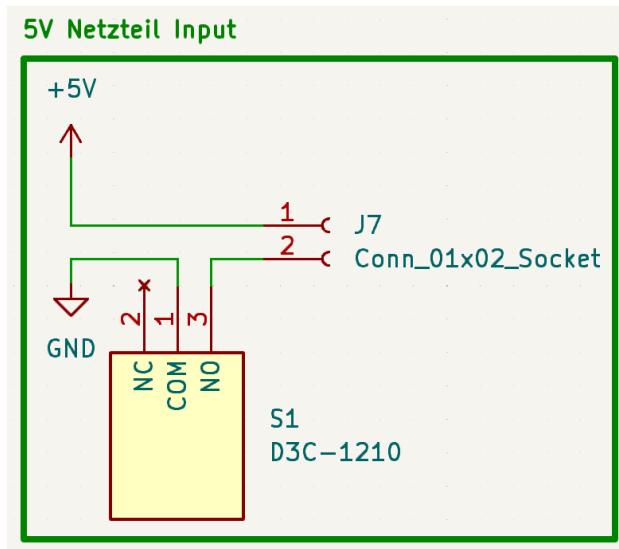


Abbildung 72: Netzteil Input Connector

#### 7.2.4.2 Serial-UART-Bridge

Der Serial-UART-Chip, hat die Aufgabe das über die USB-Buchse kommende Serial-Signal in ein UART-Signal umzuwandeln. Dies ist notwendig, da der verwendete Mikrokontroller keine Serial-Signale verarbeiten kann. Über die **ausgekreuzten** Leitungen *RXD* und *TXD*, werden die Daten zur Weiterverarbeitung übertragen. Die Versorgung basiert auf 3.3V. Die Pins *DTR* und *RTS* sind über eine Transistorschaltung ebenfalls mit dem Mikrokontroller verbunden. Die *EN* Leitung ist für die *RESET – Funktion* gedacht und die *IO0* Leitung für den Uploadvorgang von programmiertem Code.

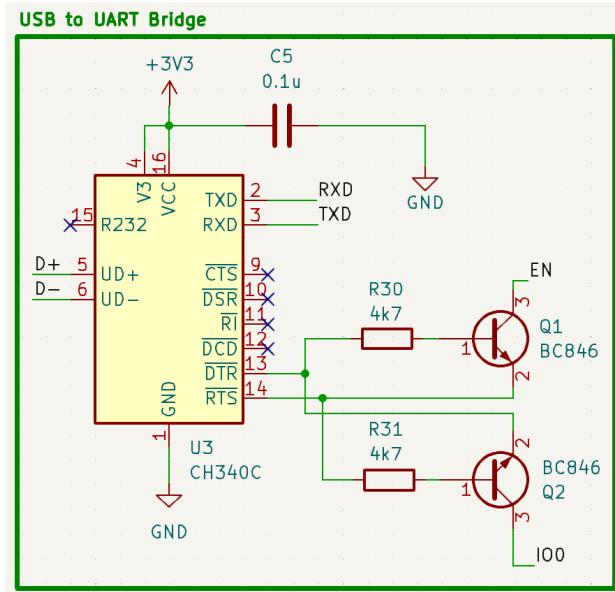


Abbildung 73: Serial-UART-Bridge

#### 7.2.4.3 Mikrokontroller

Der Mikrokontroller ist das Herz der Schaltung. Aufgrund der nötigen Kompatibilität in Bezug auf den Funk, haben wir uns, genauso wie beim Eingabesubsystem, für einen ESP32-Chip entschieden. Die geneue Bezeichnung lautet *ESP32 – WROOM – 32E – N16*. Zu diesem Chip führen unter anderem die zuvor beschriebenen Leitungen der Serial-UART-Bridge wodurch die Anbindung an die Außenwelt ermöglicht wird. Von diesem Chip gehen außerdem die I2C-Datenleitungen weg, welche mit dem ADC und dem PWM-Controller verbunden sind. Um eine möglichst störungsfreie Übertragung zu gewährleisten, wurden, außer den offensichtlichen  $10k\Omega$  Pull-Up-Widerständen, noch Serienwiderstände in den Datenleitungen beigefügt. Diese dienen dem Schutz gegen Spannungsspitzen und mindern teilweise auch die Störungen die während dem normalen Betrieb auftreten können.

Die Funktionen der weiteren Pins werden in den folgenden Punkten, gemeinsam mit den dazugehörigen Bauteilen, näher erläutert.

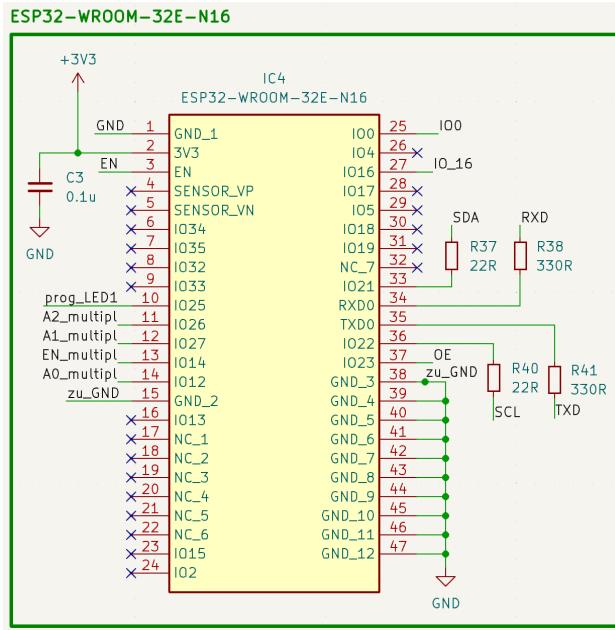


Abbildung 74: Mikrokontroller ESP32

### 7.2.4.4 Mikrokontroller Buttons

Wie auch schon bei der Schaltung des Eingabesubsystems, werden für den ESP32 jeweils ein Drucktaster für das Hochladen von Code und das Zurücksetzen des Chips benötigt. Diese Buttons sind mit dem Mikrokontroller und des Serial-UART-Bridge verbunden.

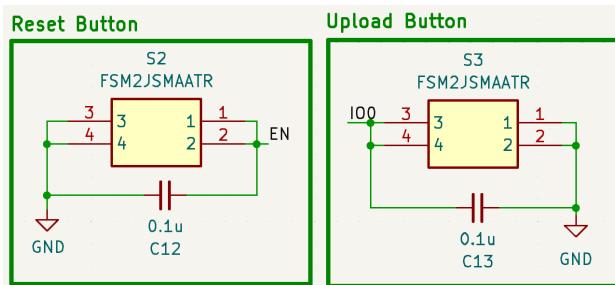


Abbildung 75: Mikrokontroller Buttons

### 7.2.4.5 Versorgung

Da nicht alle Schaltungsteile mit 5V versorgt werden können, muss die Spannung auf einen geeigneten Pegel umgewandelt werden. Hierzu verwenden wir einen Spannungswandler. Am Eingang und Ausgang des Bausteins sind Stützkondensatoren zu *GND* geschalten, um die Spannungspegel von 5V und 3.3V zu stabilisieren. Zusätzlich dazu, ist bei jedem Versorgungspin eines Schaltungsbausteins noch ein Stützkondensator vorgesehen.

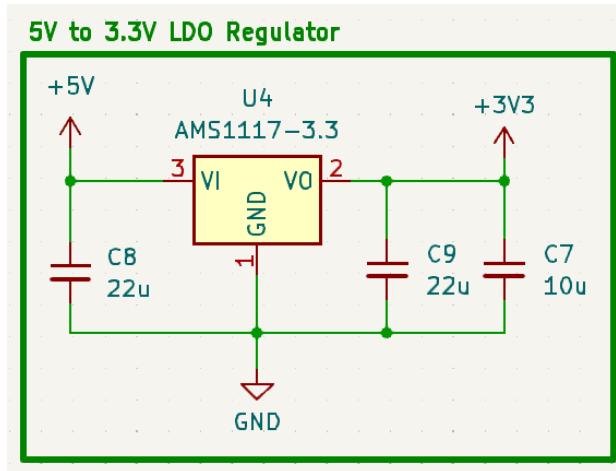


Abbildung 76: Spannungswandler 5V - 3.3V

### 7.2.4.6 PWM-Controller

Der PWM-Controller ist das Bauteil, das für die korrekte Datenübertragung der Steuerungsdaten an die Servomotoren zuständig ist. Da wir keine Funktionen und Anschlüsse des Mikrocontrollers blockieren wollen (siehe Absatz 7.2.2.3), haben wir den *PCA9685* gewählt. Dieser ist per I2C-Protokoll ansteuerbar und hilft uns daher die Anzahl der elektrischen Leitungen auf ein Minimum zu reduzieren. Um die korrekte Ansteuerung des Mikrochips zu gewährleisten, existieren natürlich auch Adresspins (*A0* - *A5*). Diese können auf *HIGH* oder *LOW* gesetzt werden, um die I2C-Adresse benutzerdefiniert einzustellen. Wir haben uns dazu entschieden diese Adressierung mit Lötjumpern einstellbar zu machen. Standardmäßig ist die Adresse auf 000000, kann jedoch gezielt durch das zulöten einzelner Jumper modifiziert werden. Der Pin *OE*, ist mit dem *ESP32* verbunden. Dieser Anschluss erlaubt uns den PWM-Output des Mikrochips, durch ein *HIGH* oder *LOW* Signal, ein -und auszuschalten. Es ist zu erkennen, dass der Chip die Möglichkeit bietet bis zu 16 Servomotoren anzusteuern. Bei unserer Anwendung werden bei der aktuellen Entwicklungsstufe allerdings nur 6 Anschlüsse benötigt. Versorgt wird der Controller mit 5V.

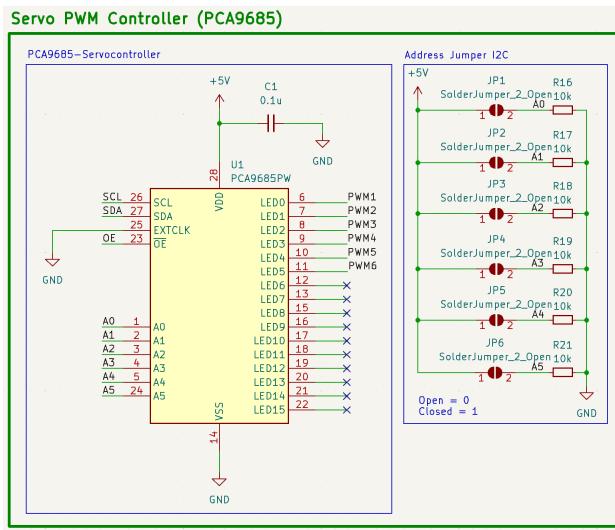


Abbildung 77: PWM Controller

### 7.2.4.7 Servoanschlüsse

Jeder Servomotor ist mit einem PWM-Pin des PWM-Controllers verbunden. Durch diese elektrischen Leitungen erhalten die Motoren ihre Drehbefehle. Um kompatibel mit dem Rest der Schaltung zu sein, haben wir die Versorgungsspannung für die Motoren mit 5V gewählt. Ein höherer Spannungspegel mit 6V wäre auch möglich gewesen, allerdings ist der Aufwand die Spannung zu erhöhen mit dem daraus resultierenden Gewinn an Zugkraft nicht zu vergleichen. Aufgrund dieses Umstands, wurde der Versorgungspegel wie beschrieben gewählt. Über den *GND – Anschluss* jedes Servomotors, wird die Strommessung (siehe Absatz 7.2.2.4) realisiert. Der Shuntwiderstandswert von  $165\text{m}\Omega$  wird, aufgrund thermischer Verluste, durch zwei parallel geschaltete  $330\text{m}\Omega$  Widerstände erzeugt, wodurch sich die Verlustleistung auf zwei Bauteile aufteilt. Die weitere Logik hinter dem *GND<sub>servo</sub> – Anschluss* wird im folgenden Punkt erklärt.

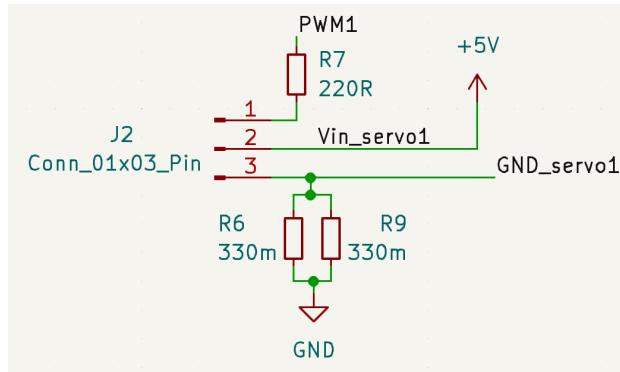


Abbildung 78: Servo Anschlüsse

#### 7.2.4.8 Strommessung zur Griffkraftkontrolle

- **Multiplexer**

Der Multiplexer ermöglicht es, die Messschaltung der Griffkraftkontrolle nur einmal bauen zu müssen. Dies ermöglicht die Fähigkeit des Bauteils, in Bruchteilen einer Sekunde zwischen 8 verschiedenen Eingangskanälen zu wechseln. Die oben dargestellten Anschlüsse für die Servomotoren, müssen für jeden Motor einzeln vorhanden sein. Dank des Multiplexers, kann jeder *GND – Pin* allerdings auch zur Messung des Spannungsabfalls am Shunt-Widerstand gemessen werden. Jede Leitung die mit dem Label *GND\_servo* beschriftet ist, wird an den Multiplexer angeschlossen, der mit dem Operationsverstärker der Messschaltung verbunden ist. Durch diese Schaltungslogik, kann jeder Motor mit nur einer einzigen Schaltung überwacht werden. Da der OPV einen sehr, sehr hohen Eingangswiderstand hat, fließt der, von den Motoren benötigte Strom, fast ausschließlich über die Shunt-Widerstände. Dies verhindert, dass der Multiplexer hohe Ströme schalten muss und somit Verluste auftreten.

Mithilfe der Pins A0 - A2, kann der Multiplexer gesteuert werden. Durch eine Bitkombination, die in der Software variiert wird, kann somit der Eingangskanal gewählt und durchgeschaltet werden. Mit den Pin *EN* kann das Bauteil deaktiviert, beziehungsweise aktiviert werden.

Der Anschluss *to\_GND\_servos*, verbindet den Multiplexer mit dem Operationsverstärker *TLV232*.

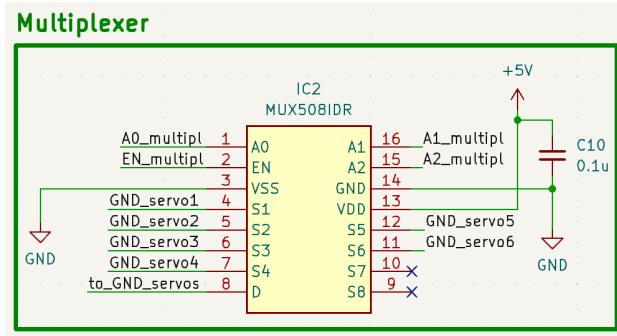


Abbildung 79: Multiplexer Schaltung

- **Operationsversätker**

Mit dem Operationsverstärker wird der Spannungsabfall am Shunt-Widerstand jedes Servos verstärkt. Der Ausgang des Multiplexers ist nun der positive Eingang des OPVs. Nach der Verstärkung des Spannungspegels, wird das Signal noch gefiltert, bevor es über das Label *to\_ADC* zum Analog-Digital-Wandler übertragen wird. Für weitere Informationen zum Schaltungsdesign, siehe Absatz 7.2.2.4

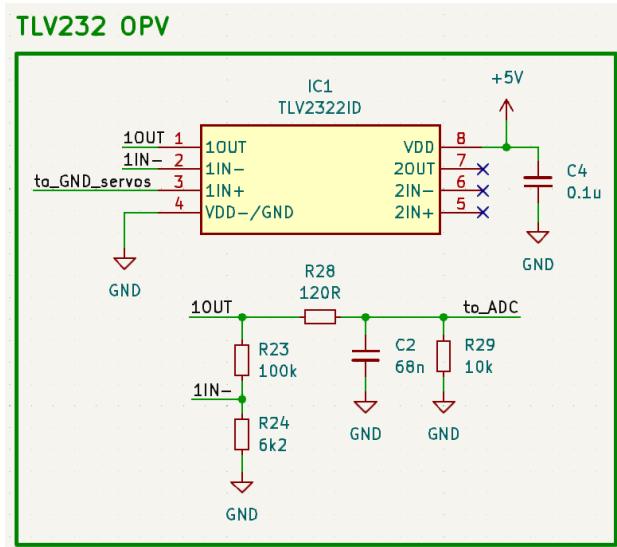


Abbildung 80: OPV Schaltung

- **Analog-Digital-Wandler**

Das Ausgangssignal des Operationsverstärkers wird nun digitalisiert, um vom Mikrokontroller verarbeitet werden zu können. Über die beiden Datenleitungen *SDA* und *SCL* werden die digitalen Daten an den ESP32 übertragen. Adresspins sind bei diesem Bauteil keine vorhanden, weshalb die Standardadresse aus dem Datenblatt zur Ansteuerung zu verwenden ist.

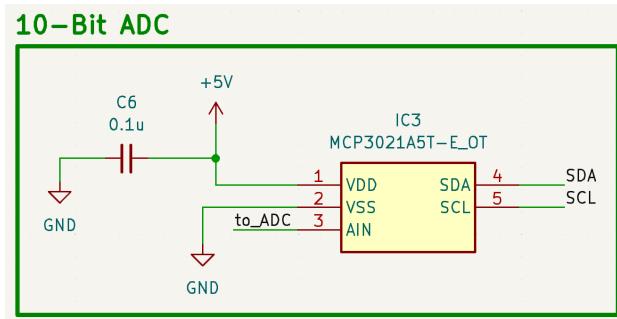


Abbildung 81: ADC Schaltung

#### 7.2.4.9 Status LEDs

Genau wie bei der Schaltung des Eingabesubsystems, soll den Benutzer des Produkts ein optisches Feedback über wichtige Parameter bekommen. Dies wird durch zwei Leuchtdioden umgesetzt. Eine leuchtet bei aufrechter Versorgung und die Funktion der anderen LED kann frei gewählt werden.

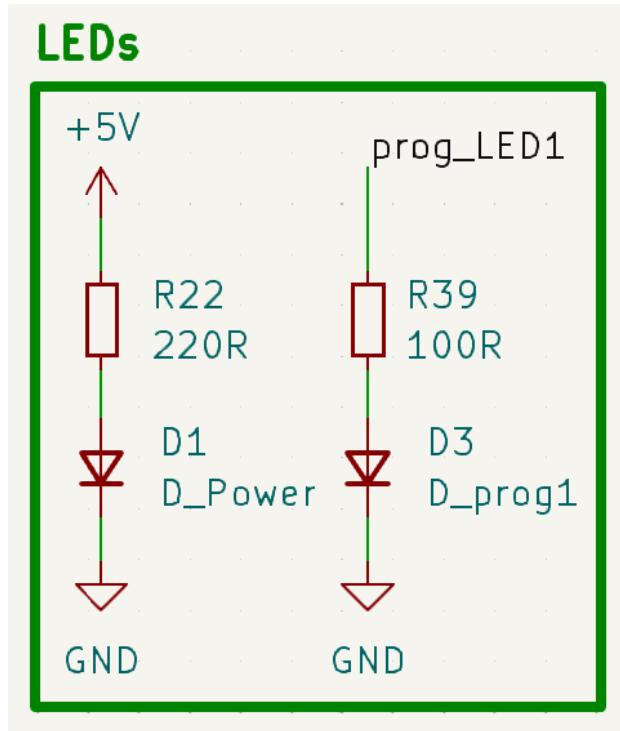


Abbildung 82: Status LEDs

### 7.2.5 Platinendesign Laci

#### 7.2.5.1 Erste Testplatine

Die erste Testplatine wurde in der schuleigenen Werkstatt, mithilfe einer Platinenfräse, gefertigt. Da das Design auch zu dem frühen Zeitpunkt des Projekts schon relativ kompliziert war, gab es bei der Fertigung einige Probleme. Aufgrund schon vorhandener Erfahrung, wurden die Bauteile mit möglichst großem Fottprint (1206) gewählt, um ein einfaches Löten per Hand zu ermöglichen. Da es ICs aber leider nur in einer, für das Bauteil festgelegten, Größe zu kaufen gibt, mussten Leiterbahnen mit einer Breite von nur 0.3mm verwendet werden. Dies führte beim Löten leider zum Ablösen dieser, weshalb einige Jumper-Kabel notwendig waren, um die kaputten Leiterbahnen zu ersetzen. Die Installation dieser Ersatzkabel, hat einige Zeit in Anspruch genommen, wodurch der Entwicklungsprozess der Hardware kurzzeitig verzögert wurde.

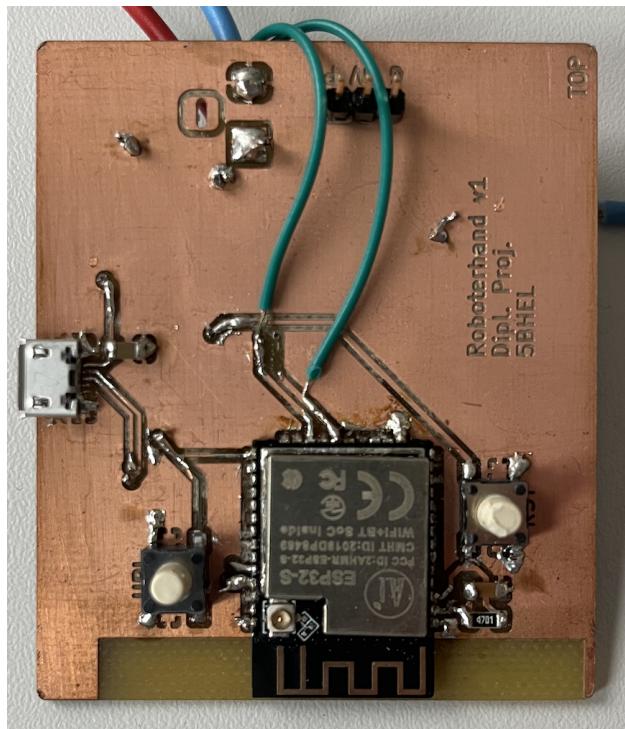


Abbildung 83: Oberseite erste Testplatine Ausgabesubsystem

Dies ist die Oberseite der ersten Testplatine. Es ist der ESP32 zu sehen. Rechts und links daneben die Taster zum Hochladen von Code und zum Zurücksetzen des Mikrocontrollers. Zu diesem Zeitpunkt der Entwicklung wurde noch eine Micro-USB-Buchse verwendet, da diese weitaus leichter zu löten ist als USB-C-Buchsen.

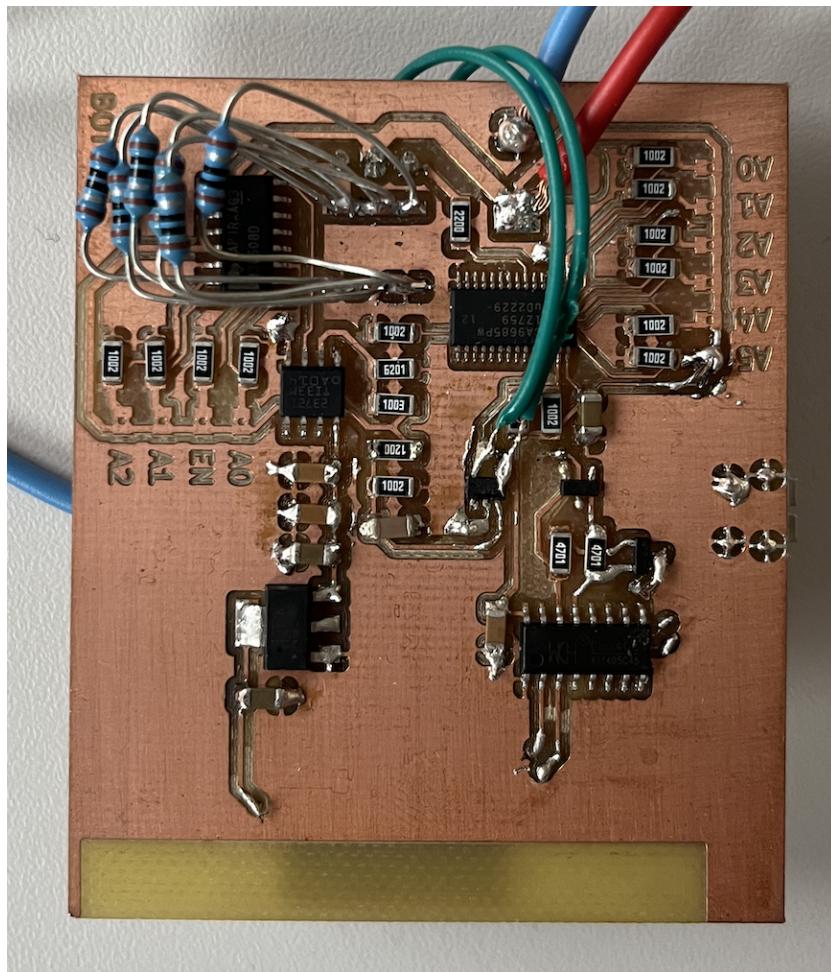


Abbildung 84: Unterseite erste Testplatine Ausgabesubsystem

Nun ist die Unterseite der ersten Testplatine zu sehen. Es kann erkannt werden, dass einige ungeplante Modifikationen vorgenommen wurden, um die Platine ein erstes mal in Betrieb setzen zu können. Unter anderem wurden zwei Pins zum ESP32 (grüne Kabel) neu verbunden und die Shunt-Widerstände mit 6, parallel geschalteten,  $1\Omega$  Widersständen realisiert, da es Lieferschwierigkeiten mit den korrekten SMD-Widerständen gab. Über das rote und blaue, etwas dickere Kabel, wurde die Platine erstmals versorgt.

Unter den grünen Kabeln ist zu erkennen, wie dünn die Leiterbahnen sein mussten, um diese mit den Pins der ICs verbinden zu können. In der rechten, oberen Ecke des Bildes, sind die Adress-Jumper des PWM-Controllers zu erkennen. Dieser wurde für Testzwecke auf dieser Platine schon integriert, obwohl nur ein Servomotor angeschlossen werden kann.

Nach dem genauen Messen der Schaltung, wurde die korrekte Funktion bestätigt und mit dem Entwurf der zweiten Platinenversion begonnen.

### 7.2.5.2 Zweite Version der Platine

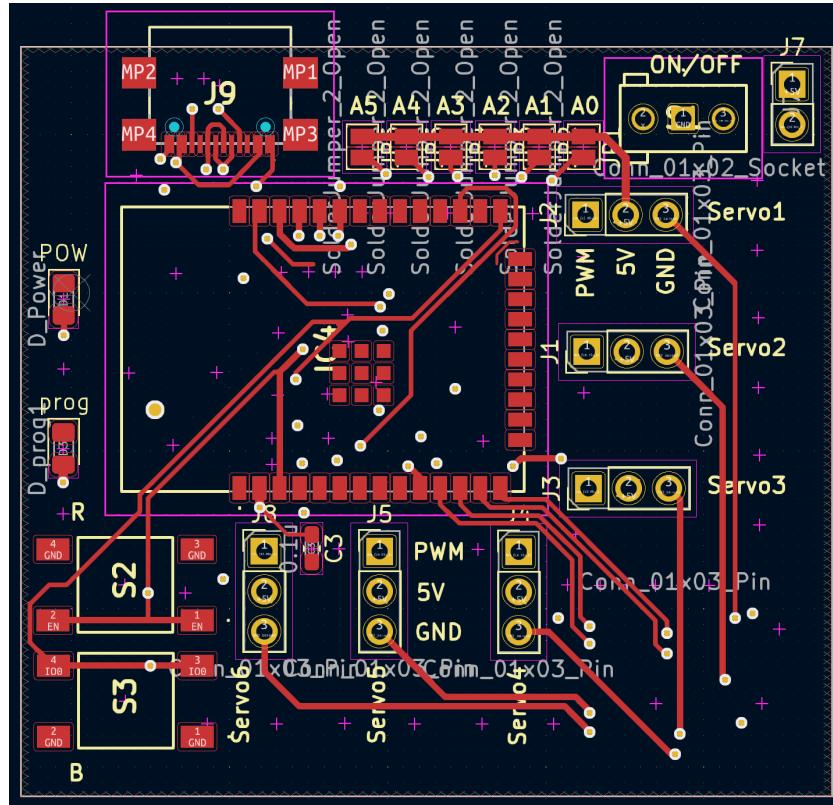


Abbildung 85: Toplayer zweites Platinendesign

In Abbildung 85, ist der Toplayer des zweiten Platinenlayouts zu sehen. Auf der Oberseite des PCBs, sind alle Komponenten angebracht, die für Wartungszwecke zugänglich sein müssen. Dies umfasst alle Anschlüsse für die Servomotoren, die Adressjumper des PWM-Controllers, den Reset- und Upload-Button und die beiden Status-LEDs zur visuellen Überwachung der Schaltung. Bei diesem Layout wurde, im Gegensatz zur ersten Testplatine, auf die Größe geachtet. Diese sollte so klein wie möglich sein, muss aber trotzdem genügend Raum für die großen Shunt-Widerstände und dicken Leiterbahnen auf dem Bottom-Layer lassen.

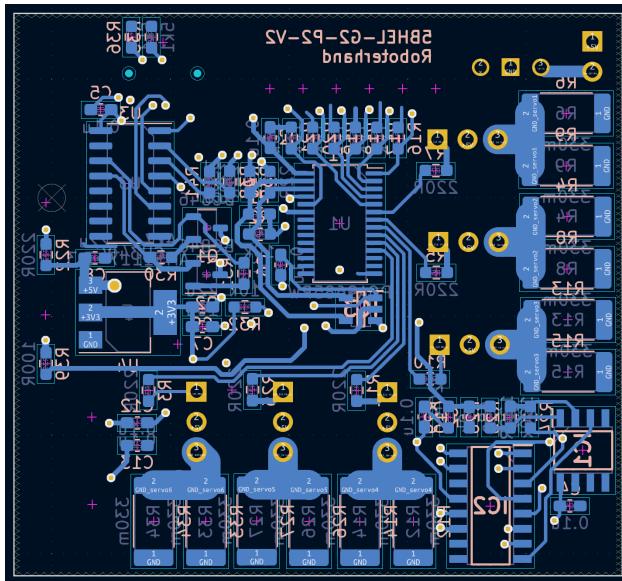


Abbildung 86: Bottomlayer zweites Platinendesign

Auf dem Bottom-Layer in Abbildung 86, wurden alle Komponenten platziert, die zur Messung und Verarbeitung der Daten notwendig sind. Am unteren und rechten Rand der Platine, sind die Shunt-Widerstände zu sehen, die aufgrund der zu vertragenden Leistung einen größeren Footprint haben. Zusätzlich dazu, sind diese nochmals parallel geschaltet, wodurch die Verlustleistung auf zwei Widerstände aufgeteilt wird. Um lange Leiterbahnen, auf denen ein hoher Strom fließt, zu vermeiden, wurden die Shunts sehr nah an den 5V Anschlüssen der Servos positioniert.

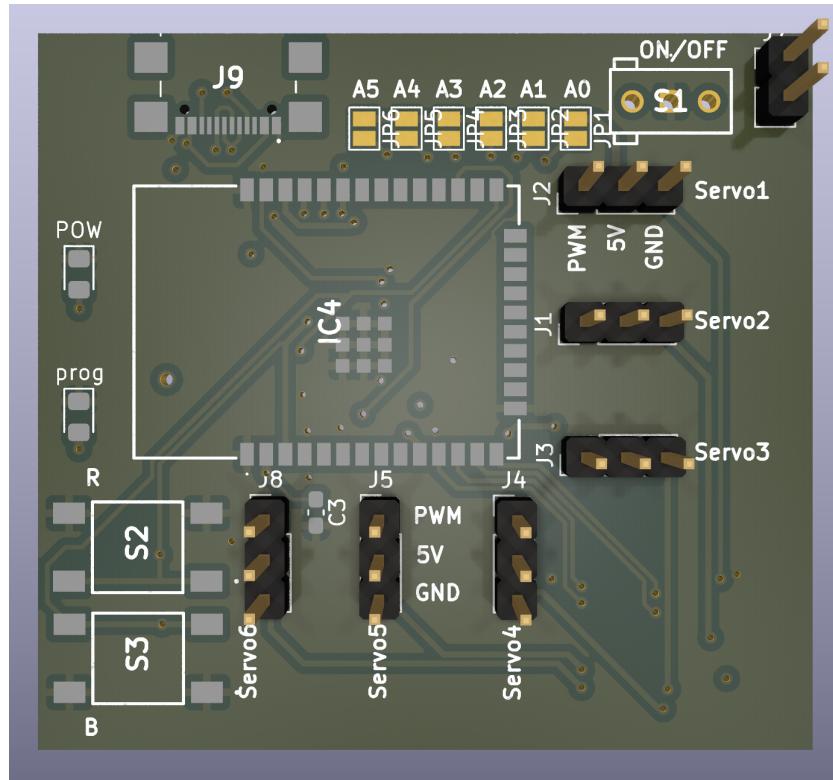


Abbildung 87: Oberseite zweites Platinenmodell

In Abbildung 87, ist das 3D-Modell der Platine für das Ausgabesbusystem zu sehen. In der rechten, oberen Ecke der Abbildung, sind die beiden Versorgungspins der Schaltung zu sehen. Diese sind bei der zweiten Platinenversion noch Pinheader, sollen später allerdings in Form eines Barrel-Jack realisiert werden. Links neben der Versorgungsanschlüssen, ist der Footprint für einen Ein-Ausschalter zu sehen, wie in Absatz 7.1.4.1 zu sehen. Alle weiteren Schaltungsteile sind in Unterabschnitt 7.2.4 näher erklärt.

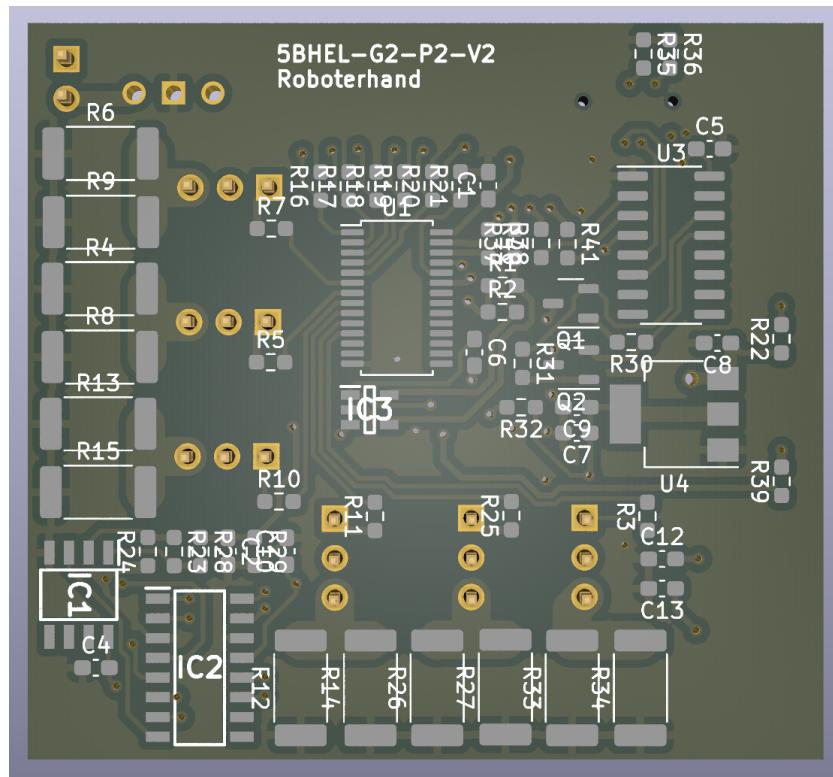


Abbildung 88: Unterseite zweites Platinenmodell

Zu sehen in Abbildung 88, ist der Bottom-Layer als 3D-Modell. Es wurde auf eine möglichst effiziente Anordnung der Bauteile geachtet, damit die Platine so wenig Platz wie möglich in Anspruch nimmt.

---

## 8 Software Realisierung

### 8.1 Handschuh

#### 8.1.1 Konzepte und Überlegungen **Fabian**

##### 8.1.1.1 Datenübertragung

Es gibt viele Möglichkeiten die Werte, die man von jedem einzelnen Flexsensor (siehe Unterabschnitt 4.1) ausliest, zu übertragen. Wichtig ist es, dass dies einfach und auf eine sehr stabile Weise funktioniert. Werden nämlich Daten fehlerhaft oder nur teilweise übertragen, dann wirkt sich das auf der Empfängerseite drastisch aus. Es könnten dadurch unerwartete Fehler passieren beziehungsweise könnte die Roboterhand von einem unstabilen System sehr hohe Schwankungen der Werte andauern wahrnehmen, was zu einer durchgängigen Belastung der Servos (siehe Unterabschnitt 4.2) führen würde. Das ist zwar nicht allzu schlimm, aber verbraucht unnötig Ressourcen. Anfangs war Bluetooth die favorisierte Option, da man im Alltag immer wieder mit Geräten zu tun hat, die Bluetooth als Standard der Funkübertragung verwenden. Allerdings haben wir uns später dann aber für Wifi (siehe Unterabschnitt 4.10) entschieden, da angenommen wurde, dass wir große Mengen an Daten versenden. Dies ist nun aber nicht nötig, da nur die Widerstandswerte, die über einen ADC (siehe Unterabschnitt 4.5) umgerechnet werden, nun versendet werden. Deshalb haben wir nach einer neuen Möglichkeit gesucht, die einfacher zu realisieren ist. Schlussendlich wurde es dann ESP-NOW (siehe Unterabschnitt 4.3), das auf 2,4GHz funk und am ehesten mit Wifi verglichen werden kann. Es können pro Sendung 250 Byte gesendet werden. Ebenso ist eine Kommunikation zwischen mehreren ESP32 in beide Richtungen möglich. Der Standard gilt allerdings wirklich nur für den ESP32, was allerdings aufgrund der Wahl von nur diesem letztgenannten, keine Probleme darstellt. ESP-NOW sendet auf dem 802.11 Protokoll (siehe Absatz 4.3.1.9), wie auch Wifi es macht. Der Vorteil liegt aber klar in der Energieeffizienz. ESP-NOW benötigt nämlich weniger Energie (vor allem, weil weniger Daten maximal gesendet werden können) als Wifi und ist deshalb gerade für unseren Zweck, die energiebetriebene Versorgung der Senderplatine, optimal. Nicht zu vergessen ist der schnellere Verbindungsaufbau. Beim Testen hatten sich beide Platinen sofort miteinander verbunden. Ein großer Vorteil ist die Peer-to-Peer Kommunikation, die zwischen den einzelnen ESP32 möglich ist. Es wird dafür kein Router oder Access Point benötigt. Dadurch, dass es auch einfach möglich ist, einzustellen, welche Platine über ESP-NOW senden, empfangen oder der Transceiver sein soll, kann man sehr leicht den spezifischen Anwendungsfall realisieren.

##### 8.1.1.2 Programmiersprache/Entwicklungsumgebung

Die Arduino IDE (1. & 2. Version) (siehe Unterabschnitt 15.4) wird als Entwicklungsumgebung verwendet, da diese für Mikrocontroller der Firma Arduino konzipiert wurde. Der ESP32 gehört auch zu den Mikrocontrollern, weshalb er ebenfalls über eigene Bibliotheken bestensfalls über die Arduino IDE angesteuert werden kann. Der Code wird in der Programmiersprache C oder C++ geschrieben, da diese für die Programmierung von Mikrocontrollern bestens geeignet ist. Mit Hilfe von verschiedenen Bibliotheken aus dem Internet wird die Implementierung von

bestimmten gewünschten Funktionen vereinfacht. Wichtig ist allerdings dabei, dass alle benötigten Funktionen auch getestet und angewandt werden können. Durch kleine Testprogramme wird also jede Funktion einzeln getestet und dann am Schluss zu einem Programm zusammengefügt, aber erst, wenn alles fehlerfrei funktioniert hat. Der Vorteil der Arduino IDE besteht darin, dass die unterschiedlichsten Möglichkeiten der Konfiguration des ESP32 darüber möglich sind. Es kann beispielsweise die Baud-Rate geändert werden. Wichtig ist, dass es auch sowohl mit der Arduino IDE der ersten und zweiten Generation ohne Probleme funktioniert, die Datei mit dem Programm auf den ESP32 zu laden.

### 8.1.1.3 Benutzerfreundlichkeit

Es ist wichtig, dass jeder, der sich ein wenig mit der Software beschäftigt, diese auch verstehen und vor allem benutzen kann. Es soll darauf geachtet werden, dass so viele Codezeilen wie möglich und nötig mit Kommentaren erklärt wird, sodass eine einfachere Bearbeitung der Software realisiert werden kann. Die Fehlerbehebung soll damit um ein Vielfaches vereinfacht werden, da man leicht abschätzen kann, wo ein Fehler liegen könnte. Zur effizienteren Erweiterung des Programmes soll es in verschiedene Blöcke aufgeteilt werden. Damit ist gemeint, dass jeder Block einzeln einmal getestet wurde, bevor dieser im endgültigen Programm in Betrieb gehen wird. Am Ende soll es möglich sein, dass nur der ESP32 per UART-Verbindung (siehe Unterabschnitt 4.9), über eine USB-C Kabel, angeschlossen wird und man das Programm nur auf diesen hochladen muss. Die optimalen Anpassungen sollen in der Standardversion des Programmes dann bereits vorhanden sein.

### 8.1.1.4 Testen

Jedes Programm muss ausführlich getestet werden. Die Tests bei der Senderplatine (Handschuh) beschäftigen sich vor allem mit dem Auslesen der Widerstandswerte der Flexsensoren. Es soll überprüft werden, ob sich die Werte in dem von dem Benutzer freiwillig ausgesuchten Bereich der map Funktion liegen. Bei dem Test soll bei dem gestreckten Flexsensor der maximale Wert angezeigt werden und bei ganz gebeugtem Zustand (maximale Biegung am Handschuh durch Fingerbiegung) der niedrigste. Es ist wichtig auch zu testen, ob der Multiplexer (siehe Unterabschnitt 4.6) überall durchschaltet und das in richtiger Weise. Nachdem dies erfolgreich implementiert wurde, sollen die Werte des ADCs überprüft werden, der hinter den Multiplexer geschaltet wurde. Wenn diese Werte ebenfalls realistisch sind, dann können diese Werte wie vorher bereits erwähnt gemappt werden und dann in einem bestimmten Format über ESP-NOW an die Empfängerplatine (Roboterhand) gesendet werden. Dabei soll überprüft werden, ob auch wirklich Daten gesendet werden. Bei erfolgreichem Empfangen der Empfängerplatine soll die Senderplatine im Serial Monitor der Arduino IDE ausgeben, dass die Daten erfolgreich gesendet und empfangen wurden.

### 8.1.1.5 Allgemeines Konzept

Die Senderplatine ist so konzipiert worden, dass Flexsensoren über Drähte an die Platine angeschlossen sind. An jedem Anschluss befindet sich eine Leiterbahn zu einem Multiplexer. Dieser soll über die Software angesteuert werden und immer wieder im richtigen Abstand durchschalten. Wenn dies richtig funktioniert, dann soll die Software die Werte des ADCs auslesen, da mit diesen dann später gearbeitet wird. Diese Werte werden dann jeweils in folgendes Format gebracht: `"$s : n : angepassterWert"` (n... Multiplexer 0 – 4; angepasster Wert... gemappter Wert des Flexsensors). Eine drahtlose Verbindung zur Empfängerplatine (Roboterhand) ist über ESP-NOW herzustellen. Wenn die Verbindung von der Sender- zur Empfängerplatine erfolgreich hergestellt wurde, dann können die Werte, die in das vorher beschriebene Format gebracht wurden, per ESP-NOW versendet werden. Als Antwort soll man im Serial Monitor sehen können, ob die Werte erfolgreich empfangen wurden.

### 8.1.1.6 Minimaler & maximaler Widerstandswert

Anfangs gingen wir davon aus, dass die Werte der Flexsensoren sehr genau ausgelesen werden können. Wir sind davon ausgegangen, dass jedes Mal die Werte je bestimmter Biegung sehr ähnlich sein werden. Grundsätzlich ist dies nicht falsch, jedoch gibt es ein Problem, das wir erst im Laufe der Zeit wahrgenommen haben. Die Flexsensoren halten nämlich nicht so gut wie gedacht auf dem Handschuh. Oftmals rutschen diese hin und her und der Wert, der sich je nach Biegung des Fingers bei der Messung variiert. Das stellt ein großes Problem dar, denn jedes Mal müsste dann ein neuer minimaler und maximaler Wert angenommen werden, was einen Mehraufwand verursacht. Wenn die Flexsensoren dann aber fest am Handschuh befestigt sind, es eine geeignete Lösung dafür gibt, dann wäre keine Einschränkung der Werte notwendig.

Es ist nun allerdings nötig, dass minimale und maximale Werte auf jeden Fall festgelegt werden. Der minimale und maximale Wert jedes einzelnen Flexsensors muss unbedingt festgelegt werden. Ohne diese Vorgehensweise würde es oft der Fall sein, dass nie der minimale oder maximale Wert erreicht wird, egal wie viel man jeden Finger zu biegen oder strecken versucht. Durch die Festlegung dieser Grenzwerte ist es möglich, dass der minimale und der maximale Wert auf jeden Fall erreicht werden. Die Werte, die dann unter dem minimal festgelegten Widerstandswert liegen, werden auf den minimal eingestellten Wert gesetzt. Jene die über dem festgelegten Widerstandswert liegen, werden auf den maximal eingestellten Wert gesetzt. Somit ist es ohne Probleme möglich, dass Werte, die ebenso als Fehlmessungen gezählt werden können, schon im Vorhinein aus dem Wertebereich, aus dem Werte später drahtlos übertragen werden, herausgefiltert werden. Viel zu hohe Werte, die daraus resultierend sind, dass die Verbindung zum Flexsensor fehlerhaft ist, werden somit automatisch gelöscht und nicht per ESP-NOW später übertragen. Man verliert einen sehr kleinen Wertebereich durch diese festgelegten Grenzwerte, allerdings ist dies nicht merkbar und reduziert deutlich Unregelmäßigkeiten und Fehlmessungen. Wenn nun die Flexsensoren zusätzlich auch noch am Handschuh verrutschen, dann wird trotzdem noch eine sehr gute Messung und valide Werte zur Übertragung möglich sein.

### 8.1.1.7 Toleranz bei Werten

Da alle Widerstandswerte der Flexsensoren auf die Stelle genau und beinahe ohne Pause ausgelesen werden, wird jede kleinste Veränderung wahrgenommen. Die Finger der Roboterhand fangen zum Zittern an. Die gesamte Mechanik wird instabil, da durch diese kleinsten Änderungen diese nur noch am Zittern ist. Das könnte zu Schäden führen. Um dies zu vermeiden, wird ein Toleranzbereich festgelegt, in unserem Fall  $+10$ . Das bedeutet, dass Änderungen des digitalen Wertes erst ab  $+10$ , im Vergleich zum zuletzt gesendeten Wert, wahrgenommen und an die Empfängerseite drahtlos übertragen werden. Dies führt zu deutlich ruhigeren und realistischeren Bewegungen der Finger auf der Empfängerseite.

### 8.1.1.8 Verworfene Ideen

Datenkorrektur auf Senderseite Ursprünglich wurde mit dem Gedanken gespielt, dass die eingelesenen Werte bereits auf der Senderseite korrigiert werden. Allerdings haben wir uns dann überlegt, dass es aus energietechnischer Sicht sinnvoller ist, wenn man es auf der Empfängerseite macht. Die Datenkorrektur hätte nach folgendem System aufgebaut werden sollen:

Die Werte jedes einzelnen Widerstands werden eingelesen. Dabei vergleicht man die ersten zehn eingelesenen Werte mit jeweils einem neuen Wert. Der arithmetische Mittelwert soll hierbei herangezogen werden. So würde man auch ein sehr schnelles Öffnen und Schließen der Hand unterbinden, da dies auch durch Fehlmessungen verursacht werden könnte. Der arithmetische Mittelwert liefert dann sozusagen eine Rampe in die eine oder in die andere Richtung, wenn man eine Faust im Handschuh machen würde. Dass der Median allerdings als viel sinnvoller betrachtet wird, das fiel uns mit der Zeit erst auf. Beim Median nehmen wir die letzten 5 eingelesenen Werte her und nehmen davon immer den Median. Dadurch wird der Wert genommen, der sich in der größentechnischen Mitte der Werte befindet. Sollte also ein extrem hoher oder extrem niedriger Wert zufällig eingelesen werden, obwohl dies nicht so sein sollte, dann wird dieser nicht bei der Ausgabe an den Servo berücksichtigt.

Die Handschuhplatine wird über einen Akku versorgt. Deshalb sollte diese Platine nur für die wichtigsten beziehungsweise nötigsten Messungen und Berechnungen verwendet werden. Der Rest kann und soll dann lieber auf der per Netzteil betriebenen Roboterhandplatine durchgeführt werden, da in diesem Fall keine begrenzte Spannungsversorgung besteht, da kein Akku verwendet wird.

### Verzögerung beim Senden

Eine Verzögerung vor dem Senden des ESP32 auf der Senderseite einzubauen war eine Idee, an der sehr lange festgehalten wurde. Es schien äußerst sinnvoll, eine gewisse selbst gewählte Zeit abzuwarten, bis man dann die Daten von der Sender- zu der Empfängerplatine sendet. Dies ist grundsätzlich keine schlechte Idee, allerdings konnte erfreulicherweise folgendes festgestellt werden: Nämlich, dass es auch ohne Abwarten eines bestimmten Intervalls möglich ist, die Daten zu senden. Der ESP32 auf der Empfängerseite kann die Daten genug schnell empfangen, ohne, dass ein bestimmter zeitlicher Abstand eingehalten werden muss. Das Beste daran ist, dass auch

die Verarbeitung der Daten ohne Probleme abgearbeitet werden kann. Dies ist möglich, da so wenige Rohdaten, wie möglich, von der Senderseite aus übermittelt werden, sodass alles auf der Empfängerseite dann verarbeitet wird. Durch sehr schnelles Senden und Empfangen der Daten, kann die schnellstmögliche Bewegung der Roboterhand realisiert werden.

### **Modus zum Einstellen des minimalen und maximalen Wertes jedes Flexsensors**

Es war ursprünglich geplant, dass beim Starten des Programmes am ESP32, der mit der Arduino IDE verbunden ist, jeder einzelne Wert der Flexsensoren jeweils eingelesen wird. Der große Vorteil davon wäre, dass es bei jedem Start des Programmes, samt Serial Monitor in der Arduino IDE, es möglich wäre, dass neue minimale und maximale Werte jedes Flexsensors ausgelesen werden. Durch einen festgelegten Befehl, den man in den Serial Monitor eingibt, könnte somit beispielsweise ein Wert des Flexsensors vom kleinen Finger gemessen werden. Für das Einstellen, welcher Finger und ob der minimale oder maximale Wert ausgelesen werden soll, hätte es dann verschiedene Befehle gegeben, die man in den Serial Monitor eingegeben hätte. Jeder einzelne minimale und maximale Wert hätte somit ausgelesen werden sollen und bis zum Neustart des ESP32 erhalten bleiben sollen.

Dies stellte sich aber als keine sinnvolle Lösung heraus. Als Erstes ist zu erwähnen, dass dieselben Flexsensoren jedes Mal zum Messen des jeweiligen Fingers genutzt werden. Es gibt keinen Wechsel dieser, es sei denn, einer geht kaputt. In diesem Fall muss für diesen dann manuell der minimale und maximale Wert ausgelesen werden. Des Weiteren verändert sich deshalb der minimale und maximale Wert jedes Flexsensors nicht. Diese beiden Grenzen werden einmalig manuell ausgelesen und nun im Programm als eigene Variable festgelegt. Dies funktioniert dann viel besser. Bei der manuellen Messung wurden die minimalen und maximalen Werte öfter ausgelesen. Es wurde dann nach unten hin ein kleiner Spielraum gelassen, sowie auch nach oben hin. Das bedeutet, dass man beim Abbiegen oder Strecken des Fingers bereits den minimalen oder maximalen Wert kurz vor dem kompletten Abbiegen und Strecken erreicht. Dieser Schutzbereich garantiert, dass der minimale und maximale Wert immer erreicht werden können und die map-Funktion somit immer gute Werte übermitteln kann.

## **8.1.2 Realisierung und Gliederung Fabian**

### **8.1.2.1 Realisierung**

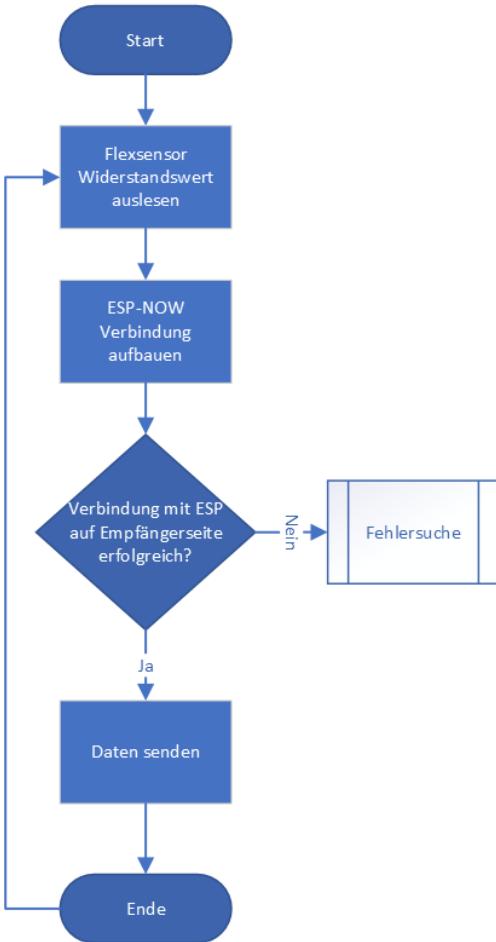


Abbildung 89: Flussdiagramm Software Eingabesubsystem

Der Code der Programmierung des ESP32 wurde in C/C++ geschrieben. Am Anfang werden die einzelnen Werte der Flexsensoren ausgelesen, die per Ansteuerung des Multiplexers und des danach sitzenden ADCs bestimmt werden. Die ESP-NOW Verbindung wird danach aufgebaut. Diese dient zur drahtlosen Datenübertragung. Wenn die Verbindung mit dem ESP32 der Empfängerseite erfolgreich war, dann werden die Daten an die Empfängerplatine (Roboterhand) gesendet. Falls dies allerdings nicht erfolgreich war, dann muss eine Fehlersuche durchgeführt werden. Ein oftmals auftretender Grund ist, dass die beiden ESPs zu weit auseinander liegen. Falls diese Schritte alle abgearbeitet wurden, dann geht das ganze Prozedere wieder von vorne los.

### 8.1.2.2 Gliederung

Am Anfang des Senderprogramms werden die zu inkludierenden Bibliotheken eingebunden. Diese beinhalten die I2C Kommunikation (siehe Unterabschnitt 4.8) für den ADC, die Möglichkeit ESP-NOW zu verwenden, indem auch zusätzlich die Wifi Bibliothek inkludiert wird.

Danach werden die ADC-Parameter festgelegt, sowie die Ports zur Ansteuerung des Multiplexers. Minimale und maximale Widerstandswerte werden festgelegt, damit es bei einer kleinen Toleranz im ganz unteren und oberen Bereich trotzdem immer den minimalen oder maximalen Wert erreicht.

Die ESP-NOW Parameter sind noch zu definieren. Also die Empfänger-Adresse, die Nummer des Flexsensors, sowie der Wert. Außerdem wird eine Funktion erstellt, die später dazu verwendet wird, um zu überprüfen, ob die Empfängerplatine (Roboterarm) die Daten erfolgreich empfangen hat.

Im Setup Teil werden anfangs die Ausgänge festgelegt, die für den ADC notwendig sind, sowie die SDA und SCL. Danach wird das ESP-NOW-Setup erstellt. In diesem Bereich werden alle notwendigen Schritte abgearbeitet, sodass man über ESP-NOW erfolgreich Daten senden kann.

In der Loop wird dann der Multiplexer angesteuert und immer wieder aufsteigend durchgeschalten. Der Wert, der vom ADC, der hinter den Multiplexer geschalten wurde, ausgegeben wird, wird dann durch eine Funktion in einen Wert umgerechnet.

Dieser Wert wird dann in Form einer Zeichenkette, die im Format `"$s : n : angepassterWert"` geschrieben wird, an den ESP32 der Empfängerplatine gesendet (n... Eingang des Multiplexers 0 – 4; angepassterWert... ausgelesener Wert (liegt zwischen Minimum und Maximum, das oben festgelegt wurde)). Es wird dann noch im Serial Monitor ausgegeben, ob die Senderplatine (Handschuh) die Daten erfolgreich senden konnte oder nicht.

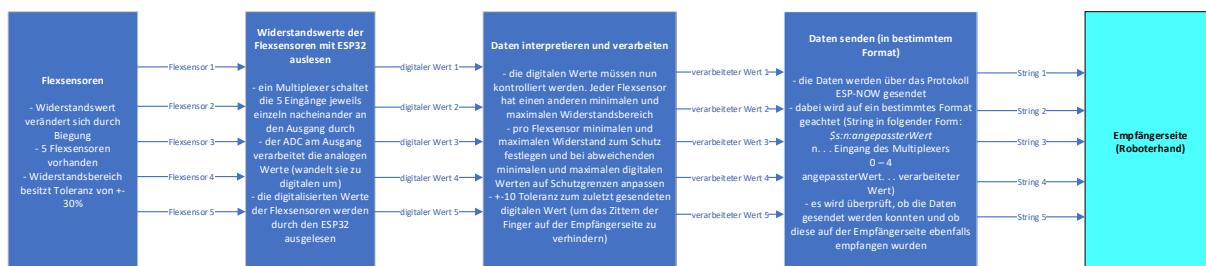


Abbildung 90: Softwaremap Eingabesubsystem

### 8.1.2.3 Wichtige Codezeilen

```

int minWiderstand[5] = {1350, 1520, 1845, 1330, 1750};
//Daumen, Zeigefinger, Mittelfinger, Ringfinger, kleiner Finger

```

```
int maxWiderstand [5] = {1900, 2000, 2135, 1900, 2050};
```

In dem obenstehenden Code kann man zwei erstellte integer Arrays sehen. Eines ist für das Speichern des minimalen Widerstandwertes zuständig. Dort werden dann die manuell gemessenen Werte eingetragen. Das Array besitzt fünf Variablen. Die Variable auf Stelle 0 des minWiderstand Arrays gibt den minimalen Wert des Daumens an. 1 gibt den jeweiligen für den Zeigefinger an, 2 gibt den jeweiligen für den Mittelfinger an, 3 gibt den jeweiligen für den Ringfinger an und 4 gibt den jeweiligen für den kleinen Finger an. Somit kann jeder Finger im Array leicht einen neuen Wert zugewiesen bekommen und das Programm arbeitet automatisch mit den neu zugewiesenen Werten bei erneutem Hochladen auf den ESP32.

```
uint16_t result_mcp = mcp3021.read();
value [ i - 1] = mcp3021.toVoltage(result_mcp , ref_voltage);

if (value [ i - 1] < minWiderstand [ i - 1]) {
    value [ i - 1] = minWiderstand [ i - 1];
}
if (value [ i - 1] > maxWiderstand [ i - 1]) {
    value [ i - 1] = maxWiderstand [ i - 1];
}
// Anpassen des Wertes
int angepassterWert = map( value [ i - 1] , minWiderstand [ i - 1] ,
maxWiderstand [ i - 1] , 0 , 100);
```

Obenstehend ist nun der Codeabschnitt zu sehen, in welchem der vom ADC verarbeitete Wert des Flexsensors von dem ESP32 ausgelesen wird. Dieser wird dann mit dem Wert des minimalen und maximalen Widerstands verglichen. Wenn der ausgelesene Widerstandswert unter jenem des minimal erlaubten ist, dann wird dieser auf den minimal erlaubten gesetzt, um grobe Fehlmessungen zu vermeiden. Das gleiche gilt für den anderen Fall, nämlich, wenn der ausgelesene Wert über dem maximal festgelegten liegt. Dann wird der ausgelesene auf den maximal zulässigen Wert gesetzt. Dies ist sehr wichtig, da ein zu hoher Wert meist indiziert, dass es keine Verbindung zwischen dem Flexsensor und den anderen Komponenten, wie dem ADC, mehr besteht. Der Wert ist dann um ein Vielfaches höher als gewünscht. Um eine Beschädigung auf Empfängerseite, beziehungsweise dem dadurch folglich falschen Ansteuern der Servos zu verhindern, ist diese Maßnahme notwendig. Die Servos könnten sich ansonsten zu weit drehen und eine Beschädigung an der Mechanik hinterlassen.

```
if( value [ i - 1] <= vergleich [ i - 1]-10 || value [ i - 1]
    >= vergleich [ i - 1]+10){
    value [ i - 1];
} else {
    value [ i - 1] = vergleich [ i - 1];
}
```

In obenstehendem Code wird der Wert des Flexsensors ausgelesen und dann mit dem vorherigen Wert verglichen. Man nimmt also eine Toleranz zur Hilfe, um ein starkes Zittern der Finger zu verhindern. Es ist wichtig dies zu begrenzen, da man bei jeder minimalsten Änderung des Wertes ein extremes Zittern der Finger beziehungsweise der Servos auslösen würde, was einen Schaden an der Mechanik auslösen kann (ganzer Roboterarm zittert extrem). Es wird also der Wert des Flexsensors ausgelesen und falls dieser nicht mindestens um 10 größer oder kleiner als der zuvor gemessene Wert ist, dann wird weiterhin der vorherige Wert herangezogen. Erst sobald der momentan gemessene Wert den zuletzt gespeicherten um 10 über- oder unterschreitet, wird der aktuell gemessene Wert dann an die Empfängerseite übermittelt, ansonsten der alte Wert, der in der Variable „vergleich“ gespeichert steht.

```
void selectMuxChannel(int channel){

    if (channel == 1) {
        digitalWrite(EN, HIGH);
        digitalWrite(A_0, LOW);
        digitalWrite(A1, LOW);
        digitalWrite(A2, LOW);
    } else if (channel == 2)
        ...
        ...
}
```

In obenstehendem Codeabschnitt wird der Multiplexer richtig eingestellt. Je nach Channelnummer wird dann festgelegt, welcher Eingang zum ADC weitergeschalten werden soll. Durch einige if-Bedingungen kann somit immer sichergestellt werden, dass man zu jeder Zeit den richtigen, gewünschten Wert auslesen kann.

```
String package = "$s:" + String(i-1) + ":" + String(angepassterWert);

Serial.println(package);

esp_err_t result = esp_now_send(broadcastAddress,
                                (uint8_t *)package.c_str(), package.length() + 1);
```

Obenstehend sieht man nun das Format, in welchem die Daten an die Empfängerseite gesendet werden. Es wird ein String geformt. Dieser beginnt mit einem \$. Dies zeigt an, dass nun ein Befehl gesendet wird und man diesen verwerten soll. Danach noch ein säls Abkürzung für Servo. Der String von der Laufvariable gibt dann an, welcher Wert nun gesendet wird, also beispielsweise \$s:1 würde bedeuten, dass nun der Wert des Daumens gesendet wird. Der gibt dann eine Trennung an. Auf der Empfängerseite muss erkannt werden, wo die Trennung zwischen der Nummer des Fingers und des Wertes, zu dem sich der Servo dann hinbewegen soll, liegt. Der String von

## 8.1 Handschuh

---

angepassterWert gibt dann den Wert an, den die Servoposition dann endgültig erreichen soll. Dabei ist es wichtig, dass dies allerdings auf der Empfängerseite dann umgesetzt wird. Auf der Senderseite wird der angepasste Wert zuvor möglicherweise durch die Grenzen von minimalem oder maximalem Widerstandswert begrenzt und dann noch mit der map-Funktion auf Werte von 0-100 gebracht, da damit dann auf der Empfängerseite gearbeitet wird. Gesamt schaut der String für den Zeigefinger mit dem angepassten Wert und einem halben Abknicken dieses also wie folgt aus: \$s:2:50: Schlussendlich wird dann der gesamte String namens package über drahtloser Übertragung über ESP-NOW von der Sender- zu der Empfängerplatine gesendet.

## 8.2 Roboterhand Amir

### 8.2.1 Grundlegende Voraussetzungen

Die Software des Roboterarmes fungiert als Herzstück der mechanischen Roboterhand und legt damit eine Brücke zwischen den auf dem Handschuh erfassten humanoiden Bewegungen und der Hand des Roboters. Empfangene Daten, sei es auf drahtlosen oder direktem Wege, müssen interpretiert, analysiert und bei Gegebenheit auch korrigiert werden. Die Software soll in der Lage sein, Fehler frühzeitig zu erkennen und entsprechend auf diese zu reagieren. Ein Zittern oder ungewünschte Bewegungen der Fingern sollen durch diese Algorithmen verhindert werden. Da es zwei Steuerungsmöglichkeiten gibt, per Handschuh oder Userinterface, muss automatisch zwischen den gewollten Modis unterschieden werden. Eine intelligente Strommessung der Servoaktoren erweitert dieses System und liefert dem Userinterface relevante Daten.

### 8.2.2 Realisierung: Überblick

Die Vielfalt der Software fordert Struktur und eine systematische Gliederung. Empfangene Daten durchlaufen mehrere Strecken, an denen diese dann verarbeitet werden, um letztlich dann die gewünschte Bewegung des Fingers umgesetzt.

#### 8.2.2.1 Programmfluss

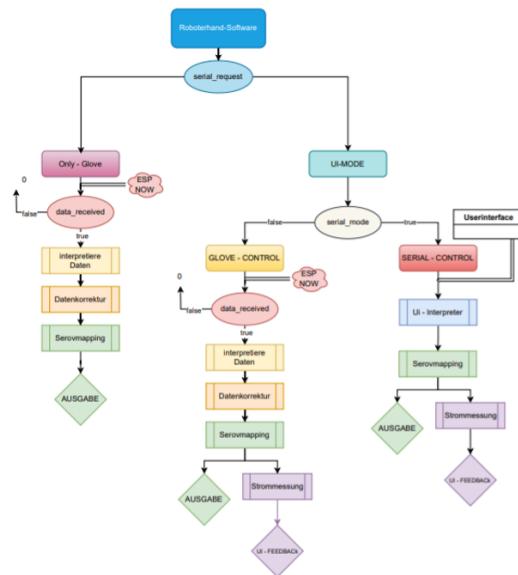


Abbildung 91: Programmfluss1

Das Diagramm in ?? veranschaulicht den Fluss empfangener Daten, diese durchlaufen bedingungsabhängig verschiedene Routen und werden auf diesen verarbeitet und interpretiert.

Zu Beginn der Software muss geklärt werden ob ein seriell verbundenes Userinterface erkannt wurde oder nicht. Ermöglicht wird dies durch eine Anfrage, welche die Roboterhand auf den Seriellen Monitor schreibt. Dies erfolgt durch den String „\$req.UI“, welcher vom Userinterface als Anweisung zum Antworten interpretiert wird. Antwortet das Userinterface wie erwartet versetzt sich die Roboterhand in den Userinterface-Modus, in der Grafik als „UI-MODE“ gekennzeichnet. Erfolgt keine Antwort, konfiguriert sich die Hand ausschließlich darauf, auf Daten welche über das drahtlose ESP-NOW Protokoll empfangen werden zu reagieren. Der Modus in dem die Hand ist wird als „ONLY – GLOVE“ in dieser Grafik bezeichnet.

Ist die Hand im Userinterface-Modus ist dennoch eine Kommunikation und Steuerung über die drahtlos verbundene Roboterhand möglich. Die Software unterscheidet somit ob ausschließlich die Hand oder beides, das Unterinterface und die Hand, genutzt wird. Wird beides genutzt gibt es jedoch auch ein Feedback an das Userinterface, beispielsweise Messdaten des Stromflusses.

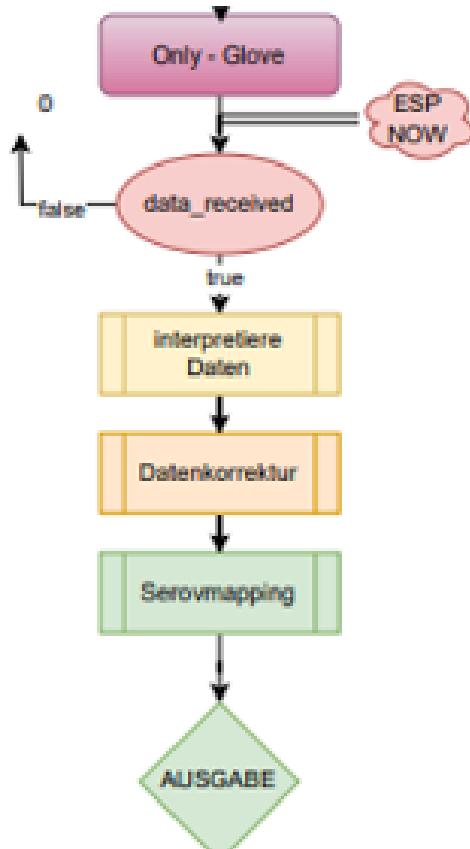


Abbildung 92: Programmfluss2

### ONLY-GLOVE Modus:

Dieser Modus ist im Vergleich zum zweiten einfacher und besteht nur aus einer „Datenroute“.

Die über das ESP-NOW Protokoll empfangenen Daten, welche die erfassten Fingerpositionen mitteilen, werden auf dieser Route interpretiert und verarbeitet. Im folgendem Kapitel (8.2.5.3 ESP-NOW und Wifi.h) wird die Anwendung dieser Komponente in dem System detailliert erläutert.

Wurden Daten empfangen wird die Zustandsvariable (data\_received) auf den Wert (true) gesetzt und alle erfassten Daten werden in einem String gespeichert. Dieser String wird daraufhin an die Komponente „interpretiere Daten“ vom Kapitel (8.2.4.1 Parser und Interpretation: drahtloser Datenströme) weitergegeben, dort werden diese interpretiert. Die extrahierte Fingerposition sowie der Index werden an die Komponente „Datenkorrektur“ übergeben, dort wird anhand eines Algorithmus die Sinnhaftigkeit der Daten überprüft. Eine genauere Erläuterung dieses Algorithmus ist in dem Kapitel (8.2.4.3 proaktiver Datenkorrekturalgorithmus) zu finden.

Wurden die Daten überprüft und je nach Situation auch korrigiert wird dies an den Block „Servomapping“ übergeben, beschrieben unter (8.2.4.4 Datenaufbereitung und Anweisung für Servoaktoren), dort wird der Positionswert des Fingers noch auf den Motor und seine Zugcharakteristik gemappt. Letztlich wird dann die Position an dem Motor übergeben.

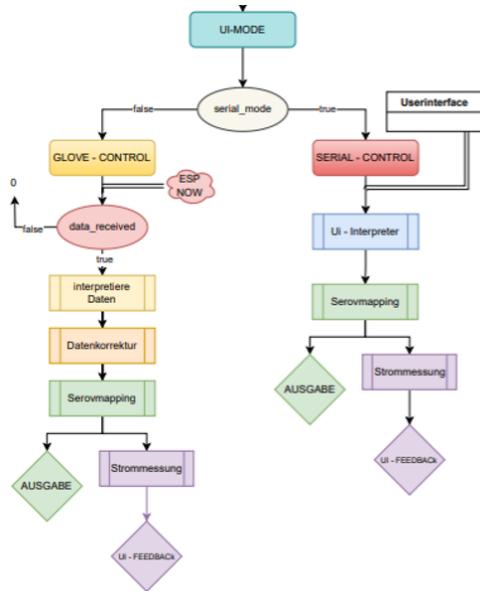


Abbildung 93: Programmfluss3

### UserInterface Modus:

Dieser Modus ermöglicht es die Hand über jeweils das Userinterface und auch den Handschuh zu steuern, jedoch getrennt voneinander. Oft wird dieser Modus missverstanden, da interpretiert wird, dass er ausschließlich eine Steuermöglichkeit über das Userinterface bietet.

Welche Steuerungsmöglichkeit gewählt wird kann über das Interface per „Switch-Button“ eingestellt werden. Das Userinterface sendet bei der nächsten Anfrage von der Roboterhand-Software die Information mit auf welchen Modus der Schalter gestellt wurde.

Erhält das Userinterface eine Anfrage antwortet dieses entsprechend mit - §answ.here und fügt daraufhin je nachdem welcher Modus im Userinterface gewählt wurde ein .han, für die Steuerung per Handschuh, oder ein .ser, für eine Steuerung per Interface, an diesen String an. Die Antwort vom Userinterface somit ausschließlich „\$answ.here.ser“ oder „\$answ.here.han“ sein. Im Kapitel (8.3.4 Realisierung des Frontends) sind zusätzliche Beschreibungen und Veranschaulichungen des Interfaces zu finden.

Wird nun über das Userinterface die Anweisung mitübergeben, dass die Steuerung ausschließlich per Handschuh erfolgt, werden alle Empfangenen Befehle vom diesem ignoriert, außer die Anweisung die Steuerungsoption zu wechseln. Der Ablauf in dieser Option gleicht dem des im Only-Glove Modus, die drahtlos empfangenen Daten werden interpretiert, daraufhin geprüft und eventuell korrigiert und letztlich gemappt bis sie dann an den Motor übergeben werden. Zusätzlich wird direkt nachdem die Anweisungen an den Motor übergeben wurden, eine Strommessung gestartet – wessen Messergebnisse in der Form einer Rückmeldung an das Userinterface über die Serielle Schnittstelle gesendet werden.

Entscheidet sich der Nutzer nun doch die Roboterhand mit den vom Userinterface gebotenen Bedienungsmöglichkeiten zu steuern, wird ebenso bei der nächsten erhaltenen Anfrage diese Änderung übergeben. Wird diese Änderung erkannt, reagiert die Software auf alle vom Interface übergeben Positionsanweisungen. Die Anweisungen landen in erster Hand in der Komponente „UI-Interpreter“ dort werden diese Anweisungen in ihre Parameter geteilt und das Programm übergibt diese an die folgende Komponente des „Servomappings“. Unmittelbar nachdem die angepassten Werte dem Motor übergeben wurden, wird eine Strommessung von der Software aus gestartet. Diese sendet die erfassten Werte des Messfensters an das Userinterface.

### 8.2.2.2 Effiziente Gliederung essentieller Systemkomponenten

Der Programmcode wurde modular in Form einzelner Komponenten realisiert, anhand dessen wird die Möglichkeit geschaffen diese mehrfach zu verwenden und die Komplexität der Software zu minimieren. Die bereits beschriebenen Routen welche von einem Datenpacket durchlaufen werden, setzen sich aus diesen repetitiven Komponenten zusammen.

Die einzelnen Komponenten sind getrennt von dem Hauptprogramm und sind somit nicht in diesem vorhanden, jedoch werden sie mit einer .h Datei verknüpft. In Form einer Funktion können diese implementiert werden.

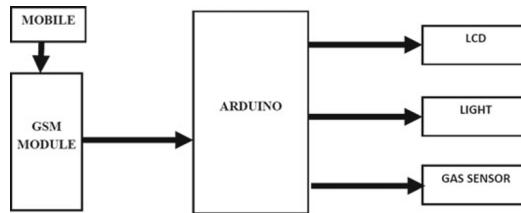


Abbildung 94: effiziente Gliederung

Diese Struktur des Programms bringt zahlreiche Vorteile mit sich, unter anderem, dass sich wiederholende Code-Abschnitte nun anhand der Gliederung mit einer Komponente effektiv implementieren lassen. Aber auch die Übersicht spielt eine große Rolle, das Hauptprogramm ist bereits äußerst überfüllt mit Konfigurationsmöglichkeiten für beispielsweise die drahtlose Kommunikation beider Subsysteme.

### 8.2.3 Kommunikation - Protokolle

Für den Datenaustausch zwischen dem Eingabe- und dem Ausgabesubsystem wird ein einheitliches Protokoll benötigt, welches auch während der Entwicklung ohne große Mühe interpretiert und verstanden werden kann. Dabei erleichtert ein einheitliches Protokoll die Realisierung der Komponenten welche für das Interpretieren der Datenzuständig sind. Die Funktionalität beider Parser wird im Kapitel (8.2.4.1 Parser und Interpretation: drahtloser Datenströme) und (8.2.4.2 Parser und Interpretation: serieller Datenströme).

#### 8.2.3.1 Drahtlose Kommunikation

Unter der drahtlosen Kommunikation kann die zwischen dem Eingabesubsystem und dem Ausgabesubsystem, welches über ESP-NOW erfolgt, verstanden werden. Übergeben werden die Datenpakete auf dieser Strecke unidirektional, von dem Eingabesubsystem zu dem Ausgabesubsystem. Ein Datenpaket enthält immer jeweils die Position eines Fingers und dessen Index. Unter der drahtlosen Kommunikation kann die zwischen dem Eingabesubsystem und dem Ausgabesubsystem, welches über ESP-NOW erfolgt, verstanden werden. Übergeben werden die Datenpakete auf dieser Strecke unidirektional, von dem Eingabesubsystem zu dem Ausgabesubsystem. Ein Datenpaket enthält immer jeweils die Position eines Fingers und dessen Index.

Beispiel für ein Paket:

$\$S : 1 : 50$

$\$$  Ist der primäre Präfix von jedem gesendeten Datenpaket und die Grundvoraussetzung, dass der jeweilige Parser es überhaupt interpretiert.

Das folgende „S“ steht in diesem Fall für Servo und deutet daraufhin, dass nun ein variabler Wert folgt welcher den Index des jeweiligen Motors angibt. Dieser kann in diesem Fall von 0-4 variieren und wird mit einem „:“ getrennt.

Es ist ein verpflichtender Teil des Protokolls, dass alle variablen Werte auf diese Weise getrennt werden.

Nach dem nächsten „:“ muss nun ein Wert zwischen 0 und 100 folgen. Dies ist ein gemappter Wert, welcher von der Software dann auf die Zugcharakteristik der Motoren rückgemappt wird, und gibt die Fingerstellung an.

Das Paket aus dem Beispiel bedeutet somit: „Der Servomotor-1 soll seinen Finger auf eine Faltung von 50% stellen“.

### 8.2.3.2 Serielle Kommunikation

Die Serielle Kommunikation beschreibt den Datenverkehr zwischen dem Ausgabesubsystem und dem Userinterface. Aufgrund der vielfältigen Funktionen und auch der bidirektionalen Kommunikationslinie muss das Protokoll in mehrere Abschnitte unterteilt werden. Darunter Fingersteuerungen, Gesten und das Feedback.

Fingersteuerungen:

Finger	Index	Wert (in %)	Format
Daumen	Th_S	0 bis 100	\$Th_S:Wert
Zeigefinger	In_S	0 bis 100	\$In_S:Wert
Mittelfinger	Mi_S	0 bis 100	\$Mi_S:Wert
Ringfinger	Ri_S	0 bis 100	\$Ri_S:Wert
Kleinerfinger	Pi_S	0 bis 100	\$Pi_S:Wert
Alle Finger	Al_S	0 bis 100	\$Al_S:Wert

Dies ist ein Überblick des Protokolls welches beschreibt wie das Userinterface Anweisungen an das Ausgabesubsystem sendet. Das Format ist wie folgt: \$FINGERINDEX:WERT, der Index des Fingers über gibt welcher Finger gemeint ist, der Wert besagt auf welche Position sich der Finger stellen soll. Es gibt auch eine Anweisung, welche alle Finger ansteuert, diese könnte beispielsweise wie folgt aussehen: \$Al\_S:50 und würde die Finger auf 50% Faltung stellen.

Neben dem einzelnen Steuern der Finger soll auch die Möglichkeit gegeben werden Gesten an die Hand zu übergeben.

Geste	Index	Format	Beschreibung
Standby	St_B	$\$G : V_1, V_2, V_3, V_4, V_5$	Eine Stellung aller Finger auf 50%
Thumb Up	Th_B	$\$G : V_1, V_2, V_3, V_4, V_5$	Die Geste Daumen hoch
Peace	Pe_B	$\$G : V_1, V_2, V_3, V_4, V_5$	Friedensgeste - Peace
Open Hand	Op_B	$\$G : V_1, V_2, V_3, V_4, V_5$	Hand öffnen
Yeah	Ye_B	$\$G : V_1, V_2, V_3, V_4, V_5$	Rockergeste
Fist	Fi_B	$\$G : V_1, V_2, V_3, V_4, V_5$	Eine faust formen

Das Format der Geste beschreibt sich wie folgt:  $\$G : V_1, V_2, V_3, V_4, V_5$  – Das G sagt dem Interpreter das dies eine Anweisung ist welche eine Geste beschreibt, darauf folgend fünf Werte von 0 bis 100 welche als V1 bis V5 benannt wurden, diese geben die Position in Prozent jedes einzelnen Fingers an.

Aufgrund des bidirektionalen Datenverkehrs zwischen dem Ausgangssubsystem und dem Userinterface muss nun auch ein Protokoll für das Feedback geschaffen werden.

An das Userinterface werden bei aktuellstem Entwicklungsstand ausschließlich Daten der Strommessung gesendet.

Dies erfolgt in folgendem Format:

$\$A:INDEX:WERT$

„A“ bedeutet für das Userinterface, dass nun mit einem Messerwert der jeweiligen Ströme gerechnet werden muss. An der Stelle von „INDEX“ ist ein Wert von 0 bis 5 erlaubt, welcher angibt zu welchem Motor der Strom zuzuordnen ist. Darauf folgt, dann der Messwert „WERT“.

### 8.2.4 Individuelle Komponenten

Wie in dem Kapitel (8.2.2.2 Effiziente Gliederung essenzieller Systemkomponenten) bereits erläutert, setzt sich das Hauptprogramm aus mehreren Unterprogrammen zusammen, diese werden als „Komponente“ bezeichnet und bringen zahlreiche Vorteile mit sich. Unter anderem ermöglichen diese, dass sich die jeweiligen Routen, die ein Datenpacket im Hauptprogramm durchlaufen kann, aus den einzelnen Komponenten zusammensetzen. Dadurch wird für mehr Überblick und Effizienz im Programm des Ausgabesubsystems gesorgt. Unter „individuellen Komponenten“ können eigens entwickelte Unterprogramme verstanden werden, welche in diesem Fall Korrekturalgorithmen oder Parser bzw. Interpreter sind.

#### 8.2.4.1 Parser und Interpretation: drahtloser Datenströme

Die Interpretation drahtloser Datenströme findet auf zwei Datenrouten im Programm ihre Anwendung. Einmal im „ONLY-GLOVE“ Modus und auch im Userinterface Modus, sofern dieses auch auf den „GLOVE“ Modus gestellt wurde.

```
void interpretiereDaten(rueckgabe_st * rueckgabe, const uint8_t * data, int data_len)
```

Die Systemkomponente erhält drei Parameter, der erste beinhaltet einen Zeiger auf die Struktur `rueckgabe_st`. Dieser ermöglicht es die später extrahierten Daten in einer Struktur zu speichern und diese im Hauptprogramm zu nutzen. Der nächste Parameter beinhaltet einen Zeiger auf den Speicherort des empfangenen Datenpakets. Der dritte Parameter gibt die Länge des Datenpakets an.

```
if (data_len >= 5 && data[0] != '$' && data[1] != 's' && data[2] != ':') {
    rueckgabe->servoIndex = data[3] - '0';
    rueckgabe->grad = atoi((char*)&data[5]);
}
else {
    Serial.println("Ungültige -Daten- erhalten");
    return;
}
```

Das Unterprogramm prüft ob die Grundbedingungen, somit das erforderte Format wie in Kapitel (8.2.3.1 Drahtlose Kommunikation) beschrieben, erfüllt wird.

Ist dies der Fall wird in die Struktur „Rückgabe“ der extrahierte Servoindex und die gesendete Fingerposition gespeichert.

### 8.2.4.2 Parser und Interpretation: serieller Datenströme

Sofern das Userinterface angeschlossen ist und eine Steuerung über das Interface vorgibt, kommt diese Komponente zum Einsatz. Ihr Zweck ist es die über die serielle Schnittstelle empfangenen Daten zu parsen. Wie in dem Kapitel (8.2.3.2 Serielle Kommunikation) beschrieben, hat dieser Parser mit Datenpaketen zu rechnen, die umfangreichere Daten enthalten. Zu einem Anweisungen für einzelne Finger, sowie auch Gesten.

Platzhalter für Amir

### 8.2.4.3 proaktiver Datenkorrekturalgorithmus

Aufgrund der Volatilität der Flex-Sensoren, sowie auch der Verdrahtung, welche bei Bewegung eines Fingers mitbelastet wird, besteht eine Chance dass es zu fehlerhaften Messwerten kommt. Um das Eingabesubsystem möglichst performant und einfach zu halten wird ein Filter und Prüfsystem auf das Ausgabesubsystem ausgelagert. Benötigt wird ein Algorithmus welcher Fehler erkennt und diese unmittelbar sinnhaft korrigiert.

Der proaktive Datenkorrekturalgorithmus basiert auf dem Prinzip der Verzögerung und lässt empfangene Daten einen Prozess durchlaufen, bis diese korrigiert und an den Motor übergeben werden.

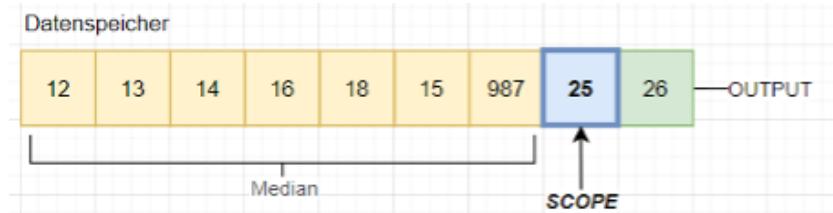


Abbildung 95: Median

Im Fokus des Algorithmus liegt ein Array, ein zusammenhängender Speicherblock, in diesem werden alle Empfangenen Daten chronologisch gespeichert. Jedem der Fünf Finger ist solch ein Array zugeordnet, welches beispielsweise die letzten zehn Fingerpositionswerte beinhaltet.

Parst das Ausgangssubsystem einen neuen Wert, landet dieser im Array des Fingers und alle bereits vorhandenen Wert werden um eine Position nach rechts verschoben.

Die Elemente des Arrays haben jeweils einen eigenen Nutzen. Aus dem ersten bis zum siebten Element wird bei jeder Iteration ein Median berechnet und in einer externen Variable gespeichert. Das achte Element ist der sogenannte „SCOPE“, dieser Wert ist die zweite Komponente einer Vergleichsoperation. Das neunte Element ist das letzte des Speichers, in diesem befindet sich der am aktuellsten geprüfte und je nach Umständen auch korrigierte Wert.



Abbildung 96: Median

In der Abbildung 96 werden mehrere Iterationen gezeigt, welche die Fehlererkennung und Eliminierung dieses veranschaulichen. Gezeigt wird hier derselbe Speicher, chronologisch wird gezeigt, wie von oben nach unten sich der Speicher pro Iteration ändert. Bei 1. ist zu sehen, dass sich im Scope der Wert „25“ befindet. Der berechnete Median beträgt 15.

Nun erfolgt eine Vergleichsoperation zwischen dem Wert im Scope und dem Median, überschreitet deren Differenz einen maximalen Grenzwert muss dieser korrigiert werden. Da dies nicht der Fall ist wird der Wert 25 in der nächsten Iteration auf das Element für den OUTPUT geschoben.

Bei der nächsten Iteration, in Abbildung 96 an Stelle 2. zu finden, wurden alle Werte nach rechts verschoben. Nun steht der Wert 987 im Scope, bei der Vergleichsoperation mit dem Median von 14 ist ersichtlich, dass deren Differenz die maximal konfigurierte Grenze überschreitet. Das Ergebnis: Eine Korrektur ist erforderlich. Die nächste Iteration zeigt, dass dieser Wert korrigiert wurde. Der als fehlerhaft identifizierte Wert wurde mit dem berechneten Median von 14 ersetzt.

Anhand der Verzögerung der Elemente, beträgt im Falle des Beispiels neun, ist der Algorithmus in der Lage einen Wert anhand der zukünftigen zu beurteilen. Dies verhindert, dass eine rapide Änderung als Fehler identifiziert wird. Aufgrund des hohen Datenflusses sind Verzögerungen von rund zehn Elementen für den Gebraucher nicht wahrnehmbar.

### REMINDER AN AMIR CODE EINFÜGEN UND DIE BERECHNUNG DES MEDIANS IM CODE DARSTELLEN

#### 8.2.4.4 Datenaufbereitung und Anweisung für Servoaktoren

Jeder der im Projekt verwendeten Servomotoren, der MG995R, hat im Originalzustand einen maximalen Drehwinkel von 180°- entsprechend auf diesen Winkel wurde der Mechanismus, wie im Kapitel (6.2.5 Vierte Hand (Hybrid-Konzept)) näher beschrieben, entworfen. Die Softwareansteuerung ist das letzte fehlende Glied, welches dafür sorgt, dass die Finger sich optimal bewegen.

Grundsätzlich steht keiner der Finger oder der Daumen in der gleichen Faltstellung sobald sich der Motor um beispielsweise 30° dreht. Um ein einheitliches System zu schaffen, muss somit ermittelt bei welchem Drehungswinkel der Finger bereits gestreckt oder in der maximalen Falteposition, sprich vollkommen eingefaltet ist.

Finger	min. Drehwinkel <sub>ausgestreckt</sub>	max. Drehwinkel <sub>max.gefaltet</sub>
Zeigefinger	104°	0°
Mittelfinger	140°	20°
Ringfinger	4°	108°
Kleinerfinger	0°	108°
Daumen	52°	0°

Das Eingangssubsystem misst die Stellung der Flex-Sensoren und übersetzt diese dann auf einen

Bereich von 0 bis 100%, das Ausgangssubsystem empfängt die prozentuale Faltstellung des Fingers.

Die Funktion „HPCA9685.Servo“ erfordert einen anderen Wertebereich als der empfangene von 0 bis 100 und somit müssen diese Werte gemappt werden.

```
int value_out = map(rueckgabe.grad, 0, 100, fingerLimits[rueckgabe.servoIndex][0],  
fingerLimits[rueckgabe.servoIndex][1]);  
HCPCA9685.Servo(rueckgabe.servoIndex.XXXXX);
```

**REMINDER AN AMIR DEN CODE UNBEDINGT NOCH ANPASSEN**

## 8.2.5 Fremdkomponenten

### 8.2.5.1 MCP3X21

### 8.2.5.2 HCPCA9685

### 8.2.5.3 ESP-NOW und Wifi.h

## 8.3 User Interface

### 8.3.1 Grundlegende Voraussetzungen

Grundvoraussetzungen die das User-Interface erfüllen muss sind eine Anzeige der aktuellen Griffkraft in kg und eine Anzeige für den aktuellen Winkel den jeder Servomotor zur Zeit hat. Es ist zwingend notwendig, dass für die Anbindung des GUIs keine technischen Vorauskenntnisse benötigt werden. Der Verbindungsaufbau muss daher ebenfalls fast von selbst erfolgen. Multiplatform Kompatibilität und eine stabile Laufleistung sollten ebenfalls gegeben sein, um dem Endbenutzer keine Technologie aufzuzwingen, die nicht gewünscht ist.

### 8.3.2 Entwicklungsumgebung **Laci**

Entschieden haben wir uns für QT. Dies ist eine C++ Entwicklungsumgebungen, die besonders Plattform unabhängig ist. Es sind keine Änderungen des Quellcodes notwendig, um die Anwendung für MAC, LINUX oder WINDOWS kompatibel zu machen. Dies bezeichnet man auch als source code portability. Ebenfalls ist es relativ einfach möglich QML-Interfaces zu erstellen und diese mit umfangreichen Funktionalitäten auszustatten.

Weitere Eigenschaften von Qt sind:

- ein GUI-Designer
- ein Debugger
- eine 3D-Umgebung um Modelle zu animieren

### 8.3.3 Konzepte und Überlegungen **Laci**

#### 8.3.3.1 Allgemeines Konzept

Zunächst musste überlegt werden, wie die grafische Oberfläche aussehen soll. Diese wird nämlich immer vor allen Funktionalitäten erstellt, um die anschließende Programmierung übersichtlicher und gegliedert durchführen zu können. Das Erstkonzept sieht folgendermaßen aus:

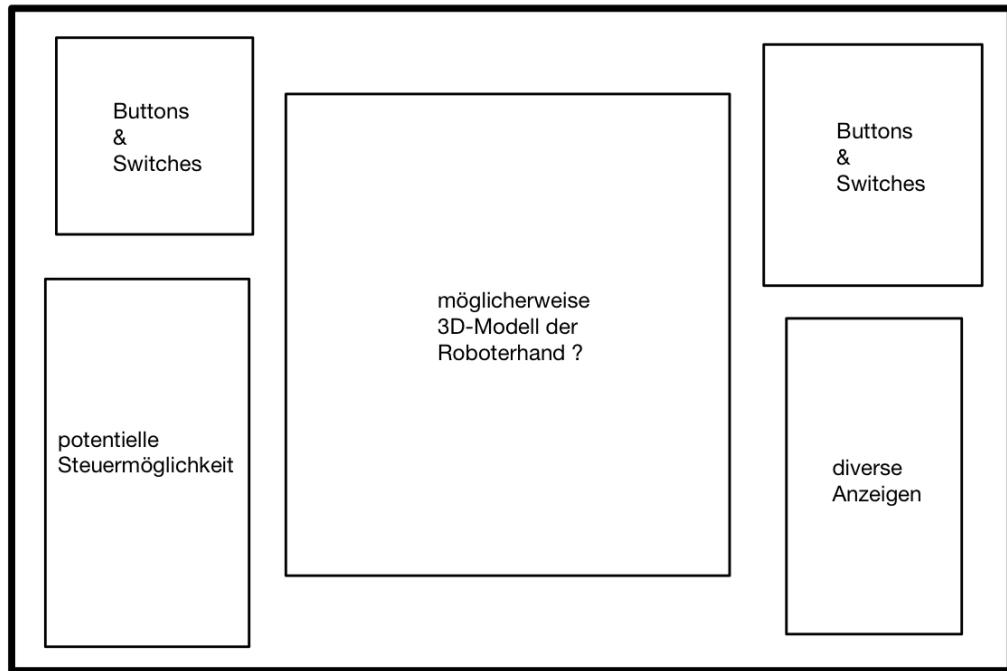


Abbildung 97: User-Interface Konzept

Zu sehen sind die ersten Überlegungen bezüglich des visuellen Layouts. Auf der rechten Seite des Bildschirms sollen die erforderlichen Anzeigen für die Parameter der Griffkraft und des Servodrehwinkels abgebildet werden. Links und rechts oben sollen diverse Knöpfe und Schieberegler zur weiteren Navigation und Kontrolle des UIs platziert werden. Optional kann noch ein 3D-Modell und Regler für die externe Steuerung der Roboterhand hinzugefügt werden. Diese Features sind allerdings nicht gefordert.

### 8.3.4 Realisierung des Frontends **Laci**

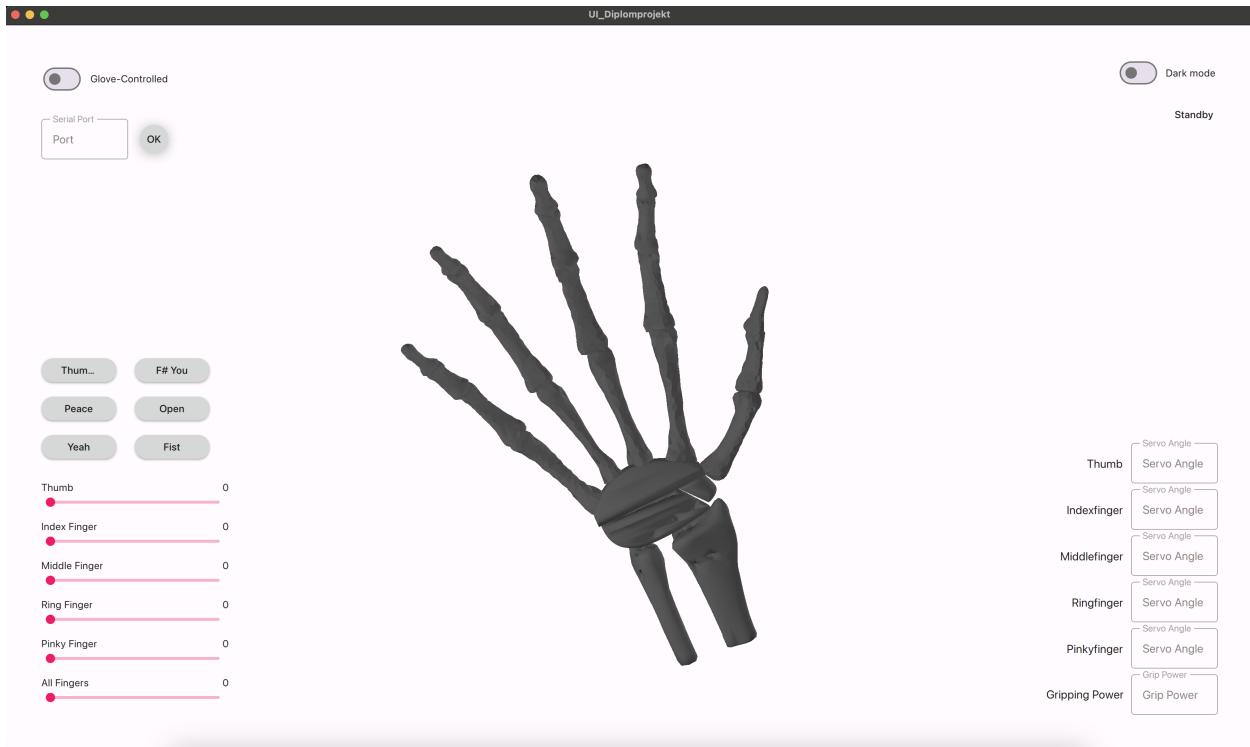


Abbildung 98: UI Benutzeroberfläche

Die Benutzeroberfläche wurde mit der Beschreibungssprache *QML* programmiert. Mit dieser können sehr einfach Tabellen und benutzerdefinierte Anordnungen erstellt werden, mit denen folglich ein visuell ansprechendes GUI kreiert werden kann.

#### 8.3.4.1 Erläuterung der Grafikoberfläche

##### Einstellung des Steuermodus

Links oben in Abbildung 98, sind zwei Elemente zu sehen. Der obere Schalter, ein sogenannter *Toggle-Switch*, ist dafür zuständig zwischen zwei Kontrolmodi für die Roboterhand zu wechseln. In der gezeigten Stellung, kann das Ausgabesubsystem ausschließlich vom Eingabesubsystem, mithilfe der angebrachten Flexsensoren gesteuert werden. Wird dieser Schalter betätigt, so kann die Roboterhand nur mehr mit dem User-Interface bewegt werden.

Als erweiterte Funktion für die Zukunft, kann auf der Platine des Eingabesubsystems ein Schalter angebracht werden, der bei Bestätigung den Steuermodus automatisch umstellt. Hierzu müsste ein kurzer String übergeben werden, der folglich in der QT-Entwicklungsumgebung verarbeitet und erkannt wird.

### Serial-Port Initialisierung

Direkt unter dem Schalter zur Einstellung des Steuermodus, befindet sich das Eingabefeld für den Serial-Port, der vom Benutzer eingegeben werden muss. Durch einen Druck mit der Maus of den *OK – Button*, oder durch ein Drücken der *Enter – Taste* nach dem Eingeben des Portnamen, wird die Serial-Kommunikation mit der Platine des Ausgabesubsystems initialisiert und gestartet.

### Dark-Mode-Switch

In der rechten, oberen Ecke der UI-Oberfläche, befindet sich ein Schalter, mit dem der *Theme* des Interfaces zwischen *Dark* und *Light* gewechselt werden kann. Dies dient ausschließlich der visuellen Darstellung und hat keinen Einfluss auf die Funktionalitäten der Anwendung. QT führt die Farbänderung aller Elemente automatisch durch, wodurch Programmieraufwand gespart wird.

### Standby-Switch

Unterhalb des *Dark – Mode – Switches*, befindet sich ein *Standby – Button*. Wird dieser Knopf gedrückt, so bewegen sich alle Finger der Roboterhand in eine halbgeschlossene Stellung. Der Standby-Modus wird aufgehoben, sobald eine erneute Eingabe erfolgt. Wichtig zu beachten ist allerdings, dass die *Standby – Funktion* im User-Interface nur verwendet werden kann, wenn der Schalter zum Einstellen des Steuermodus auf *UI – Controlled* gestellt ist.

Als zukünftige Verbesserung, könnte, wie auch schon bei der Einstellung des Steuermodus erwähnt, eine Funktion auf der Platine des Eingabesubsystems eingebaut werden, mit der eine *Standby – Condition* ebenfalls bei der Steuerung mit dem Handschuh erzwungen werden kann. Um nicht zu viele Schalter und Drucktaster auf der Platine verbauen zu müssen, könnte diese Eingabe auch sprachlich realisiert werden.

### Vorgefertigte Posen

Links, mittig im User-Interface, befindet sich eine Tabellenstruktur. Diese ist mit sechs *Buttons* bestückt, die, bei Betätigung dieser, eine schon vorgefertigte Pose an das Ausgabesubsystem schicken. Diese Handstellung wird folglich von der Roboterhand eingenommen.

Bei jeder, durch das User-Interface, initiierten Bewegung, wird ein *String* an das Ausgabesubsystem geschickt. Durch dieses, im Vorhinein, festgelegte Protokoll, wird die Servostellung übergeben, wodurch die Finger der Roboterhand in die gewünschte Position bewegt werden. Das Code des Ausgabesubsystems prüft die ankommenden Werte vor der Bewegung der Motoren auf Plausibilität.

### Finger-Slider

Im untere, linken Bereich des Grafikoberfläche, befindet sich ein *Stack – Layout*, das mit sechs *Slidern* gefüllt ist. Diese Schieberegler ermöglichen es dem Benutzer jeden Finger der Roboterhand einzeln zu steuern. Dies funktioniert allerdings wieder nur, wenn der *Steuermodusschalter* auf *UI – Controlled* gestellt ist. Mit dem untersten Slider, können auch alle Finger auf einmal bewegt werden. Steuermöglichkeiten wie diese sind vorgesehen, damit der Benutzer bei einer möglichen Fehlfunktion des Eingabesubsystems nicht handlungsunfähig wird. Mit dem User-Interface kann somit noch weitergearbeitet werden, wenn auch nicht so praktikabel wie per Eingabesubsystem,

nämlich dem Handschuh.

Die Übergabe der Steuerwerte erfolgt mit einem Protokoll ([Verweis auf Amir Protokollbeschreibung](#))

### Kontrollanzeigen

Im unteren, rechten Bereich von Abbildung 98, befinden sich die *Kontrollanzeigen* für die Servostellungen und die Griffkraft der Roboterhand. Mittels dem Protokoll ([Verweis auf Protokollbeschreibung von Amir](#)), werden die Servowinkelwerte jedes einzelnen Servos an das User-Interface gesendet. Diese werden interpretiert und auf Plausibilität geprüft. Folglich werden die vorherigen Werte, die im Anzeigefenster standen, überschrieben und der neue aktualisierte Wert angezeigt.

### 3D-Modell

In der Mitte der Grafikoberfläche, ist die, nicht übersehbare, Darstellung eines echten Handskelets zu sehen. Dieses 3D-Modell wurde in Fusion 360 erstellt. Jedes einzelne Teil wurde anschließend exportiert und im QT-DesignStudio korrekt positioniert. Wichtig hierbei ist es zu beachten, dass jedes Hand -und Fingerglied vor dem Export aus Fusion 360 genau an seinem Ursprung liegen muss, da die Beugung der Finger ansonsten nicht korrekt nachgestellt werden kann. Die Drehachse jedes Gelenks, wäre durch einen inkorrekteten Export verschoben.

#### 8.3.5 Realisierung des Backends **Laci**

Mithilfe von C++, wurde im QT-Designer das Backend programmiert. Dies beinhaltet alle Funktionen die für die Serial-Kommunikation notwendig sind und alle weiteren Funktionalitäten die bei der Benutzerinteraktion gefordert sind.

Das Backend ist in ein *Main-Programm*, ein *Header-File* und ein *Source-File* gegliedert. Im Header-File *functions.h*, werden alle benötigten Funktionen erstellt. Im Source-File *functions.cpp*, werden die Funktionen anschließend ausprogrammiert und miteinander verknüpft. Das Wort *Verknuepfen*, bezieht sich dabei auf das *Signal – Slot – System* von QT, durch das Funktionen anhand von bestimmten Ereignissen aufgerufen werden können. Im Main-Programm *main.cpp*, werden die *Signals* und *Slots* in Abhängigkeit gesetzt und somit die Funktionen des Backends miteinander verknüpft.

Der gesamte Code des User-Interface, ist in Punkt Unterunterabschnitt 16.4.2 zu sehen.

Bei der Programmierung des Backends, wurden grundlegende C++-Kenntnisse aus dem Unterricht angewendet, die in dieser Dokumentation nicht näher erläutert werden.

---

## 9 Tests und Messungen

### 9.1 Widerstandsmessung der Flexsensoren **Fabian**

#### Ziel und Sinn der Messung

Jeder Flexsensor besitzt einen bestimmten Widerstandsbereich ( $25k\Omega$  bis maximal  $125k\Omega$ ). Laut Datenblatt muss allerdings von einer Toleranz von  $+ - 30\%$  ausgegangen werden. Deshalb muss der Widerstandswert jedes einzelnen Flexsensors ausgelesen werden, um festzustellen, wie sehr dieser je Flexsensor variiert.

Als Erstes wurde eine Messung jedes einzelnen Flexsensors durchgeführt. Dabei wurden zuerst Punkte auf einem Blatt Papier aufgezeichnet Abbildung 99. Der Flexsensor, der am Anfang, bevor dieser gebogen wird, nur auf der y-Achse liegt, wird dann immer mehr Richtung der jeweilig gekennzeichneten Punkte gebogen. Der untere Teil des Flexsensors bleibt stets auf den Koordinaten (0|0), damit der Winkel von der y-Achse zur x-Achse gemessen werden kann. Punkt Nummer 1 liegt beispielsweise bei dem Winkel von  $17^\circ$ . Es wurde von zwei Flexsensoren der sich ändernde Widerstandswert mit dieser Methode ausgemessen. Man hat am Ende also sieben Widerstandswerte (Winkel von  $0^\circ, 17^\circ, 27^\circ, 40^\circ, 53^\circ, 65^\circ, 76^\circ, 90^\circ$ ) bei jedem der beiden Flexsensoren gemessen (siehe Tabelle 5).

Man kann klar erkennen, dass die Widerstandswerte von Flexsensor 1 klar von jenen von Flexsensor 2 abweichen, gerade immer höheren Bereich des Biegsgrades. Flexsensor 1 erreicht bei einer Biegung von  $90^\circ$  einen maximalen Widerstandswert von  $110k\Omega$ , bei Flexsensor 2 sind es nur  $45k\Omega$ . Des Weiteren konnte festgestellt werden, dass sich der Wert, aufgrund der Befestigung am Handschuh, welche keine stabile Position garantiert, teils bei jeder getätigten Messung wieder etwas verändert hat.

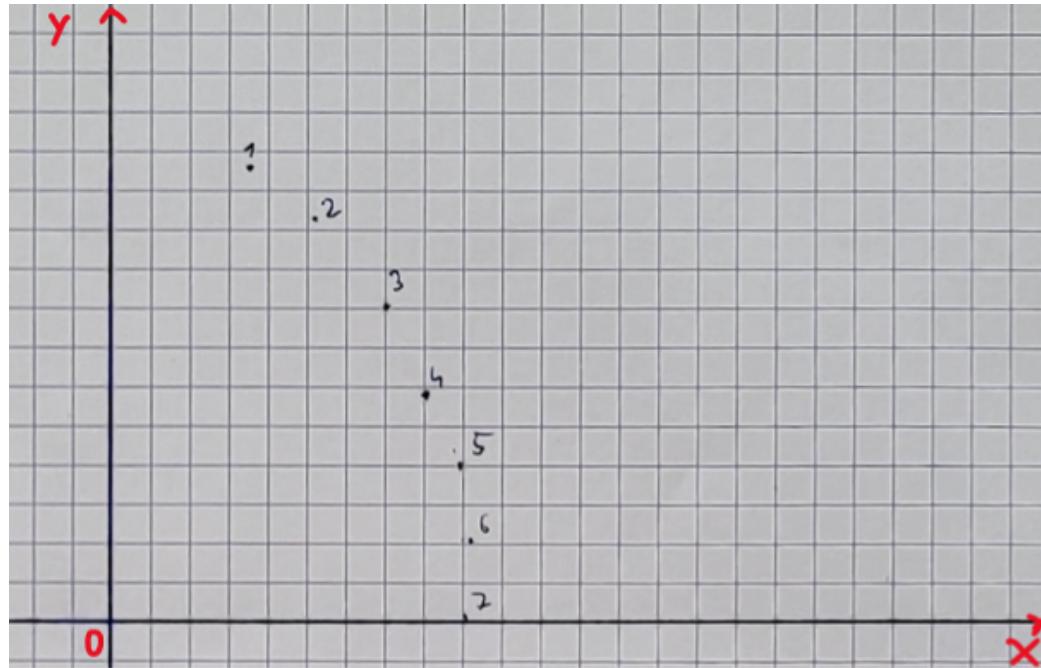


Abbildung 99: Punkte zur Messung (Winkel)

Flexsensor 1		Flexsensor 2	
Winkel ( $^{\circ}$ )	Widerstand ( $k\Omega$ )	Winkel ( $^{\circ}$ )	Widerstand ( $k\Omega$ )
0	44	0	25
17	55	17	28
27	65	27	31
40	75	40	36
53	85	53	39
65	92	65	41
76	102	76	42
90	110	90	45

Tabelle 5: Messergebnisse am Papier

Als Zweites wurden die ausgemessenen Werte von Flexsensor 2 Tabelle 5 übernommen. Zusätzlich wurde nun der Flexsensor 2 auf den Handschuh am Bereich des Zeigefingers der rechten Hand montiert. Die in Tabelle 6 zu sehende Einstellung ( $x/8$ ), gibt den in der linken Tabelle nebenstehenden Winkel jeweils dazu an. Das bedeutet, dass man den Finger ungefähr bei der Einstellung von 2 um  $17^{\circ}$  abgebogen hat. Die nachfolgenden Abbildungen zeigen das unterschiedlich starke Abbiegen des Zeigefingers je Einstellung (1-8).

<b>Flexsensor 2 (Papier)</b>		<b>Flexsensor 2 (Handschuh)</b>	
Winkel ( $^{\circ}$ )	Widerstand ( $k\Omega$ )	Einstellung (x/8)	Widerstand ( $k\Omega$ )
0	25	1	25
17	28	2	27
27	31	3	29
40	36	4	31
53	39	5	33
65	41	6	35
76	42	7	37
90	45	8	39

Tabelle 6: Messergebnisse am Handschuh



Abbildung 100: Einstellung 1

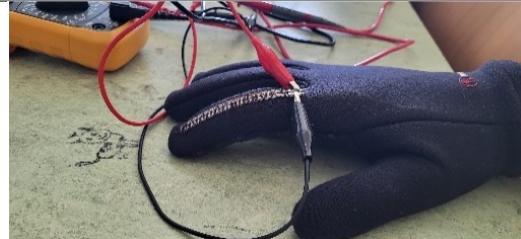


Abbildung 101: Einstellung 2



Abbildung 102: Einstellung 3



Abbildung 103: Einstellung 4



Abbildung 104: Einstellung 5



Abbildung 105: Einstellung 6



Abbildung 106: Einstellung 7



Abbildung 107: Einstellung 8

Tabelle 7: Messpositionen mit dem Handschuh

---

# 10 Ergebnisse und Erkenntnisse

## 10.1 Mechanik **Amir**

### 10.2 Hardware **Laci**

In diesem Punkt werden alle, im Laufe der Projektentwicklung gewonnenen, Erkenntnisse erläutert. Dazu zählen nicht nur die positiven Erfolge, sondern auch die negativen Aspekte und Rückschläge, die wir erfahren haben.

Zunächst wurde ein erstes Konzept entwickelt. In Zuge dessen, wurde hauptsächlich recherchiert und überlegt, wie die einzelnen Ideen zum Bau einer bionischen Hand umgesetzt werden können. Dies betrifft die Messung der Fingerbeugung durch geeignete Sensorik, die Datenübertragung und die Aktorik in Form von Motoren.

Anfangs war der Fortschritt sehr groß. Jede Woche wurden neue Konzepte getestet und durchdacht. Es war sehr schnell klar, dass wir Flexsensoren als Sensorik verwenden wollen und somit konnte der Schaltungsentwurf beginnen. Entscheidungsfreudigkeit am Anfang des Projekts zahlt sich im Laufe der Entwicklung aus, da nicht unnötige Zeit verschwendet wird unnütze und nicht funktionierende Konzepte durchzusetzen. Die Fehler wurde bei uns vermieden und somit hatten wir schon sehr früh Schaltungen und Testplatinen. Durch diese konnten wir Verbesserungen vornehmen und Fehler frühzeitig erkennen.

Frühzeitige Fehler bei der Testschaltung des Eingabesubsystems waren keine Seltenheit. Rund ein -bis zwei Wochen wurden damit verbracht die Platine funktionsfähig zu machen, da durch die Schuliinterne Fertigung einige Fehler beim Fräsen der Leiterbahnen aufgetreten sind. Nicht nur Probleme bei der Fertigung, sondern auch beim Anlöten der Bauteile stoppten den Entwicklungsprozess kurzzeitig. Nachdem mit Jumper-Kabeln alle kaputten elektrischen Leitungen überbrückt worden sind, konnte die Schaltung auf Funktionalität durchgemessen werden. Diese Messungen verliefen, nach der Behebung aller Leitungsdefekte, reibungslos und das Konzept der Messschaltung wurde bestätigt. Dies gab uns die Sicherheit nun die zweite Version der Platine für das Eingabesubsystem zu entwerfen und extern, bei der Firma JLCPCB, fertigen zu lassen.

Durch diese relativ gut verlaufene Phase der Schaltungsentwicklung, hat sich gezeigt, dass es sich auszahlt im Vorhinein zu planen, Schaltungen zu simulieren und zu berechnen und das initiale Konzept mehrmals durchzudenken.

Ein sehr ähnlicher Entwicklungsprozess kam bei der Fertigung der Platine für das Ausgabesubsystem zu tragen. Nach dem Entschluss Servomotoren für die Ansteuerung der Roboterfinger zu verwenden, begann das erneute Überlegen. Diesmal musste allerdings der beste und einfachste Weg für eine Motoransteuerung, bestenfalls mit einer Griffkraftüberwachung, konzeptioniert werden. Die gewählte Schaltung wurde erneut simuliert und auf einem Steckbrett aufgebaut. Danach wurde in der Schule eine Testplatine gefertigt und bestückt. Erneut sind allerdings die gleichen Probleme wie bei der Entwicklung der Schaltung für das Eingabesubsystem aufgetreten. Leiterbahnen haben sich beim Löten der elektronischen Bauteile gelöst, weshalb Jumper-Kabel verwendet werden mussten. Nach der Überbrückung der kaputten Leitungen, wurde die Platine

erneut durchgemessen und ein Betrieb simuliert. Dadurch konnten wir die korrekte Funktion der Schaltung nachweisen und mit dem Design der zweiten Platine für das Ausgabesubsystem beginnen. Diese wurde ebenfalls bei der Firma JLCPCB bestellt.

Erneut war zu erkennen, dass sich die Mühe am Anfang der Entwicklung ausgezahlt hat um mögliche Probleme, die später im Entwicklungsprozess auftreten könnten, auszuschließen.

### 10.3 Software

#### 10.3.1 Eingabesubsystem **Fabian**

Als Erstes wurde ein Konzept entwickelt, was das Eingabesubsystem alles machen soll. Es wurden etliche Überlegungen getroffen, inwiefern welche Funktionen implementiert werden sollen oder nicht, um eine zuverlässige Steuerung des Ausgabesystems zu ermöglichen. Nachdem das Grundkonzept für die Hardware fertiggestellt worden war, konnte mit der detaillierten Planung für das Eingabesubsystem angefangen werden.

Anfangs waren zu viele verschiedene Funktionen, die das Eingabesubsystem unterstützen soll, angedacht gewesen. Es wurde die ersten Wochen mit sehr optimistischen Zügen in die Zukunft geschaut. Es stellte sich jedoch nach nicht allzu langer Zeit heraus, dass es nicht möglich ist, all diese Funktionalitäten zu integrieren. Deshalb wurde ein erneut ein Konzept erstellt, in dem die wichtigsten davon vorhanden waren.

Das Konzept sah wie folgt aus: Es soll über einen ESP32 jeder einzelne Widerstandswert der Flexsensoren beinahe pausenlos ausgelesen werden. Diese Werte sollen dann verarbeitet und, falls die durch den ADC umgewandelten Werte außerhalb des festgelegten Wertebereichs liegen sollten, korrigiert werden.

Mit der Überlegung, welche Datenübertragung gewählt werden soll, wurde ebenfalls Zeit verbracht. Die zu Beginn angestrebte Übertragung per Bluetooth wurde schlussendlich verworfen. Schlussendlich wird diese nun per ESP-NOW Protokoll (ähnlich zu Wifi) durchgeführt. Einige Wochen vergingen, bis klar war, inwiefern die Datenübertragung per ESP-NOW realisiert werden kann. Als es aber dann möglich war, einen stabilen Datenaustausch zwischen Eingabe- und Ausgabesubsystem herzustellen, wurde dieser Schritt, nicht Bluetooth zu verwenden, als der absolut richtige angesehen.

Nachdem das Auslesen der Daten der Flexsensoren vom Eingabesubsystem erfolgreich durchgeführt werden konnte, wurde eine Datenkorrektur angestrebt. Zwischendurch wurde angedacht, keine Korrektur durchzuführen, aber schlussendlich wurde eine Schutzmaßnahme eingeführt, um das Zittern der Finger des Ausgabesystems zu verringern, indem die Werte, die aus dem ADC ausgelesen werden, nur dann gesendet werden, wenn diese um  $\pm 10$  vom zuletzt gesendeten Wert abweichen. Durch diese Maßnahme konnte eine deutliche Verbesserung der natürlich aussehenden Bewegung der Roboterhand vom Ausgabesubsystem sichergestellt werden. Schäden an der Mechanik konnten hiermit vermieden werden.

Die Arduino IDE hat sich durchwegs als passende Entwicklungsumgebung bewährt. Sie ermöglicht es, den Code problemlos zu interpretieren, zu formatieren, zu kompilieren und auf den ESP32 hochzuladen. Allerdings kann es zu einem Problem mit der Arduino IDE 2 kommen, da sie einen bestimmten Codeabschnitt nicht korrekt interpretieren kann. Dies liegt an den Unterschieden in den Bibliotheken im Vergleich zur Arduino IDE 1. Durch Anpassungen am Code kann dieses Problem jedoch gelöst werden, sodass beide Versionen der Arduino IDE genutzt werden können.

Im Verlauf des Entwicklungsprozesses können immer wieder Rückschläge auftreten, unabhängig von deren Ursache. Oft wird die Komplexität bei der Umsetzung bestimmter Funktionen unterschätzt und im Laufe der Zeit muss erkannt werden, dass entweder ein neues Konzept entwickelt oder mehr Zeit in die Realisierung investiert werden muss.

In der Regel erfordert die Umsetzung verschiedener Softwareblöcke mehr Zeit, als ursprünglich angenommen. Der Code funktioniert meist erst nach mehreren Überarbeitungen so, wie es sich von Anfang an gewünscht wurde. Dennoch können zahlreiche neue Bereiche im Softwarebereich erschlossen werden, einschließlich der Ansteuerung und Datenverarbeitung.

Letztendlich erweisen sich die Überlegungen, die bereits am Anfang angestellt wurden, als sinnvoll. Dies führt dazu, dass nicht alle Überlegungen zwischendurch verworfen werden müssen.

### 10.3.2 Ausgabesubsystem **Amir**

## 10.4 Gesamtintegration **Fabian**

In diesem Abschnitt reflektieren wir die Vielfalt an Erkenntnissen, die wir während der Projektentwicklung gesammelt haben. Dabei nehmen wir sowohl die positiven Errungenschaften als auch die Herausforderungen und Rückschläge in den Blick, die wir erlebt haben.

Zunächst möchten wir die Bedeutung eines detaillierten Konzepts hervorheben. Dieses dient als solide Grundlage, auf die wir im weiteren Verlauf zurückgreifen können. Es ermöglicht uns, eine erneute Gesamtüberlegung des Projekts nach einigen Wochen zu vermeiden, obwohl es natürlich unmöglich ist, alle zukünftigen Probleme im Voraus zu antizipieren.

Unser Hauptziel bestand darin, ein umfassendes Konzept für die Integration aller Komponenten zu entwickeln. Im Klartext bedeutet dies, dass Hardware, Software und Mechanik am Ende harmonisch zusammenarbeiten und ein funktionierendes Gesamtsystem bilden sollten. In unserem Konzept haben wir von Anfang an festgelegt, was wir grundsätzlich entwickeln wollen.

Nachdem wir uns darauf geeinigt hatten, einen Roboterarm mittels eines Handschuhs zu steuern, konnten wir uns den Details zuwenden. Es stand schnell fest, dass wir die Position der Fingergelenke mithilfe von Flexsensoren erfassen würden, die auf der Außenseite der Finger angebracht sind.

## 10.4 Gesamtintegration **Fabian**

---

Ebenso war schnell klar, dass wir den Roboterarm mit einem 3D-Drucker herstellen würden. Es gibt schlichtweg keine einfachere Methode, um in kürzester Zeit manuell mit einer Modellierungssoftware Fusion 360 (siehe Unterabschnitt 15.1) erstellte Teile zu produzieren.

Die Entwicklung der Hardware konnte beginnen, sobald wir festgelegt hatten, welche elektronischen Bauteile zum Einsatz kommen sollten. Jede einzelne Verbindung, die wir in der Elektronik-Design-Automatisierungssoftware KiCad (siehe Unterabschnitt 15.3) erstellt haben, wurde sorgfältig durchdacht. Gleichermaßen gilt für die Anordnung der Bauteile, um mögliche Störungen verschiedener Signale zu vermeiden.

Die Softwareentwicklung wurde von Anfang an mithilfe der Entwicklungsumgebung Arduino IDE (siehe Unterabschnitt 15.4) durchgeführt. Der Mikrocontroller ESP32, der eine zentrale Rolle in unserem Projekt spielt, wurde von Anfang an für das Auslesen, Verarbeiten und Senden der Daten ausgewählt.

Nachdem wir unseren ersten 3D-Druck eines Open-Source Projekts der Roboterhand gedruckt hatten, konnten wir uns überlegen, inwiefern wir die Finger bewegen können und wie dies realisiert werden kann. Angelschnüre wurden als Zugmechanismus verwendet, welche durch jeden der einzelnen 3D-gedruckten Fingern verliefen und dann jeweils an einem Servomotor befestigt wurden. Mit der Zeit zeigte sich, dass die Angelschnur allerdings immer mehr an Stabilität verliert und abreißt oder aber auch, dass sie die 3D-gedruckten Teile einschneidet, was wiederum die Mechanik großflächig beschädigt.

Die Platinen waren nicht immer fehlerfrei, durch kreative Ansätze konnte allerdings jedes Mal eine Lösung gefunden werden, wie diese doch verwendet werden können, wie beispielsweise das Setzen von Drahtbrücken, falls Leiterbahnen beschädigt waren.

Die Software war anfangs sehr fehlerbehaftet und hat zu ungewünschten Situationen geführt. Es musste überlegt werden, wie weit sich beispielsweise die Servos drehen dürfen, sodass die Mechanik keinen Schaden davonträgt. Dies war nicht immer leicht, allerdings konnten diese Probleme schlussendlich behoben werden, sodass nun alle Funktionen der Software gezielt an die Hardware und die Mechanik angepasst worden sind.

Schlussendlich konnten wir eine harmonische Interaktion zwischen Hardware, Software und Mechanik erreichen. Die Software liest die Werte der Flexsensoren aus, der Code auf dem ESP32 interpretiert und überprüft diese Daten und leitet sie vom Eingabesystem zum Ausgabesystem weiter. Das Ausgabesystem interpretiert die empfangenen Daten und verarbeitet sie entsprechend. Letztendlich werden die Servomotoren mit diesen Daten angesteuert. Im Code sind Schutzzgrenzen eingebaut, um ein Überdrehen der Servos zu verhindern.

Die Hardware, einschließlich der Messschaltungen, funktioniert einwandfrei und erfüllt alle Anforderungen. In der Endversion gibt es keine Störungen bei den einzelnen Signalen mehr. Die Mechanik wurde mehrfach überarbeitet, insbesondere die Integration der Servos. Durch die Verwendung von Metallhülsen an den erforderlichen Stellen schneiden die Angelschnüre nicht mehr in den 3D-Druck ein.

Insgesamt funktioniert die Steuerung sowohl auf der Seite des Eingabesystems als auch die

Ausführung am Ausgabesystem sehr gut. Es ist uns gelungen, eine sehr natürliche Bewegung der menschlichen Hand nachzuahmen.

Während der gesamten Entwicklungsphase konnten wir zahlreiche Erfahrungen sammeln. Es gab viele Herausforderungen, aber auch daraus resultierende Erfolge waren keine Seltenheit. Oft waren wir etwas zu optimistisch, aber letztendlich konnten wir alle Meilensteine rechtzeitig erreichen. Mit dem Wissen, das wir während der gesamten Entwicklungsphase gesammelt haben, können wir nun effizienter an ähnlichen Projekten arbeiten.

---

## 11 Ausblick

Das Projekt bietet eine gute Basis für zukünftige, aufbauende Erweiterungen. Einige Änderungs- oder Verbesserungsvorschläge des Gesamtdesigns sind in Folge beschrieben.

### **Handgelenksrotation**

Auf der Platine des Eingabesystems könnte eine Steuerung für einen Gyroskopsensor integriert werden. Dieser Sensor, am Handschuh befestigt, wäre in der Lage, die Drehbewegungen des Handgelenks zu erfassen. Die erfassten Daten könnten dann durch eine speziell dafür entwickelte Software auf der Platine ausgelesen und verarbeitet werden. Dies würde es ermöglichen, nicht nur die Finger des Roboterarms zu bewegen, sondern auch eine Rotation des Handgelenks zu realisieren. Ein solches Feature würde den zusätzlichen Vorteil bieten, dass Objekte aus verschiedenen Winkeln gegriffen werden könnten.

### **Taktile Sensoren an den Fingerspitzen**

Taktile Sensoren an den Fingerspitzen der Roboterhand könnten den auf jeden einzelnen Finger ausgeübten Druck messen. Durch eine spezielle Messschaltung könnten die Daten dieser Sensoren ausgelesen und verarbeitet werden. Dies würde es ermöglichen, zu bestimmen, ob ein bestimmter Druckpunkt erreicht ist, bei dem ein gehaltenes Objekt beschädigt oder sogar zerstört werden könnte. Bei einem zu hohen gemessenen Druck (der je nach Objekt variieren und möglicherweise manuell eingestellt werden könnte) könnte eine Sicherheitsfunktion in der Software aktiviert werden. Diese würde die Roboterhand in ihrer aktuellen Position verharren lassen und keine zusätzliche Kraft auf das Objekt ausüben.

### **Akku auf Platine des Eingabesubsystems**

Es wurde bereits ein Akku für die Platine des Eingabesubsystems eingeplant. Dieser ist auch bereits in der neuesten Version der Platine vorhanden und sollte auch funktionieren. Damit wäre auch ein Betrieb dieser ohne ein Netzteil möglich. Das hat den Vorteil, dass man mit dem Handschuh nicht mehr von der Anbindung des Netzteils an die Platine abhängig und kann sich schlussfolgernd leichter durch den Raum bewegen. Die Daten werden genauso wie zuvor übertragen, da alles für beide Betriebsmöglichkeiten (Akku und Netzteil) konzipiert wurde.

### **Mehrere Modi im User-Interface**

Es soll möglich sein, dass mehrere voreingestellte Modi im User-Interface ausgewählt werden können. Über diese soll es möglich gemacht werden, dass man beispielsweise einen bestimmten Gegenstand einer definierten Liste auswählen kann. Darüber soll es dann möglich sein, dass das Ausgabesubsystem ja nach eingestelltem Modus die Gegenstände mit der richtigen Intensität greift, ohne, dass diese beschädigt oder zerstört beziehungsweise aber auch nicht fallengelassen werden. Ein anderer Modus wiederum könnte beispielsweise den Befehl an die Platine senden, dass die Roboterhand in der derzeitigen Position halten soll.

### **Sicherheit beim Senden von Daten**

Es könnte auf eine Verschlüsselung beim Senden von Daten gesetzt werden, damit diese kein Unbefugter abfangen und auswerten kann. In unserem Fall stellt es kein Problem dar, dass die Daten keine Verschlüsselung besitzen, aber in anderen Bereichen, beispielsweise in der Industrie, wird eine zuverlässige und verschlüsselte Übertragung der Daten essenziell sein. Keiner soll

---

dadurch das System manipulieren können.

### **Größere Analyse von Daten**

Durch einen zusätzlichen Datenträger, wie zum Beispiel einer SD-Karte, können Daten dauerhaft gespeichert werden. Dadurch wäre es möglich eine Analyse dieser über einen festgelegten Zeitraum durchzuführen. Es könnte dadurch herausgefunden werden, wie viel Strom bei einem festgelegten Gegenstand beispielsweise bei einer bestimmten Drehung des Servos von der Schaltung verbraucht wird. Wenn dies über einige erneut folgende Messungen beinahe gleich ist, dann kein der arithmetische Mittelwert herausgefunden werden. Dieser kann dann für bestimmte Sicherheitsvorkehrungen verwendet werden. Umso öfter Daten gesammelt und analysiert wurden, umso leichter ist es, festzulegen, wann die Schaltung nicht mehr richtig funktioniert. Außerdem kann über eine ausführliche Analyse verschiedenster gemessener Werte besser analysiert werden, wie oft gewisse Stromspitzen auftreten.

### **Mechanische Komponenten kompakter verbauen**

Das mechanische Gehäuse für den Motorblock kann kompakter modelliert werden. Des Weiteren ist der 3D-gedruckte Unterarm etwas zu groß geraten. In Zukunft müsste hier auf Maße geachtet werden, die dem menschlichen Unterarm entsprechen, um eine realistischere Darstellung zu gewährleisten. In vielen Bereichen des modellierten Roboterarms sind momentan zu viele Lufträume, die bei zukünftigen Modellierungen deutlich reduziert werden können.

### **Professionell gefertigte mechanische Teile**

---

## **12 Anhang**

### **13 Abkürzungsverzeichnis**

**ADC** Analog-Digital-Wandler

**CAD-Software** Computer Aided Design Software

**FDM** Fused Deposition Modelling

**GUI** Graphical User Interface

**GND** Ground

**IC** Integrated Circuit

**I2C** Inter-Integrated Circuit

**IP-Adresse** Internet Protocol Address

**MAC-Adresse** Media Access Control Address

**OPV** Operationsverstärker

**PCB** Printed Circuit Board

**PLA** Polyactid

**SLS** Selective Laser Sintering

**SMD** Surface Mount Device

**THT** Through Hole Technology

**UART** Universal Asynchronous Receiver / Transmitter

**UI** User Interface

**USB** Universal Serial Bus

**WIFI** Wireless Fidelity

**WLAN** Wireless Local Area Network

---

## 14 Projektmanagement

### 14.1 Projektstrukturplan

### 14.2 Milestoneplan

Milestonenummer	Beschreibung	Fälligkeitsdatum
1	Lastenheft fertig	10.10.2023
2	Kostenkalkulationen fertig	24.10.2023
3	Prototypen proof of concept	21.11.2023
4	Hardwaredesign fertig	05.12.2023
5	PCB-Design fertig	09.01.2024
6	Softwareimplementierung für Integrationstest fertig	30.01.2024
7	Integration aller Komponenten	06.02.2024
8	User Interface fertiggestellt	13.02.2024
9	Abnahme durch die Projektbetreuer	12.03.2024

Tabelle 8: Milestoneplan

### 14.3 Gantt-Diagramm

## 14.4 Arbeitstunden

Datum	Arbeitstunden	Uhrzeit	Person	verrichtete Arbeit
07.09.2023	3,5	9:50 - 13:20	Fabian Schweitzer	Datenbanktexte fertigstellen, mechanische Überlegungen für Roboterhand konkretisieren
07.09.2023	3,5	9:50 - 13:20	Amir Al-Maytah	Datenbanktexte fertigstellen, mechanische Überlegungen für Roboterhand konkretisieren
07.09.2023	3,5	9:50 - 13:20	Ladislaus Szabo	Datenbanktexte fertigstellen, mechanische Überlegungen für Roboterhand konkretisieren
12.09.2023	3,5	13:20 - 16:50	Fabian Schweitzer	Kostenkalkulationen begonnen
12.09.2023	3,5	13:20 - 16:50	Amir Al-Maytah	Testaufbau der Roboterhand mit Servomotoren, Netzteil und Potentiometern
12.09.2023	3,5	13:20 - 16:50	Ladislaus Szabo	Platinen fertig bearbeitet und Prof. Fuchsberger zur Inspektion und möglichen Bestellung gegeben
19.09.2023	3,5	13:20 - 16:50	Fabian Schweitzer	Kostenkalkulationen weitergeführt
19.09.2023	3,5	13:20 - 16:50	Amir Al-Maytah	Testaufbau der Roboterhand weitergemacht
19.09.2023	3,5	13:20 - 16:50	Ladislaus Szabo	Hardwaredesign der Messschaltungen für Strom und Spannung mittels ADC
20.09.2023	3,5	12:30 - 16:00	Fabian Schweitzer	Mindmap für Programmierung erstellt, Software Roadmap erstellt
20.09.2023	3,5	12:30 - 16:00	Amir Al-Maytah	Testaufbau Roboterhand, Konzept für die Programmierung überlegt

Datum	Arbeitstunden	Uhrzeit	Person	verrichtete Arbeit
20.09.2023	3,5	12:30 - 16:00	Ladislaus Szabo	Hardwareentwicklung Spannungs - und Strommessung, Kicad Schaltplan bearbeiten
21.09.2023	3,5	9:50 - 13:20	Fabian Schweitzer	Softwareentwicklung, Übertragungsraten der ADCs angeschaut
21.09.2023	3,5	9:50 - 13:20	Amir Al-Maytah	Überarbeitung der Mechanik
21.09.2023	3,5	9:50 - 13:20	Ladislaus Szabo	Hardwareentwicklung, Simulationen
26.09.2023	3,5	13:20 - 16:50	Fabian Schweitzer	Flexsensoren ausmessen
26.09.2023	3,5	13:20 - 16:50	Amir Al-Maytah	Servomotor ausmessen
26.09.2023	3,5	13:20 - 16:50	Ladislaus Szabo	bei beiden Messungen unterstützt
27.09.2023	3,5	12:30 - 16:00	Fabian Schweitzer	Ausmessung der Flexsensoren und Dokumentieren der Ergebnisse
27.09.2023	3,5	12:30 - 16:00	Amir Al-Maytah	Kalibrierung der Roboterhand
27.09.2023	3,5	12:30 - 16:00	Ladislaus Szabo	Versuchsaufbau Spannungsmessung mit INA129 und Hardwareentwicklung
28.09.2023	3,5	9:50 - 13:20	Fabian Schweitzer	Dokumentation und Testen der Flexsensoren
28.09.2023	3,5	9:50 - 13:20	Amir Al-Maytah	Verbesserung der Roboterhand
28.09.2023	3,5	9:50 - 13:20	Ladislaus Szabo	Hardwareentwicklung und PCB-Design, Bauteilliste aktualisieren
03.10.2023	3,5	13:20 - 16:50	Fabian Schweitzer	Software für das Auslesen der Flexsensoren geplant und Strommessung mitaufgebaut
03.10.2023	1,5	12:30 - 14:00	Amir Al-Maytah	Strommessschaltung (Servoansteuerung)

## 14.4 Arbeitstunden

---

Datum	Arbeitstunden	Uhrzeit	Person	verrichtete Arbeit
03.10.2023	3,5	13:20 - 16:50	Ladislaus Szabo	Auswertung der Strommessung am Steckbrett
04.10.2023	3,5	12:30 - 16:00	Fabian Schweitzer	Servomessung und Versuchsaufbau
04.10.2023	3,5	12:30 - 16:00	Amir Al-Maytah	Servomessung und Versuchsaufbau
04.10.2023	3,5	12:30 - 16:00	Ladislaus Szabo	Richtlinien heraussuchen und Pflichtenheft verfasst
05.10.2023	3,5	9:50 - 13:20	Fabian Schweitzer	Dokumentation der Richtlinien und das Projekthandbuch weitergemacht
05.10.2023	3,5	9:50 - 13:20	Amir Al-Maytah	Programmierung des User-Interface
05.10.2023	3,5	9:50 - 13:20	Ladislaus Szabo	Platinen fräsen und in der mechanischen Werkstatt wegen dem Fräsen von Teilen fragen
10.10.2023	3,5	13:20 - 16:50	Fabian Schweitzer	Platinen gelötet
10.10.2023	3,5	13:20 - 16:50	Amir Al-Maytah	Roboterhand zusammengebaut
10.10.2023	1,0	13:20 - 14:20	Ladislaus Szabo	Dokumentation und Hilfe beim Löten
11.10.2023	3,5	12:30 - 16:00	Fabian Schweitzer	Platinen gelötet
11.10.2023	3,5	12:30 - 16:00	Amir Al-Maytah	Roboterhand zusammengebaut
11.10.2023	3,5	12:30 - 16:00	Ladislaus Szabo	Dokumentation und Messprotokoll erstellt
12.10.2023	3,5	9:50 - 13:20	Fabian Schweitzer	Am Programm zum Auslesen der Flexsensoren weitergearbeitet
12.10.2023	3,5	9:50 - 13:20	Amir Al-Maytah	Roboterhad kalibriert
12.10.2023	3,5	9:50 - 13:20	Ladislaus Szabo	Dokumentation und Messprotokolle erstellt

Datum	Arbeitstunden	Uhrzeit	Person	verrichtete Arbeit
17.10.2023	3,5	13:20 - 16:50	Fabian Schweitzer	Bestücken der Platinen
17.10.2023	3,5	13:20 - 16:50	Amir Al-Maytah	Mechanik und erste Überlegungen für die Programmierung
17.10.2023	3,5	13:20 - 16:50	Ladislaus Szabo	Bestückung der Platinen
18.10.2023	3,5	12:30 - 16:00	Fabian Schweitzer	Flussdiagramm für die Softwareentwicklung erstellt
18.10.2023	3,5	12:30 - 16:00	Amir Al-Maytah	Programmierung der Handschuhplatine
18.10.2023	3,5	12:30 - 16:00	Ladislaus Szabo	Löten der Platinen
19.10.2023	3,5	9:50 - 13:20	Fabian Schweitzer	Programmierung der ESPs
19.10.2023	3,5	9:50 - 13:20	Amir Al-Maytah	Programmierung der ESPs
19.10.2023	3,5	9:50 - 13:20	Ladislaus Szabo	Löten der Platinen
24.10.2023	3,5	13:20 - 16:50	Fabian Schweitzer	Programmierung der ESPs und Besprechung des generellen Aufbaus der Programme mit Prof. Diemberger
24.10.2023	0,0	/	Amir Al-Maytah	/
24.10.2023	3,5	13:20 - 16:50	Ladislaus Szabo	Löten der Platinen und ausbessern von diversen Fehlern und Kurzschlüssen
18.10.2023	3,5	12:30 - 16:00	Fabian Schweitzer	Programmierung der Übertragung zwischen zwei ESPs
18.10.2023	3,5	12:30 - 16:00	Amir Al-Maytah	Programmierung der Übertragung zwischen zwei ESPs
18.10.2023	3,5	12:30 - 16:00	Ladislaus Szabo	Löten der Platinen und ausbessern von Fehlern
07.11.2023	3,5	13:20 - 16:50	Fabian Schweitzer	ESP32 Programmierung
07.11.2023	3,5	13:20 - 16:50	Amir Al-Maytah	ESP32 Programmierung
07.11.2023	3,5	13:20 - 16:50	Ladislaus Szabo	Letzter Check der Platinen und diese bei JLCPCB bestellt

## 14.4 Arbeitstunden

---

Datum	Arbeitstunden	Uhrzeit	Person	verrichtete Arbeit
09.11.2023	3,5	9:50 - 13:20	Fabian Schweitzer	ESP32 Programmierung
09.11.2023	3,5	9:50 - 13:20	Amir Al-Maytah	ESP32 Programmierung
09.11.2023	3,5	9:50 - 13:20	Ladislaus Szabo	Dokument "Hardwarebeschreibung" verfasst
21.11.2023	3,5	13:20 - 16:50	Fabian Schweitzer	Programmierung der drahtlosen Schnittstelle zwischen beiden ESPs
21.11.2023	3,5	13:20 - 16:50	Amir Al-Maytah	Programmierung der drahtlosen Schnittstelle zwischen beiden ESPs
21.11.2023	3,5	13:20 - 16:50	Ladislaus Szabo	Testen der neuen Platinen
22.11.2023	0,0	11:30 - 16:00	Fabian Schweitzer	Programmierung der Übertragung zwischen zwei ESPs
22.11.2023	4,5	11:30 - 16:00	Amir Al-Maytah	Programmierung der Übertragung zwischen zwei ESPs
22.11.2023	4,5	11:30 - 16:00	Ladislaus Szabo	Löten der Platinen und ausbessern von Fehlern
23.11.2023	3,5	9:50 - 13:20	Fabian Schweitzer	ESP32 Programmierung und Platinentesting
23.11.2023	3,5	9:50 - 13:20	Amir Al-Maytah	ESP32 Programmierung und Platinentesting
23.11.2023	3,5	9:50 - 13:20	Ladislaus Szabo	Testen und Durchmesen der Platinen gemeinsam mit Softwareüberprüfung
28.11.2023	3,5	13:20 - 16:50	Fabian Schweitzer	erste Servoansteuerung mit der neuen Platine
28.11.2023	3,5	13:20 - 16:50	Amir Al-Maytah	Weiterentwicklung der Roboterhand und erneutes Testen der Handschuhplatine nach beheben eines Lötfehlers
28.11.2023	3,5	13:20 - 16:50	Ladislaus Szabo	Fehlersuche an den Platinen und Messtabellen für die Messschaltung der Flexsensoren erstellt

Datum	Arbeitstunden	Uhrzeit	Person	verrichtete Arbeit
29.11.2023	3,5	12:30 - 16:00	Fabian Schweitzer	Programmierung des ESP Senderprogramms
29.11.2023	3,5	12:30 - 16:00	Amir Al-Maytah	Programmierung des ESP Empfängerprogramms
29.11.2023	3,5	12:30 - 16:00	Ladislaus Szabo	Unterstützung bei der Programmierung
30.11.2023	3,5	9:50 - 13:20	Fabian Schweitzer	Überlegungen für Algorithmus zur Korrektur der Flexsensortoleranz
30.11.2023	3,5	9:50 - 13:20	Amir Al-Maytah	Entwicklung des Algorithmus für Fehlerwertkorrektur auf der Roboterhandplatine
30.11.2023	3,5	9:50 - 13:20	Ladislaus Szabo	ESPnow Interface testing und Datenübertragung
05.12.2023	1,5	12:30 - 14:00	Fabian Schweitzer	Testen der Strommessung der Roboterhandplatine
05.12.2023	0,0	/	Amir Al-Maytah	/
05.12.2023	1,5	12:30 - 14:00	Ladislaus Szabo	Testen der Strommessung der Roboterhandplatine
06.12.2023	3,5	12:30 - 16:00	Fabian Schweitzer	ESP Programmierung
06.12.2023	3,5	12:30 - 16:00	Amir Al-Maytah	ESP Programmierung
06.12.2023	0,0	/	Ladislaus Szabo	/
07.12.2023	3,5	9:50 - 13:20	Fabian Schweitzer	ESP Programmierung und erste Inbetriebnahme aller Komponenten gemeinsam
07.12.2023	3,5	9:50 - 13:20	Amir Al-Maytah	ESP Programmierung und erste Inbetriebnahme aller Komponenten gemeinsam
07.12.2023	3,5	9:50 - 13:20	Ladislaus Szabo	Überprüfung aller Schaltungsteile bei der ersten Komplettinbetriebnahme
12.12.2023	3,5	13:20 - 16:50	Fabian Schweitzer	ESP Programmierung
12.12.2023	3,5	13:20 - 16:50	Amir Al-Maytah	Roboterhand 3D-Design

## 14.4 Arbeitstunden

---

Datum	Arbeitstunden	Uhrzeit	Person	verrichtete Arbeit
12.12.2023	3,5	13:20 - 16:50	Ladislaus Szabo	Platinendesign letzte Iteration
13.12.2023	3,5	12:30 - 16:00	Fabian Schweitzer	ESP Programmierung
13.12.2023	3,5	12:30 - 16:00	Amir Al-Maytah	Roboterhand 3D-Design
13.12.2023	3,5	12:30 - 16:00	Ladislaus Szabo	Platinendesign letzte Iteration
14.12.2023	3,5	9:50 - 13:20	Fabian Schweitzer	ESP Programmierung
14.12.2023	3,5	9:50 - 13:20	Amir Al-Maytah	Roboterhand 3D-Design
14.12.2023	3,5	9:50 - 13:20	Ladislaus Szabo	Platinendesign letzte Iteration
19.12.2023	3,5	13:20 - 16:50	Fabian Schweitzer	ESP32 Programmierung
19.12.2023	3,5	13:20 - 16:50	Amir Al-Maytah	Design der Roboterhand
19.12.2023	3,5	13:20 - 16:50	Ladislaus Szabo	UI Programmierung
20.12.2023	3,5	12:30 - 16:00	Fabian Schweitzer	ESP32 Programmierung
20.12.2023	3,5	12:30 - 16:00	Amir Al-Maytah	Design der Roboterhand
20.12.2023	3,5	12:30 - 16:00	Ladislaus Szabo	UI Programmierung
21.12.2023	3,5	9:50 - 13:20	Fabian Schweitzer	ESP32 Programmierung
21.12.2023	3,5	9:50 - 13:20	Amir Al-Maytah	Design der Roboterhand
21.12.2023	0.0	/	Ladislaus Szabo	/
09.01.2024	3,5	13:20 - 16:50	Fabian Schweitzer	ESP Programmierung
09.01.2024	3,5	13:20 - 16:50	Amir Al-Maytah	Modellierung der Roboterhand
09.01.2024	3,5	13:20 - 16:50	Ladislaus Szabo	Handschuh mit Sensoren bestückt und verdrahtet
10.01.2024	3,5	12:30 - 16:00	Fabian Schweitzer	Handschuhplatine programmiert
10.01.2024	3,5	12:30 - 16:00	Amir Al-Maytah	C++ Dateien für Roboterhand zusammengeführt
10.01.2024	3,5	12:30 - 16:00	Ladislaus Szabo	UI Programmierung
11.01.2024	3,5	9:50 - 13:20	Fabian Schweitzer	ESP Programmierung
11.01.2024	3,5	9:50 - 13:20	Amir Al-Maytah	Roboterhand - 3D Druck
11.01.2024	3,5	9:50 - 13:20	Ladislaus Szabo	UI Programmierung

Datum	Arbeitstunden	Uhrzeit	Person	verrichtete Arbeit
16.01.2024	3,5	13:20 - 16:50	Fabian Schweitzer	ESP Programmierung
16.01.2024	3,5	13:20 - 16:50	Amir Al-Maytah	ESP Programmierung
16.01.2024	3,5	/	Ladislaus Szabo	/
17.01.2024	3,5	12:30 - 16:00	Fabian Schweitzer	ESP Programmierung
17.01.2024	3,5	12:30 - 16:00	Amir Al-Maytah	Mechanik der Roboterhand verbessert
17.01.2024	3,5	/	Ladislaus Szabo	/
18.01.2024	3,5	9:50 - 13:20	Fabian Schweitzer	ESP Programmierung
18.01.2024	3,5	9:50 - 13:20	Amir Al-Maytah	ESP Programmierung
18.01.2024	3,5	9:50 - 13:20	Ladislaus Szabo	finalen Schaltlan der Handschuhplatine erstellt
23.01.2024	3,5	13:20 - 16:50	Fabian Schweitzer	ESP Programmierung
23.01.2024	3,5	13:20 - 16:50	Amir Al-Maytah	ESP Programmierung
23.01.2024	3,5	13:20 - 16:50	Ladislaus Szabo	ESP Programmierung
24.01.2024	3,5	12:30 - 16:00	Fabian Schweitzer	ESP Programmierung
24.01.2024	3,5	12:30 - 16:00	Amir Al-Maytah	Mechanik der Roboterhand zum 3D drucken fertig gemacht
24.01.2024	3,5	12:30 - 16:00	Ladislaus Szabo	Dokumentation fortgesetzt
25.01.2024	3,5	9:50 - 13:20	Fabian Schweitzer	Tag der offenen Tür Projektvorstellung
25.01.2024	3,5	9:50 - 13:20	Amir Al-Maytah	Tag der offenen Tür Projektvorstellung
25.01.2024	3,5	9:50 - 13:20	Ladislaus Szabo	Tag der offenen Tür Projektvorstellung
30.01.2024	0,0	/	Fabian Schweitzer	/
30.01.2024	0,0	/	Amir Al-Maytah	/
30.01.2024	0,0	/	Ladislaus Szabo	/
31.01.2024	3,5	12:30 - 16:00	Fabian Schweitzer	Strommessung der Roboterhandplatine in Betrieb genommen
31.01.2024	3,5	12:30 - 16:00	Amir Al-Maytah	Zusammenbau der gedruckten Teile der Mechanik

#### 14.4 Arbeitstunden

---

Datum	Arbeitstunden	Uhrzeit	Person	verrichtete Arbeit
31.01.2024	3,5	12:30 - 16:00	Ladislaus Szabo	Dokumentation fortgesetzt und beim Zusammenbau der Mechanik geholfen
01.02.2024	3,5	9:50 - 13:20	Fabian Schweitzer	Finetuning der Sensorik am Handschuh
01.02.2024	3,5	9:50 - 13:20	Amir Al-Maytah	Feinsjustierung der Mechanik
01.02.2024	3,5	9:50 - 13:20	Ladislaus Szabo	Dokumentation fortgeführt und Verbesserungen für die Mechanik überlegt
13.02.2024	0,0	/	Fabian Schweitzer	/
13.02.2024	3,5	13:20 - 16:50	Amir Al-Maytah	Feinsjustierung der Mechanik
13.02.2024	3,5	13:20 - 16:50	Ladislaus Szabo	Dokumentation fortgeführt
14.02.2024	3,5	12:30 - 16:00	Fabian Schweitzer	an der Dokumentation weitergearbeitet
14.02.2024	3,5	12:30 - 16:00	Amir Al-Maytah	an der Dokumentation weitergearbeitet
14.02.2024	3,5	12:30 - 16:00	Ladislaus Szabo	an der Dokumentation weitergearbeitet
15.02.2024	3,5	9:50 - 13:20	Fabian Schweitzer	an der Dokumentation weitergearbeitet
15.02.2024	3,5	9:50 - 13:20	Amir Al-Maytah	an der Dokumentation weitergearbeitet
15.02.2024	3,5	9:50 - 13:20	Ladislaus Szabo	an der Dokumentation weitergearbeitet
20.02.2024	3,5	13:20 - 16:50	Fabian Schweitzer	an der Dokumentation weitergearbeitet
20.02.2024	3,5	13:20 - 16:50	Amir Al-Maytah	Feinjustierung der Mechanik und weiterarbeiten an der Dokumentation
20.02.2024	3,5	13:20 - 16:50	Ladislaus Szabo	Feinjustierung der Mechanik und weiterarbeiten an der Dokumentation

Datum	Arbeitstunden	Uhrzeit	Person	verrichtete Arbeit
21.02.2024	3,5	12:30 - 16:00	Fabian Schweitzer	an der Dokumentation weitergearbeitet
21.02.2024	3,5	12:30 - 16:00	Amir Al-Maytah	an der Dokumentation weitergearbeitet
21.02.2024	3,5	12:30 - 16:00	Ladislaus Szabo	an der Dokumentation weitergearbeitet
22.02.2024	3,5	9:50 - 13:20	Fabian Schweitzer	an der Dokumentation weitergearbeitet
22.02.2024	3,5	9:50 - 13:20	Amir Al-Maytah	an der Dokumentation weitergearbeitet
22.02.2024	3,5	9:50 - 13:20	Ladislaus Szabo	an der Dokumentation weitergearbeitet
27.02.2024	3,5	13:20 - 16:50	Fabian Schweitzer	an der Dokumentation weitergearbeitet
27.02.2024	3,5	13:20 - 16:50	Amir Al-Maytah	an der Dokumentation weitergearbeitet
27.02.2024	3,5	13:20 - 16:50	Ladislaus Szabo	an der Dokumentation weitergearbeitet
28.02.2024	3,5	12:30 - 16:00	Fabian Schweitzer	an der Dokumentation weitergearbeitet
28.02.2024	3,5	12:30 - 16:00	Amir Al-Maytah	an der Dokumentation weitergearbeitet
28.02.2024	3,5	12:30 - 16:00	Ladislaus Szabo	an der Dokumentation weitergearbeitet
29.02.2024	3,5	9:50 - 13:20	Fabian Schweitzer	an der Dokumentation weitergearbeitet
29.02.2024	3,5	9:50 - 13:20	Amir Al-Maytah	an der Dokumentation weitergearbeitet
29.02.2024	3,5	9:50 - 13:20	Ladislaus Szabo	an der Dokumentation weitergearbeitet
05.03.2024	3,5	13:20 - 16:50	Fabian Schweitzer	an der Dokumentation weitergearbeitet
05.03.2024	3,5	13:20 - 16:50	Amir Al-Maytah	an der Dokumentation weitergearbeitet
05.03.2024	3,5	13:20 - 16:50	Ladislaus Szabo	an der Dokumentation weitergearbeitet

#### 14.4 Arbeitstunden

---

Datum	Arbeitstunden	Uhrzeit	Person	verrichtete Arbeit
06.03.2024	3,5	12:30 - 16:00	Fabian Schweitzer	an der Dokumentation weitergearbeitet
06.03.2024	3,5	12:30 - 16:00	Amir Al-Maytah	an der Dokumentation weitergearbeitet
06.03.2024	3,5	12:30 - 16:00	Ladislaus Szabo	an der Dokumentation weitergearbeitet

---

# 15 Programme, Installationen und Plugins

## 15.1 Fusion 360

Fusion 360 ist eine CAD-Software, die es ermöglicht 3D-Modelle zu erstellen, diese zu bearbeiten und anschließend zu exportieren.

## 15.2 UltiMaker Cura

Ultimaker Cura ist eine Software, die es ermöglicht 3D-Modelle für den 3D-Druck vorzubereiten und Problemstellen auszubessern. Die Software bietet einige Einstellmöglichkeiten, wie beispielsweise für das Slicen von Objekten.

## 15.3 KiCad

KiCad ist ein Tool zum Erstellen von Schaltplänen und Platinenlayouts. Es können Footprints zugewiesen, Schaltungsfunktionen überprüft und anschließend Bauteile auf einem Platinenmodell angeordnet werden.

## 15.4 Arduino IDE

Die Arduino IDE ist eine Entwicklungsumgebung zur Programmierung von Arduino-Mikrokontrollern. Durch diverse Plugins können auch andere Mikrokontakte programmiert werden, wie in unserem Fall der ESP23.

## 15.5 QT Framework

QT ist ein Softwareentwicklungstool, das eine plattformunabhängige Programmierung von Anwendungen ermöglicht. Apps können ohne Änderungen des Quellcodes für Linux, Windows und Mac kompiliert werden. Es können ebenfalls Mobile-Anwendungen erstellt werden.

---

## **16 Fertigungsunterlagen**

### **16.1 Mechanik**

#### **16.1.1 Skizzen und Konzepte**

#### **16.1.2 CAD-Zeichnungen**

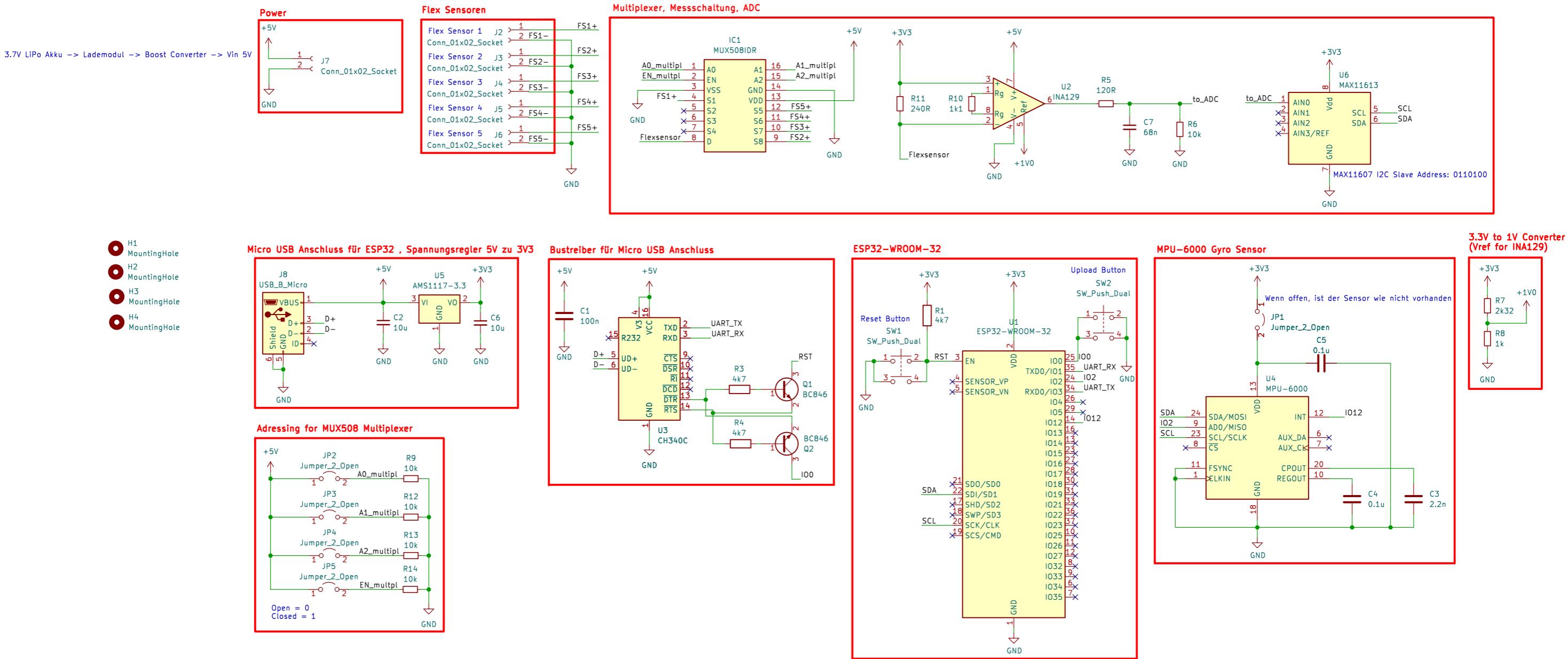
#### **16.1.3 3D-Modelle**

## 16.2 Hardware

### 16.2.1 Skizzen und Konzepte

### 16.2.2 Stromlaufpläne

# Handschoen



erste Version

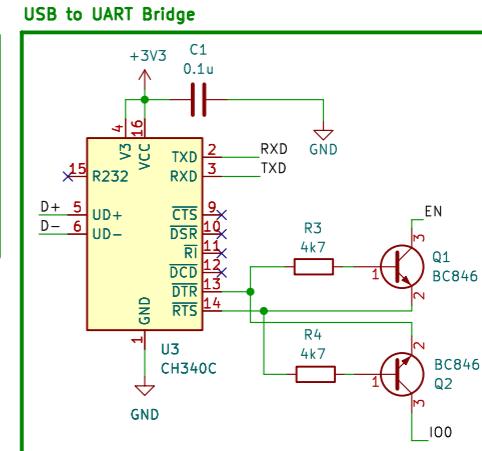
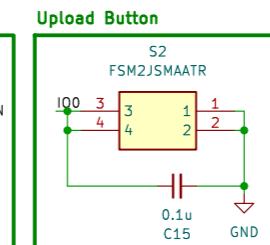
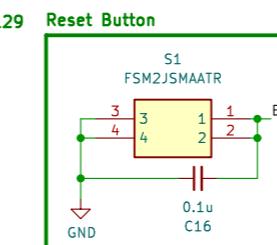
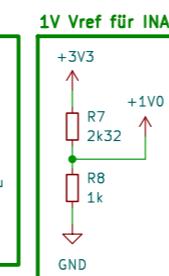
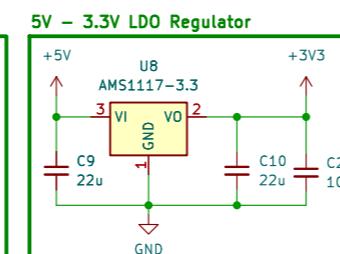
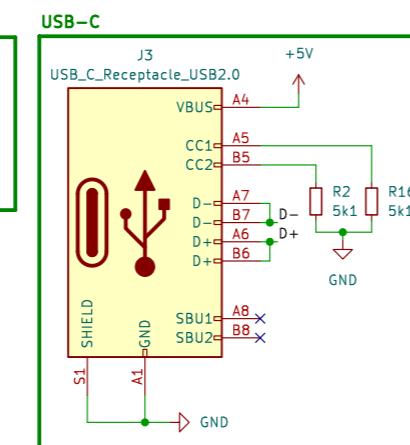
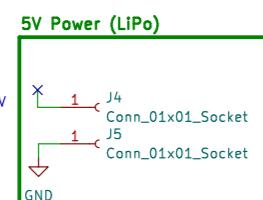
Sheet: /  
File: Handschuh.kicad\_sch

**Title: Schaltplan\_Handschuh**  
Size: A3 Date: 2023-06-27  
KiCad E.D.A. kicad 7.0.7

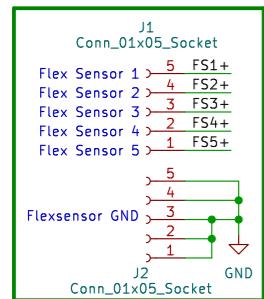
Rev: v1  
Id: 1/1

# Handschoen

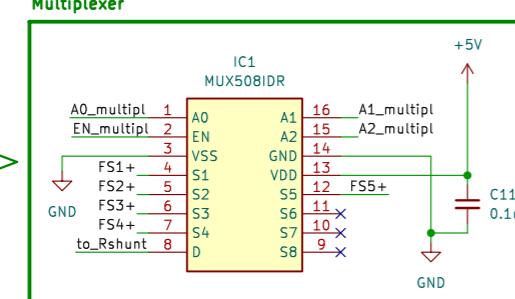
3.7V LiPo Akku -> Lademodul -> Boost Converter -> Vin 5V



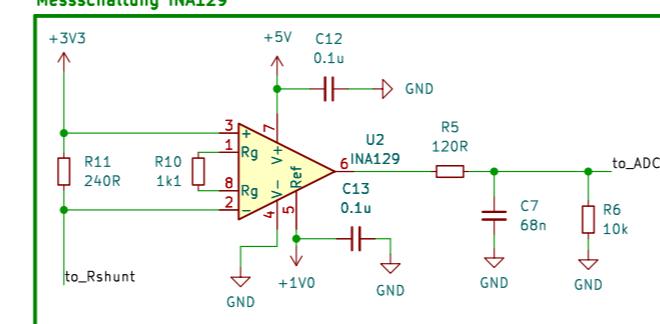
Anschlüsse Flexsensoren



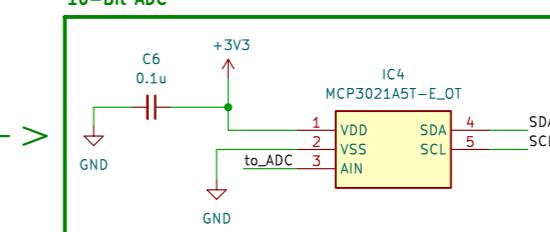
Multiplexer



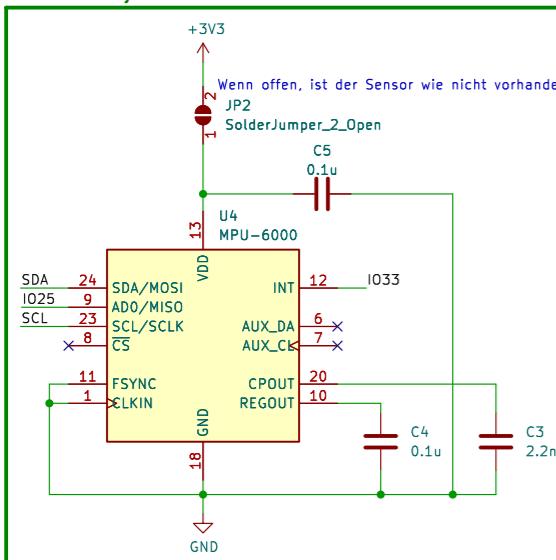
Messschaltung INA129



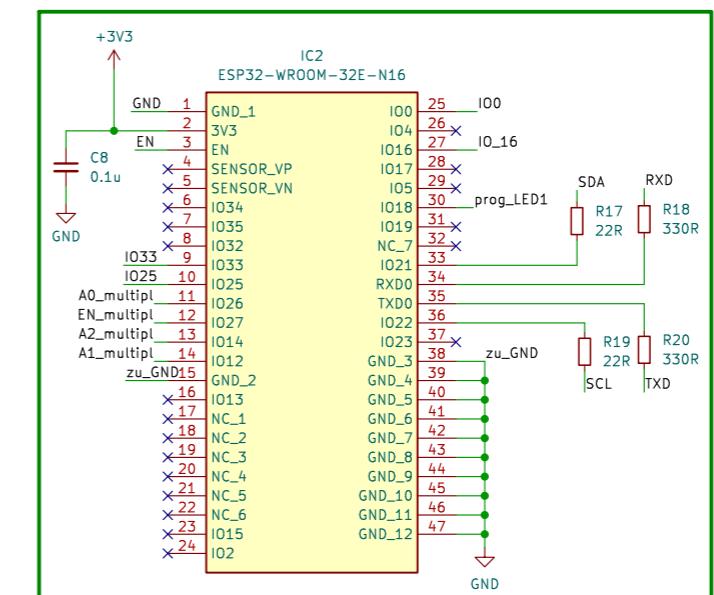
10-Bit ADC



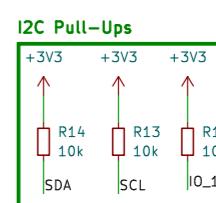
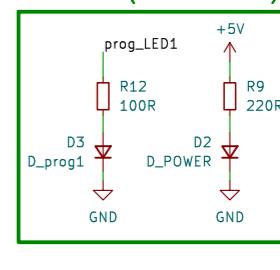
MPU-6000 Gyro Sensor



ESP32-WROOM-32E-N16



Power LEDs (Transmitter LED)



zweite Version

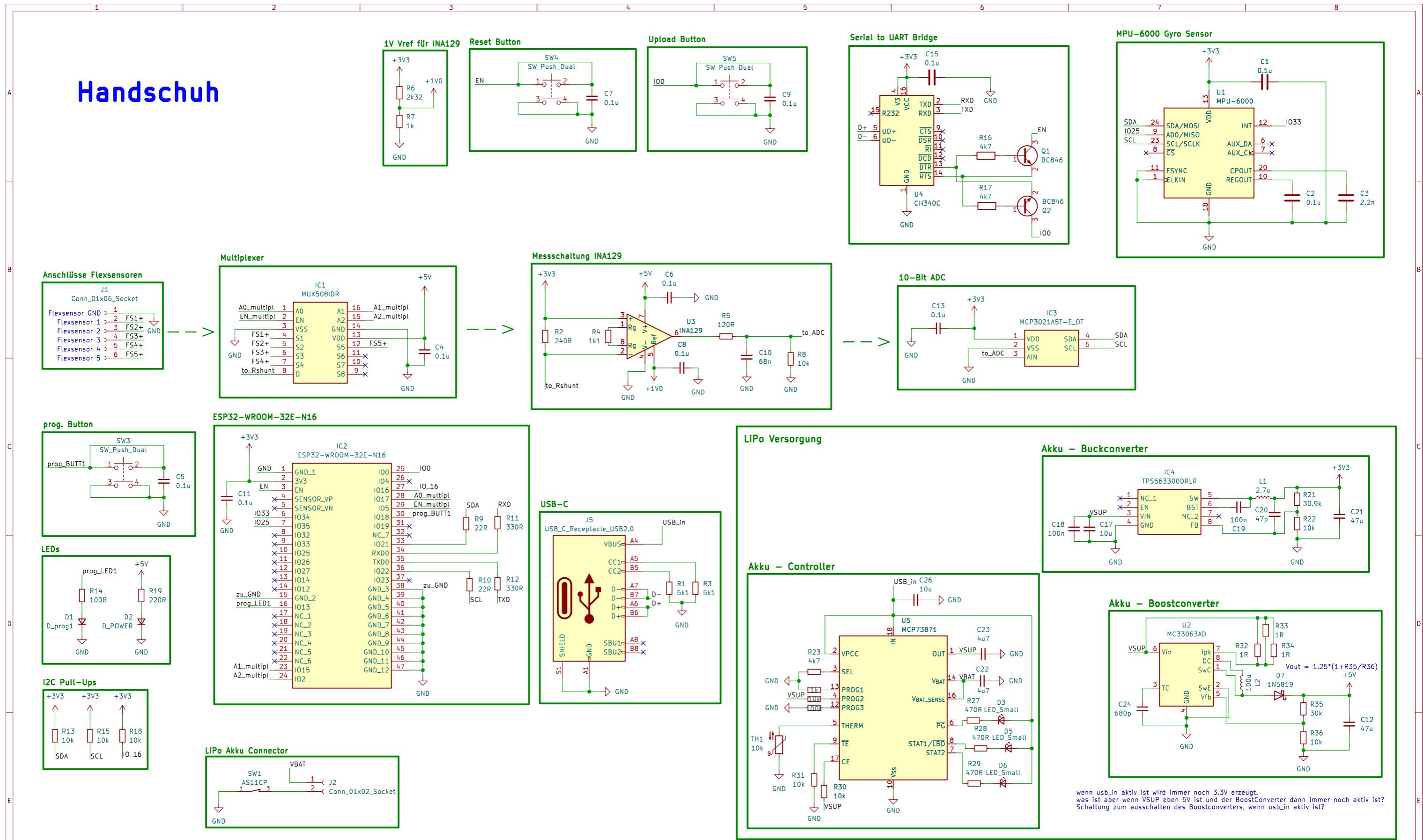
Sheet: / File: Handschuh.kicad\_sch

Title: Schaltplan\_Handschuh

Size: A3 Date: 2023-10-18  
KiCad E.D.A. kicad 7.0.7

Rev: v2 Id: 1/1

# Handschuh



wenn `usb_in` aktiv ist wird immer noch 3.3V erzeugt.  
was ist aber wenn `VSUP` eben 5V ist und der BoostConverter dann immer noch aktiv ist?  
Schaltung zum ausschalten des Boostconverters, wenn `usb_in` aktiv ist?

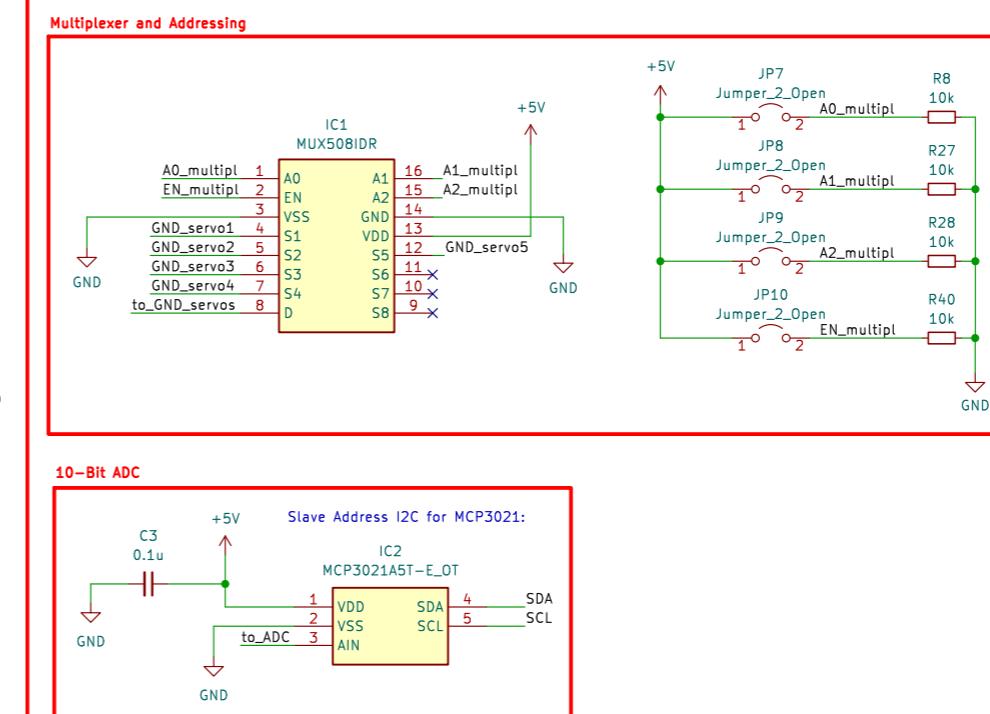
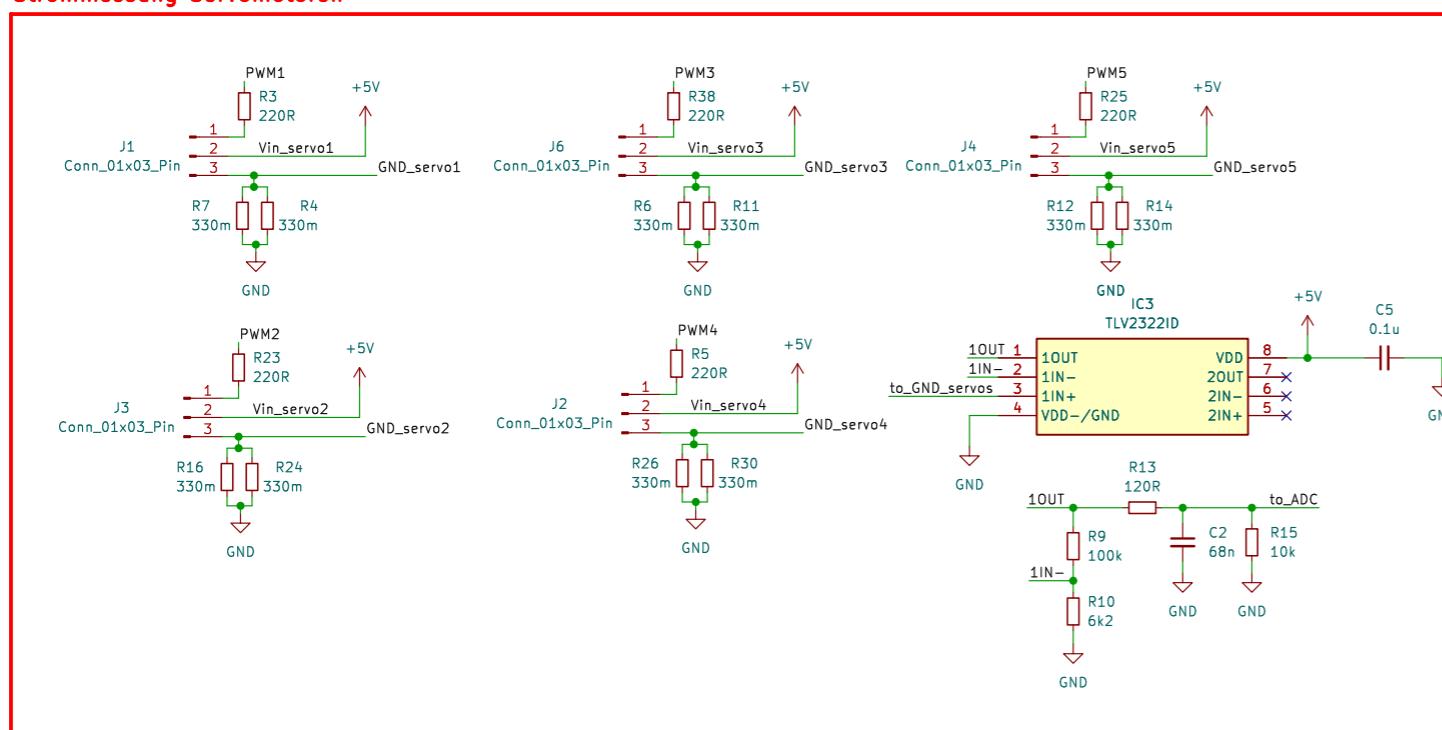
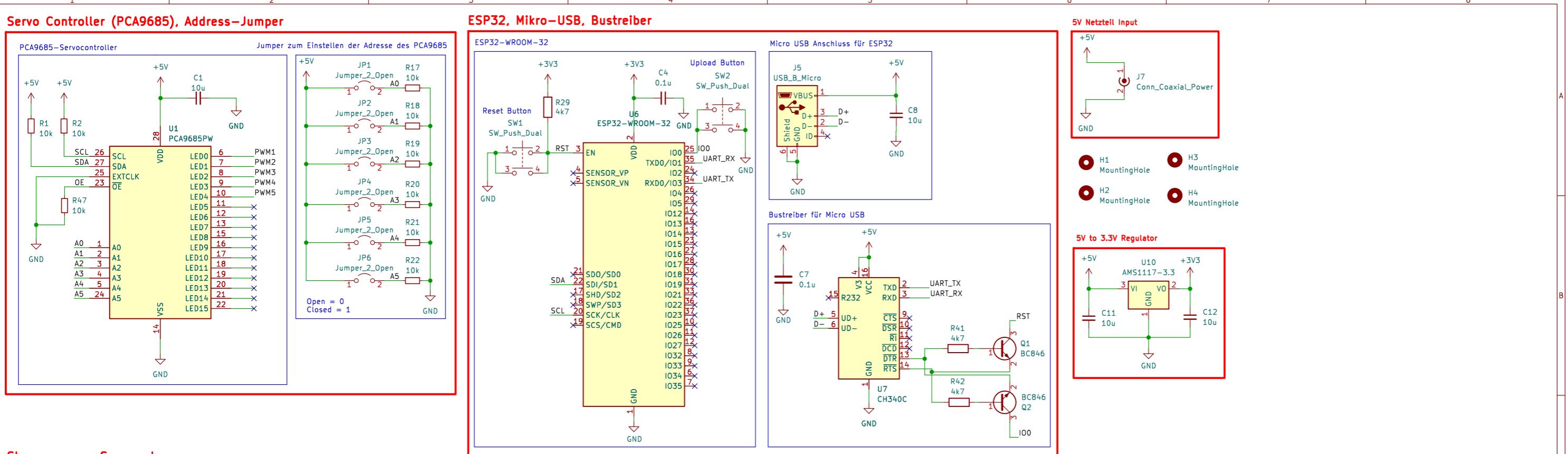
dritte Version

Sheet: /  
File: Handschuh\_v3.kicad\_sch

**Title: Schaltplan Handschuh**

Size: A3 Date: 2023-11-30  
KiCad E.D.A. kicad 7.0.7

Rev: v3  
Id: 1/1



erste Version

Sheet: /  
File: Roboterhand.kicad\_sch

Title: Schaltplan\_Roboterhand

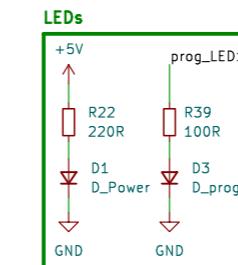
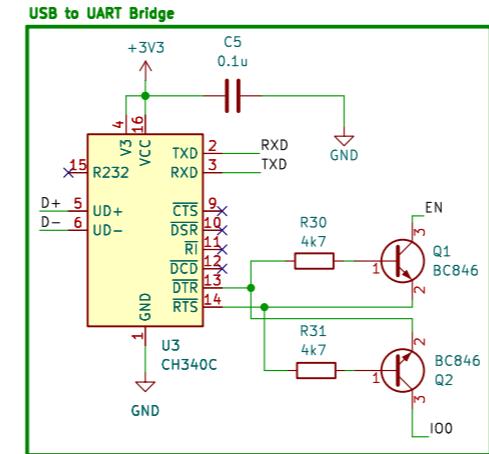
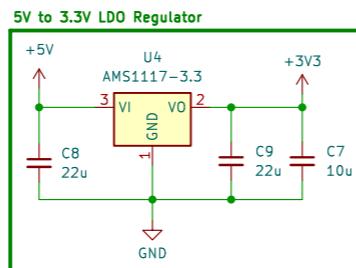
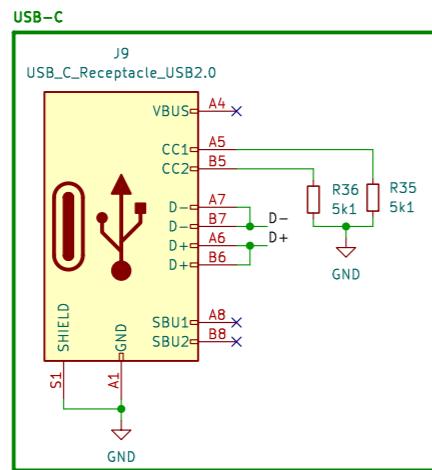
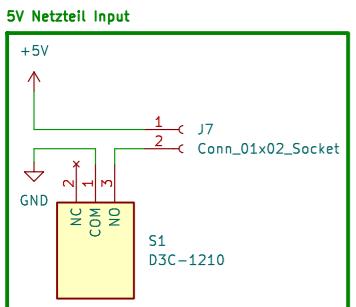
Size: A3 Date: 2023-06-27  
KiCad EDA kicad 7.0.7

KiCad E.D.A. kiCad 7.0.7

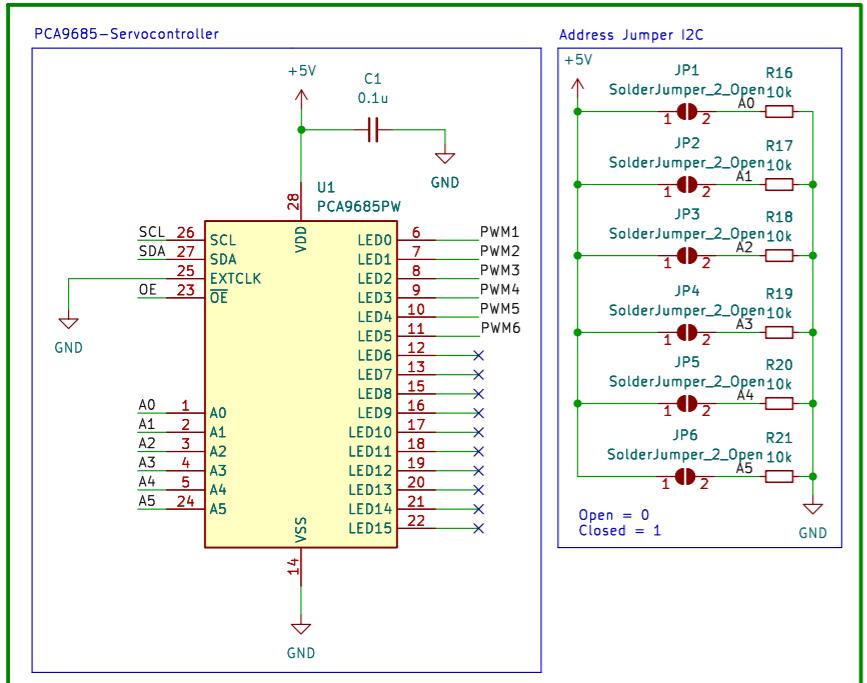
Rev: v1  
Id: 1 / 1

8

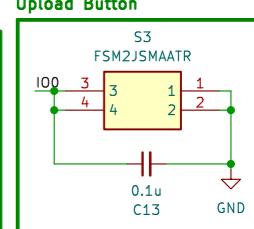
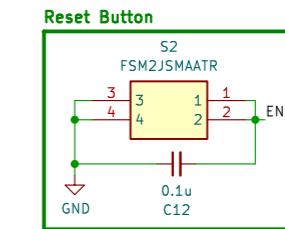
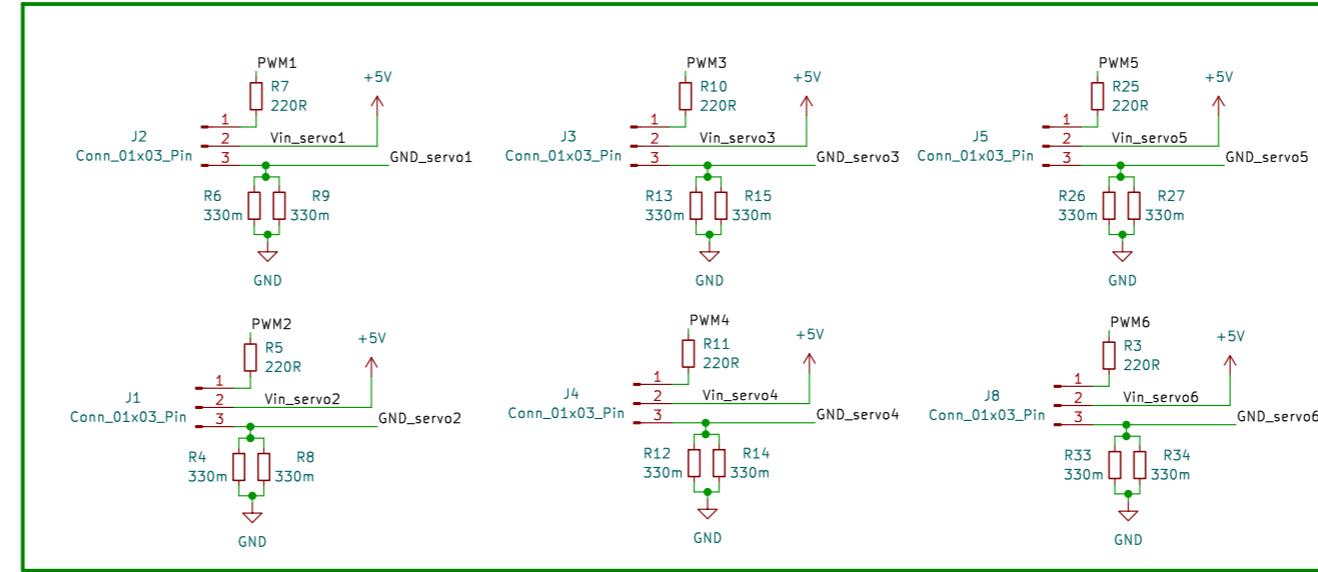
# Roboterhand



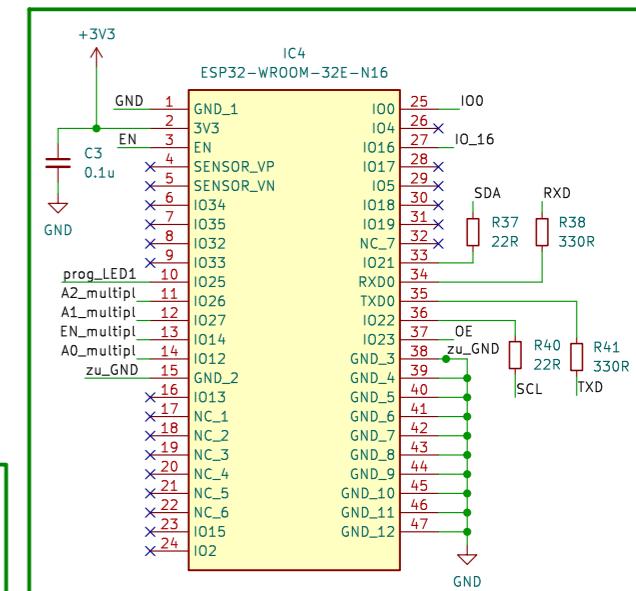
## Servo PWM Controller (PCA9685)



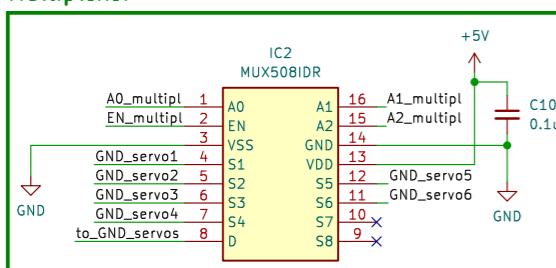
## Messschaltung und Servo Anschlüsse



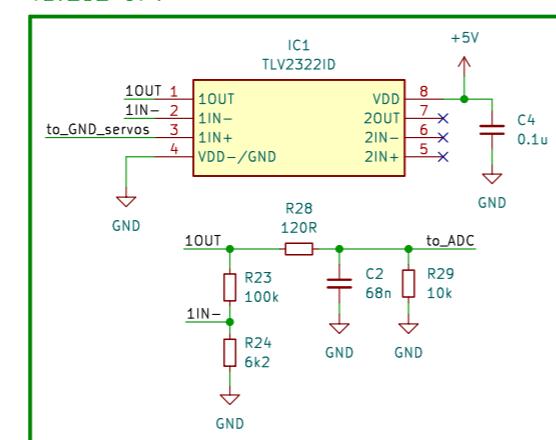
## ESP32-WROOM-32E-N16



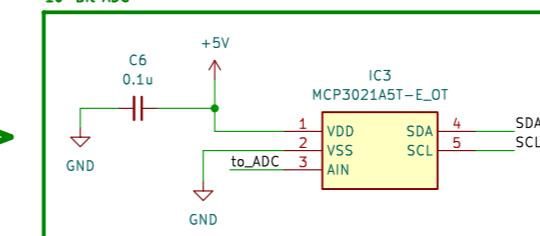
## Multiplexer



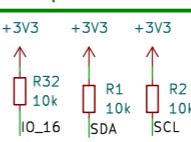
## TLV232 OPV



## 10-Bit ADC



## Pull-Ups



zweite Version

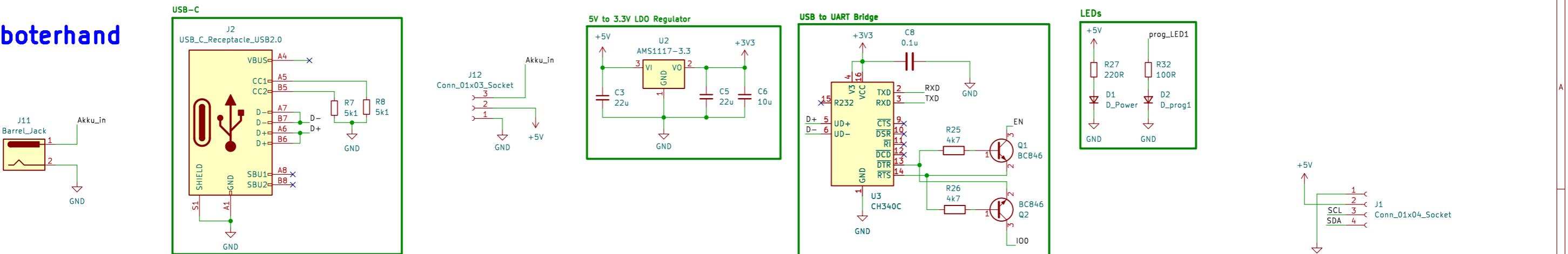
Sheet: /  
File: Roboterhand.kicad\_sch

**Title: Schaltplan Roboterhand**

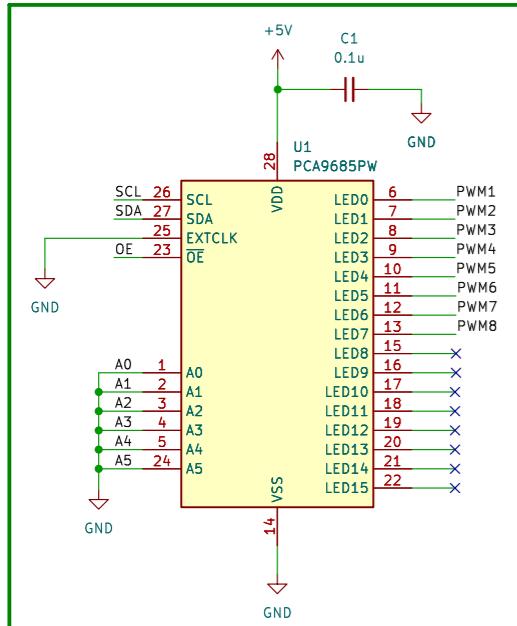
Size: A3 Date: 2023-10-18  
KiCad E.D.A. kicad 7.0.7

Rev: v2  
Id: 1/1

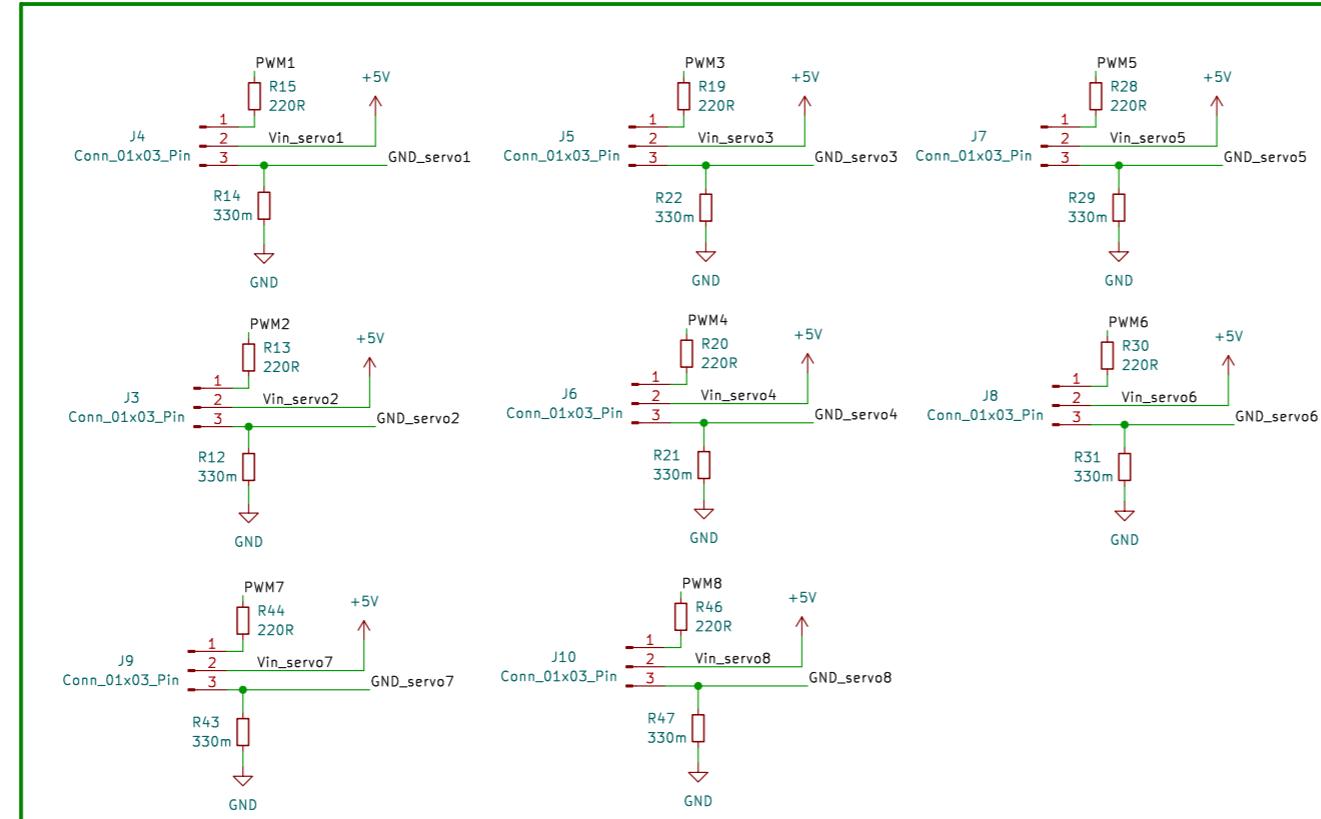
# Roboterhand



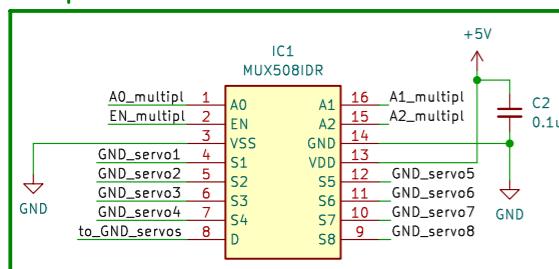
## Servo PWM Controller (PCA9685)



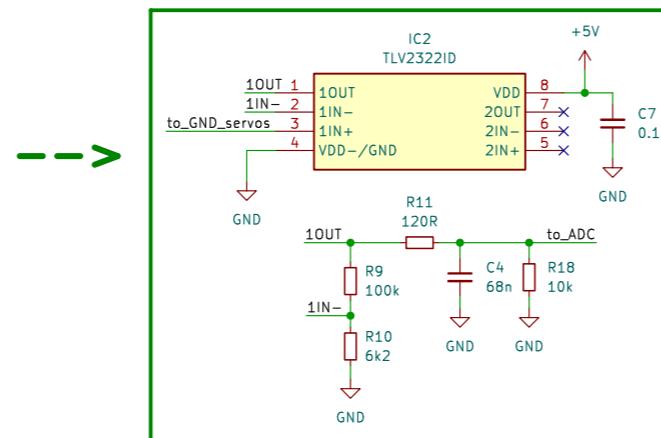
## Messschaltung und Servo Anschlüsse



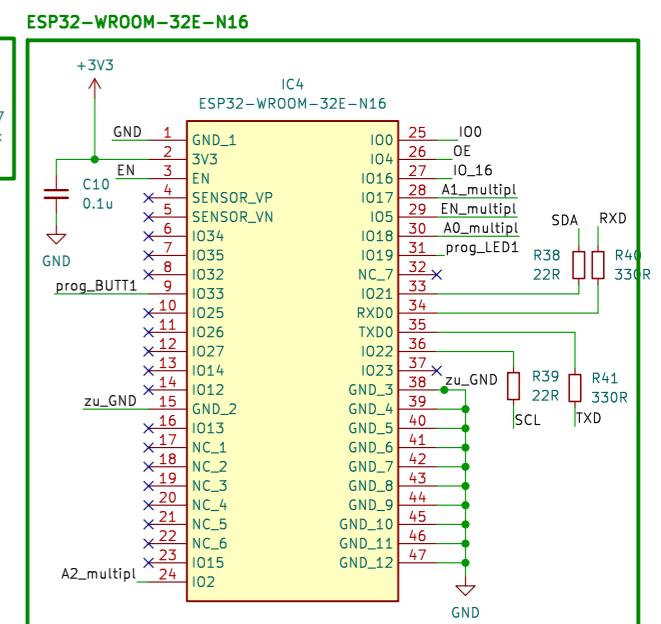
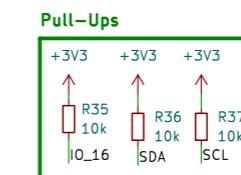
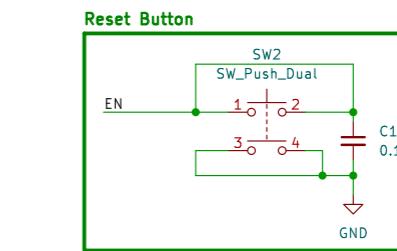
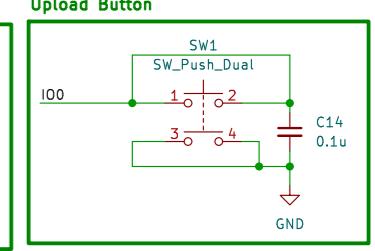
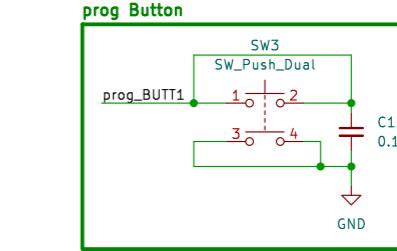
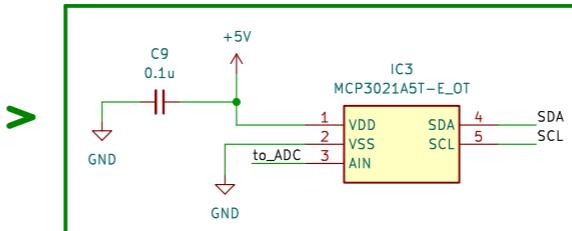
## Multiplexer



## TLV232 OPV



## 10-Bit ADC



- H1 MountingHole
- H2 MountingHole
- H3 MountingHole
- H4 MountingHole

dritte Version

Sheet: /  
File: Roboterhand\_v3.kicad\_sch

**Title: Schaltplan Roboterhand**

Size: A3 Date: 2023-11-30  
KiCad E.D.A. kicad 7.0.7

Rev: v3  
Id: 1/1

## 16.2.3 Platinenlayouts

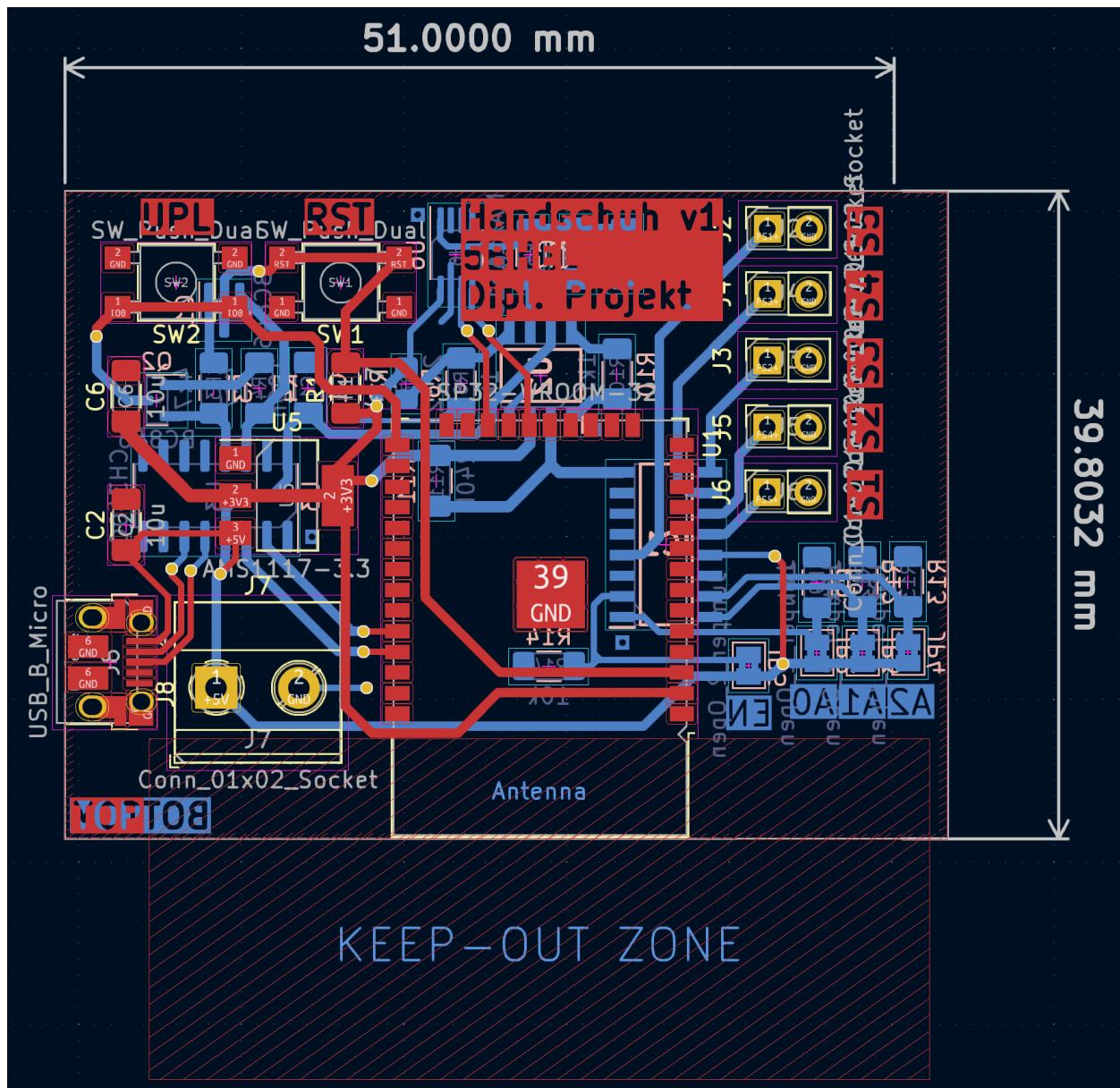


Abbildung 108: PCB Layout Eingabesubsystem Version 1

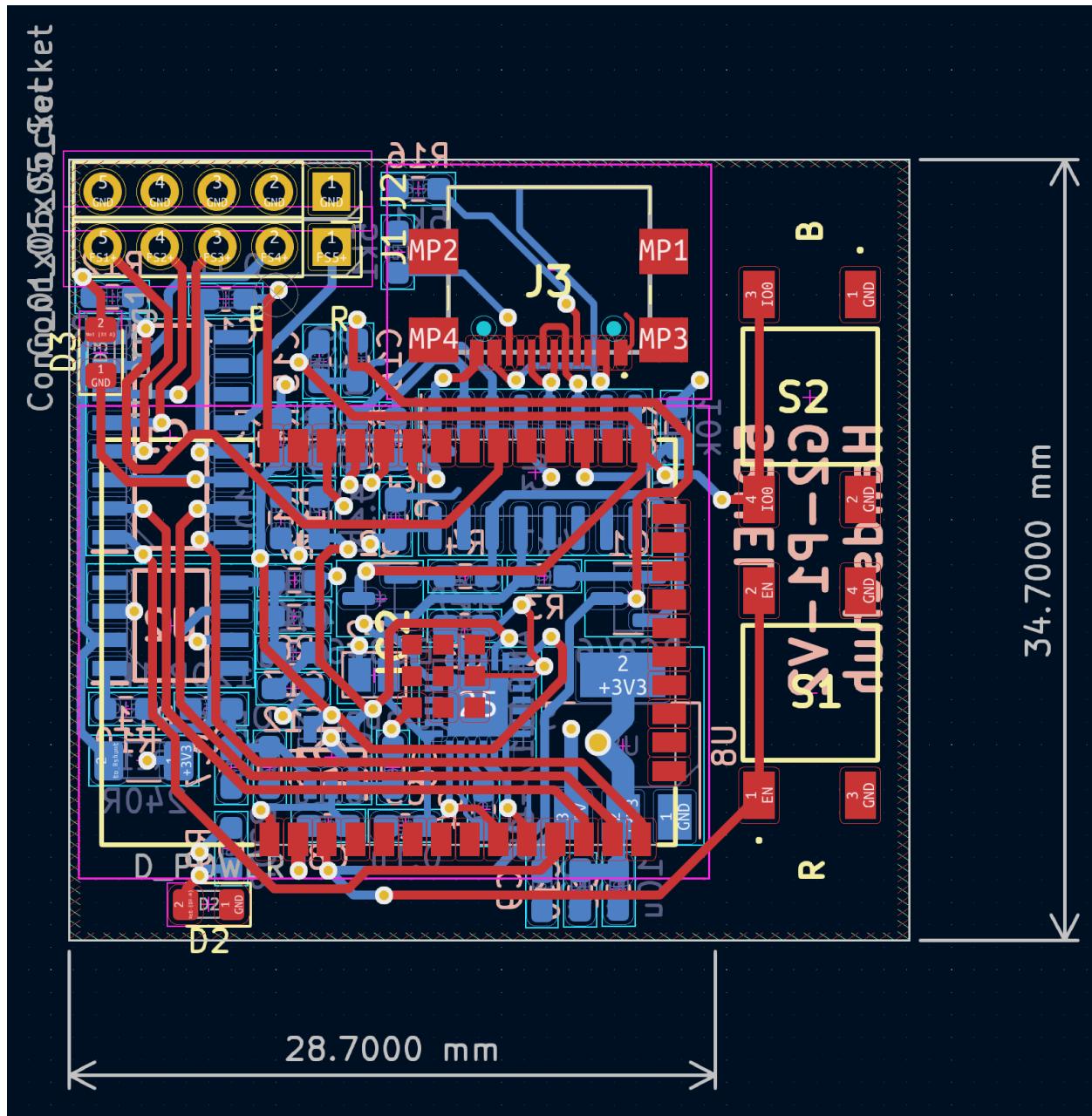


Abbildung 109: PCB Layout Eingabesubsystem Version 2

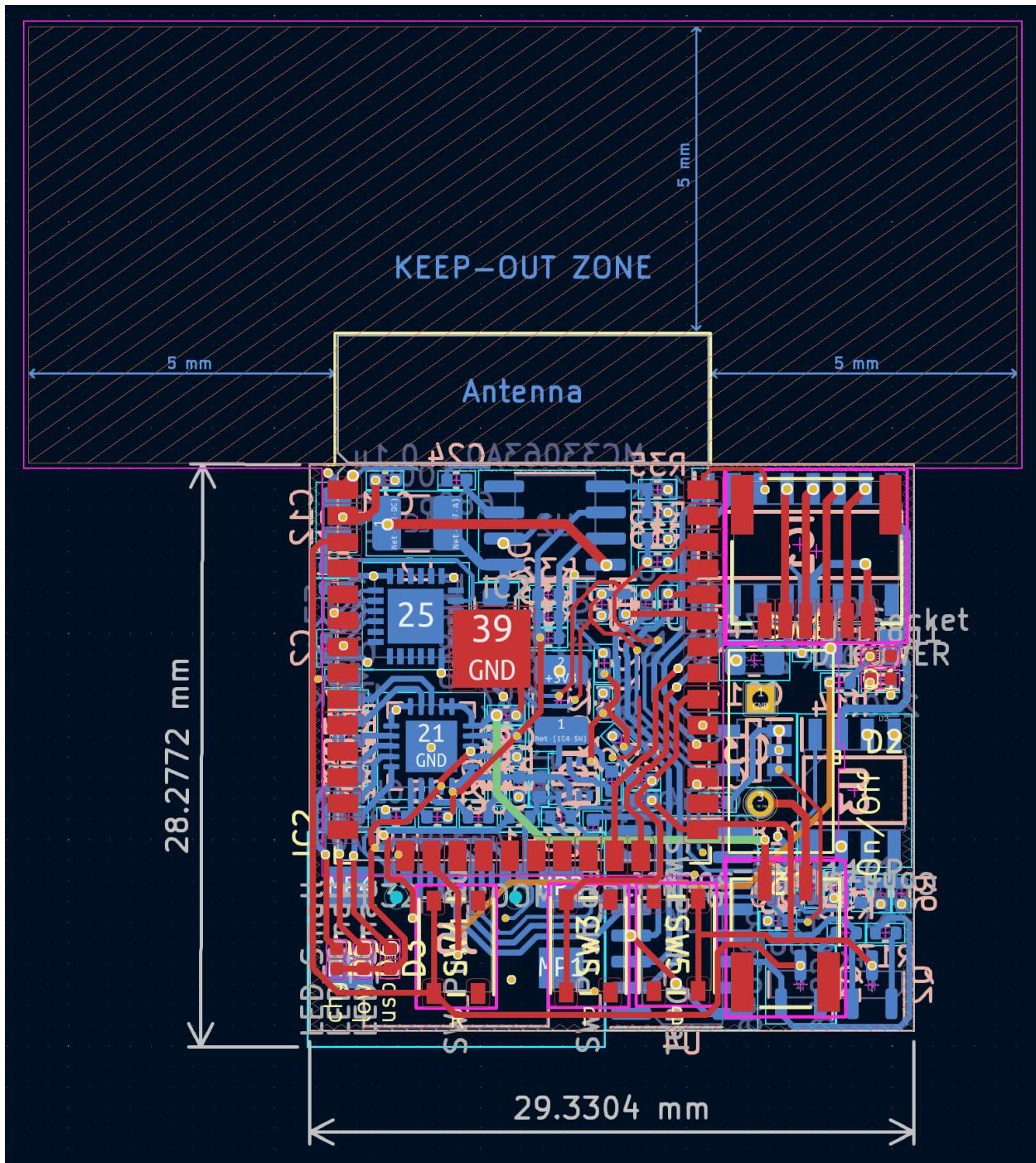


Abbildung 110: PCB Layout Eingabesubsystem Version 3

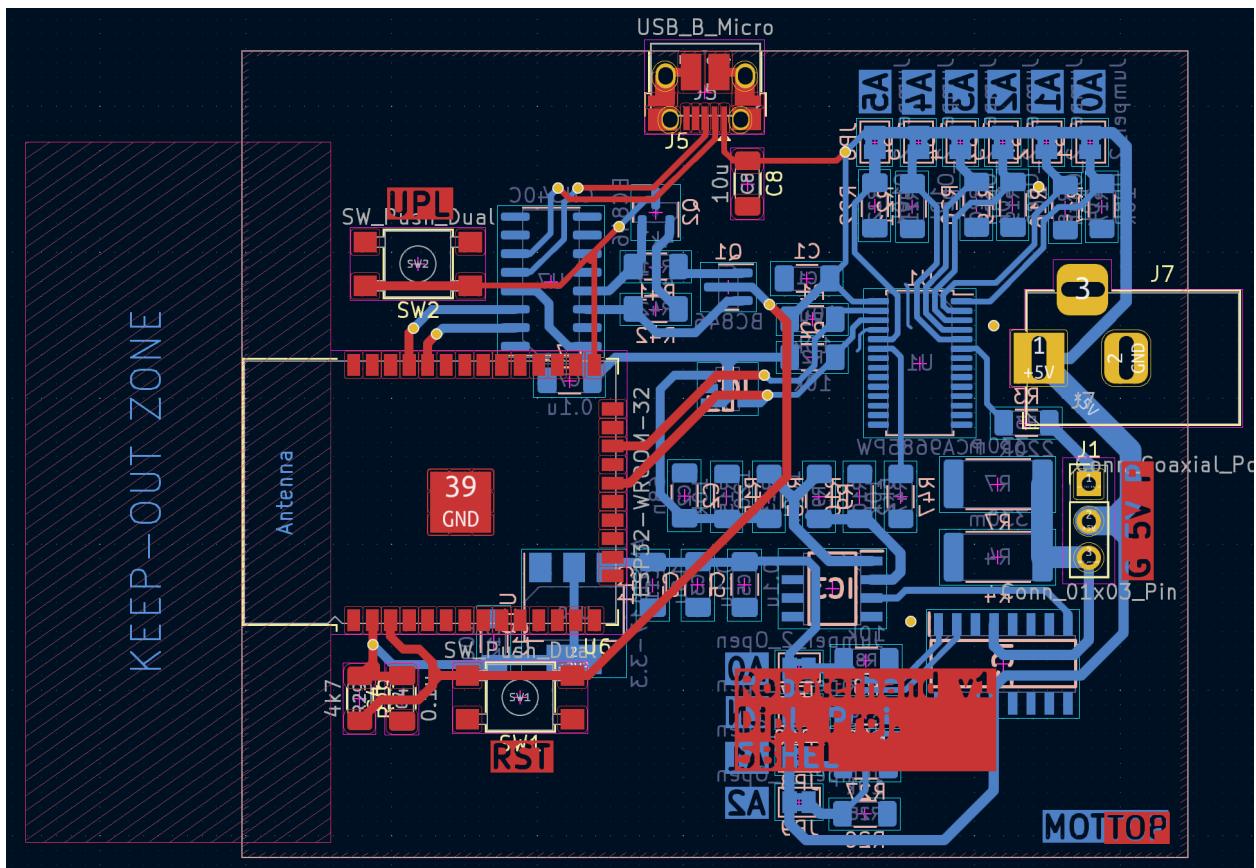


Abbildung 111: PCB Layout Ausgabesubsystem Version 1

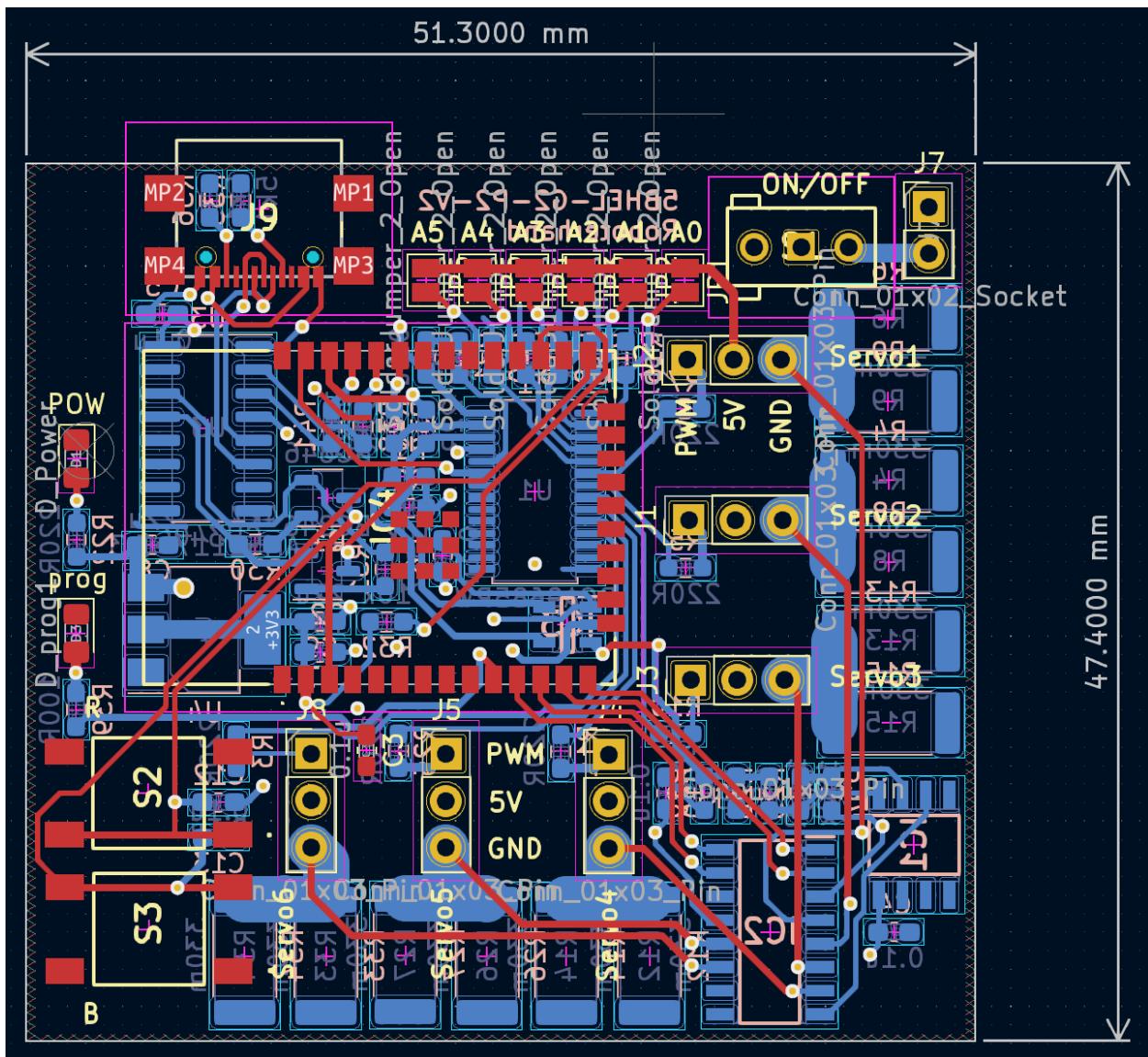


Abbildung 112: PCB Layout Ausgabesubsystem Version 2

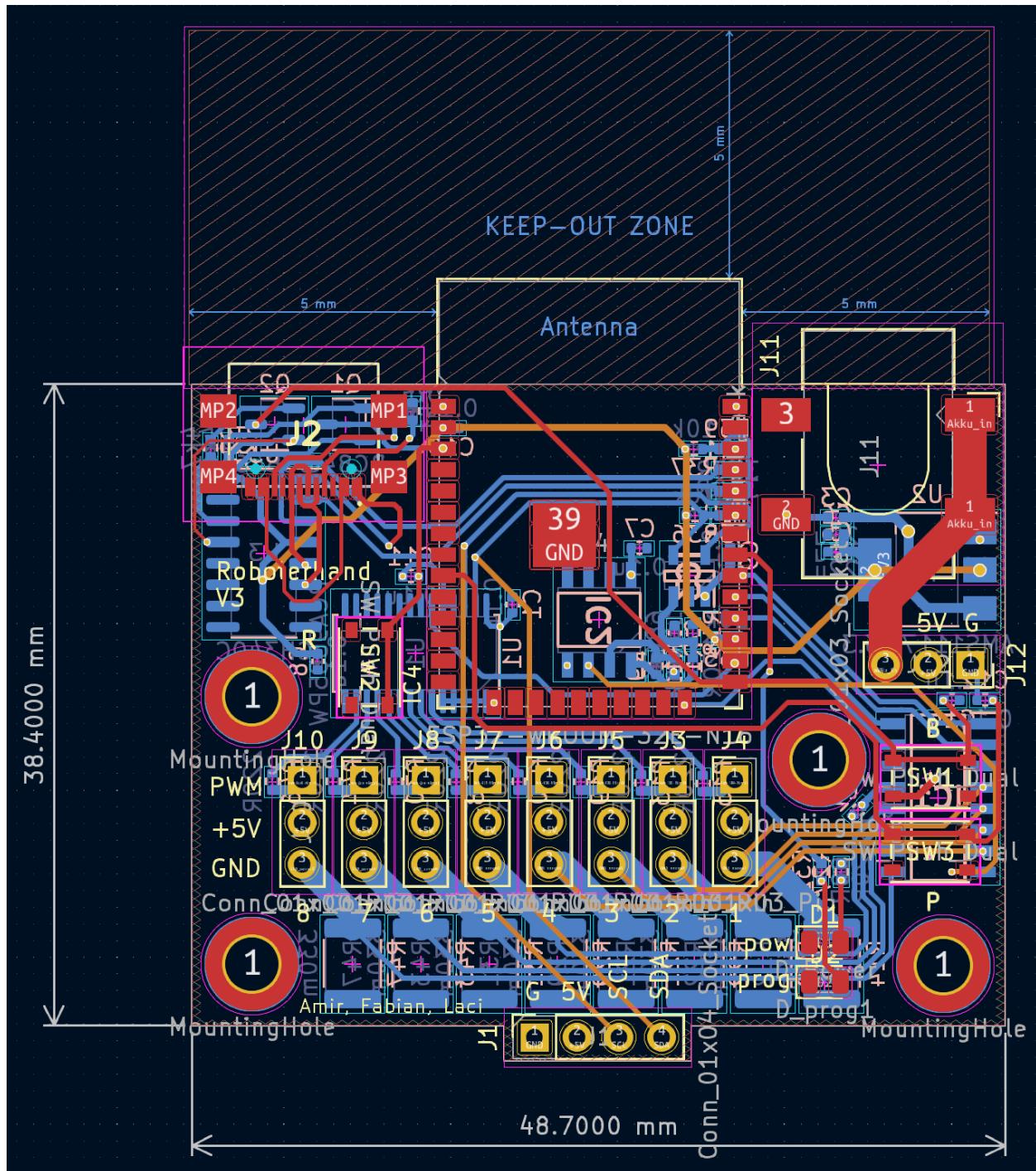


Abbildung 113: PCB Layout Ausgabesubsystem Version 3

#### 16.2.4 Bestückungslisten

References	Value	Footprint	Quantity
C2, C6	10u	C_1206_3216Metric_Pad1. 33x1.80mm_HandSolder	2
C1	100n	C_1206_3216Metric_Pad1. 33x1.80mm_HandSolder	1
C7	68n	C_1206_3216Metric_Pad1. 33x1.80mm_HandSolder	1
R6, R9, R12, R13, R14	10k	C_1206_3216Metric_Pad1. 33x1.80mm_HandSolder	5
R1, R3, R4	4k7	C_1206_3216Metric_Pad1. 33x1.80mm_HandSolder	3
R5	120R	C_1206_3216Metric_Pad1. 33x1.80mm_HandSolder	1
R7	2k32	C_1206_3216Metric_Pad1. 33x1.80mm_HandSolder	1
R8	1k	C_1206_3216Metric_Pad1. 33x1.80mm_HandSolder	1
R10	1k1	C_1206_3216Metric_Pad1. 33x1.80mm_HandSolder	1
R11	240R	C_1206_3216Metric_Pad1. 33x1.80mm_HandSolder	1
U1	ESP32-WROOM-32	ESP32-WROOM-32	1
U2	INA129	SOIC127P600X175-8N	1
U3	CH340C	SOIC-16_3.9x9.9mm _P1.27mm	1
U5	AMS1117-3.3	SOT-223-3_TabPin2	1
U6	MAX11613	MSOP-8_3x3mm_P0.65mm	1
SW1, SW2	SW_Push_Dual	SW_SPST_TL3305A	2
JP2, JP3, JP4, JP5	Jumper_2_Open	SolderJumper-2_P1.3mm _Bridged2Bar_Pad1.0x1.5mm	4
Q1, Q2	BC846	SOT-23	2
IC1	MUX508IDR	SOIC127P600X175-16N	1
J2, J3, J4, J5, J6	Conn_01x02_Socket	PinSocket_1x02_P2.54mm _Vertical	5

Tabelle 9: Betückungsliste Platine Eingabesubsystem Version 1

References	Value	Footprint	Quantity
J7	Conn_01x02_Socket	TerminalBlock_Phoenix _MKDS-1,5-2-5.08_1x02 _P5.08mm_Horizontal	1
J8	USB_B_Micro	USB_Micro-B_Amphenol _10103594-0001LF_Horizontal	1

Tabelle 10: Betückungsliste Platine Eingabesubsystem Version 1

References	Value	Footprint	Quantity
C1, C4, C5, C6, C8, C11, C12, C13, C15, C16	0.1u	C_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	10
C9, C10	22u	C_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	2
C2	10u	C_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	1
C3	2.2n	C_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	1
C7	68n	C_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	1
R6, R13, R14, R15	10k	R_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	4
R2, R16	5k1	R_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	2
R3, R4	4k7	R_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	2
R17, R19	22R	R_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	2
R18, R20	330R	R_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	2
R5	120R	R_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	1
R7	2k32	R_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	1
R8	1k	R_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	1
R9	220R	R_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	1
R10	1k1	R_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	1
R11	240R	R_1206_3216Metric_Pad1. 30x1.75mm_HandSolder	1
R12	100R	R_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	1
D2	D_POWER	D_0805_2012Metric_Pad1. 15x1.40mm_HandSolder	1

Tabelle 11: Betstückungsliste Platine Eingabesubsystem Version 2

References	Value	Footprint	Quantity
D3	D_prog1	D_0805_2012Metric_Pad1. 15x1.40mm_HandSolder	1
U2	INA129	SOIC127P600X175-8N	1
U3	CH340C	SOIC-16_3.9x9.9mm _P1.27mm	1
U4	MPU-6000	QFN50P400X400X95-25N	1
U8	AMS1117-3.3	SOT-223-3_TabPin2	1
Q1, Q2	BC846	SOT-23	2
S1, S2	FSM2JSMAATR	FSM2J_1	2
IC1	MUX508IDR	SOIC127P600X175-16N	1
IC2	ESP32-WROOM-32E-N16	ESP32WROOM32EN16	1
IC4	MCP3021A5T-E_OT	SOT95P280X145-5N	1
JP2	SolderJumper_2_Open	SolderJumper-2_P1.3mm _Open_Pad1.0x1.5mm	1
J1, J2	Conn_01x05_Socket	PinSocket_1x05_P2.54mm _Vertical	2
J3	USB_C_Receptacle_USB2.0	USB4110GFA	1

Tabelle 12: Betückungsliste Platine Eingabesubsystem Version 2

References	Value	Footprint	Quantity
C1, C2, C4, C5, C6, C7, C8, C9, C11, C13, C15, C18, C19	100n	C_0402_1005Metric	13
C12, C21	47u	C_0402_1005Metric	2
C17, C26	10u	C_0402_1005Metric	2
C22, C23	4.7u	C_0402_1005Metric	2
C3	2.2n	C_0402_1005Metric	1
C10	68n	C_0402_1005Metric	1
C20	47p	C_0402_1005Metric	1
C24	680p	C_0402_1005Metric	1
R8, R13, R15, R18, R22, R25, R30, R31, R36	10k	R_0402_1005Metric	9
R16, R17, R23	4.7k	R_0402_1005Metric	3
R27, R28, R29	470R	R_0402_1005Metric	3
R32, R33, R34	1R	R_0402_1005Metric	3
R1, R3	5.1k	R_0402_1005Metric	2
R7, R24	1k	R_0402_1005Metric	2
R9, R10	22R	R_0402_1005Metric	2
R11, R12	330R	R_0402_1005Metric	2
R2	240R	R_0402_1005Metric	1
R4	1.1k	R_0402_1005Metric	1
R5	120R	R_0402_1005Metric	1
R6	2.32k	R_0402_1005Metric	1
R14	100R	R_0402_1005Metric	1
R19	220R	R_0402_1005Metric	1
R21	30.9k	R_0402_1005Metric	1
R26	100k	R_0402_1005Metric	1
R35	30k	R_0402_1005Metric	1
L1	2.7u	L_1210_3225Metric_Pad1. 42x2.65mm_HandSolder	1
L2	100u	L_1210_3225Metric_Pad1. 42x2.65mm_HandSolder	1

Tabelle 13: Betückungsliste Platine Eingabesubsystem Version 3

References	Value	Footprint	Quantity
D3, D5, D6	LED_Small	LED_0402_1005Metric	3
D1	D_prog1	LED_0402_1005Metric	1
D2	D_POWER	LED_0402_1005Metric	1
D7	1N5819	D_SOD-123	1
U1	MPU-6000	QFN50P400X400X95-25N	1
U2	MC33063AD	SOIC-8_3.9x4.9mm_P1.27mm	1
U3	INA129	SOIC127P600X175-8N	1
U4	CH340C	SOIC-16_3.9x9.9mm _P1.27mm	1
U5	MCP73871	QFN-20-1EP_4x4mm _P0.5mm_EP2.5x2.5mm	1
SW3, SW4, SW5	SW_Push_Dual	SW4 PTS815 SJM 250 SMTR LFS_CNK	3
SW1	AS11CP	SW_AS11CP	1
Q1, Q2	BC846	SOT-23	2
IC1	MUX508IDR	SOIC127P600X175-16N	1
IC2	ESP32-WROOM-32E-N16	ESP32-WROOM-32D	1
IC3	MCP3021A5T-E_OT	SOT95P280X145-5N	1
IC4	TPS563300DRLR	SOTFL50P160X60-8N	1
TH1	10k	R_0402_1005Metric	1
J1	Conn_01x06_Socket	CONN_SMO6B_JST	1
J2	Conn_01x02_Socket	CONN_SM02B-NSHSS_JST	1
J5	USB_C_Receptacle_USB2.0	USB4110GFA	1

Tabelle 14: Betückungsliste Platine Eingabesubsystem Version 3

References	Value	Footprint	Quantity
C1, C8, C11, C12	10u	C_1206_3216Metric_Pad1. 33x1.80mm_HandSolder	4
C3, C4, C5, C7	0.1u	C_1206_3216Metric_Pad1. 33x1.80mm_HandSolder	4
C2	68n	C_1206_3216Metric_Pad1. 33x1.80mm_HandSolder	1
R1, R2, R8, R15, R17, R18, R19, R20, R21, R22, R27, R28, R40, R47	10k	R_1206_3216Metric_Pad1. 33x1.80mm_HandSolder	14
R29, R41, R42	4k7	R_1206_3216Metric_Pad1. 33x1.80mm_HandSolder	3
R4, R7	330m	R_2512_6332Metric_Pad1. 40x3.35mm_HandSolder	2
R3	220R	R_1206_3216Metric_Pad1. 33x1.80mm_HandSolder	1
R9	100k	R_1206_3216Metric_Pad1. 33x1.80mm_HandSolder	1
R10	6k2	R_1206_3216Metric_Pad1. 33x1.80mm_HandSolder	1
R13	120R	R_1206_3216Metric_Pad1. 33x1.80mm_HandSolder	1
U1	PCA9685PW	TSSOP-28_4.4x9.7mm _P0.65mm	1
U6	ESP32-WROOM-32	ESP32-WROOM-32	1
U7	CH340C	SOIC-16_3.9x9.9mm _P1.27mm	1
U10	AMS1117-3.3	SOT-223-3_TabPin2	1
SW1, SW2	SW_Push_Dual	SW_SPST_TL3305A	2
JP1, JP2, JP3, JP4, JP5, JP6, JP7, JP8, JP9, JP10	Jumper_2_Open	SolderJumper-2_P1.3mm _Bridged2Bar_Pad1.0x1.5mm	10
Q1, Q2	BC846	SOT-23	2
IC1	MUX508IDR	SOIC127P600X175-16N	1
IC2	MCP3021A5T-E_OT	SOT95P280X145-5N	1
IC3	TLV2322ID	SOIC127P600X175-8N	1

Tabelle 15: Bestückungsliste Platine Ausgabesubsystem Version 1

References	Value	Footprint	Quantity
J1	Conn_01x03_Pin	PinHeader_1x03_P2.54mm_Vertical	1
J5	USB_B_Micro	USB_Micro-B_Amphenol_10103594-0001LF_Horizontal	1
J7	Conn_Coaxial_Power	BarrelJack_Horizontal	1

Tabelle 16: Betückungsliste Platine Ausgabesubsystem Version 1

References	Value	Footprint	Quantity
C1, C4, C5, C6, C8, C11, C12, C13	0.1u	C_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	8
C8, C9	22u	C_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	4
C2	68n	C_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	1
C7	10u	C_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	1
R4, R6, R8, R9, R12, R13, R14, R15, R26, R27, R33, R34	330m	R_2512_6332Metric_Pad1. 40x3.35mm_HandSolder	12
R1, R2, R16, R17, R18, R19, R20, R21, R29, R32	10k	R_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	10
R3, R5, R7, R10, R11, R22, R25	220R	R_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	7
R30, R31	4k7	R_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	2
R35, R36	5k1	R_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	2
R37, R40	22R	R_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	2
R23	100k	R_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	1
R24	6k2	R_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	1
R28	120R	R_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	1
R39	100R	R_0603_1608Metric_Pad1. 08x0.95mm_HandSolder	1
D1	D_Power	D_0805_2012Metric_Pad1. 15x1.40mm_HandSolder	1
D3	D_prog1	D_0805_2012Metric_Pad1. 15x1.40mm_HandSolder	1
U1	PCA9685PW	TSSOP-28_4.4x9.7mm _P0.65mm	1
U3	CH340C	SOIC-16_3.9x9.9mm _P1.27mm	1

Tabelle 17: Betstückungsliste Platine Ausgabesubsystem Version 2

References	Value	Footprint	Quantity
U4	AMS1117-3.3	SOT-223-3_TabPin2	1
JP1, JP2, JP3, JP4, JP5, JP6	Jumper_2_Open	SolderJumper-2_P1.3mm_Bridged2Bar_Pad1.0x1.5mm	6
Q1, Q2	BC846	SOT-23	2
S2, S3	FSM2JSMAATTR	FSM2J_1	2
IC1	TLV2322ID	SOIC127P600X175-8N	1
IC2	MUX508IDR	SOIC127P600X175-16N	1
IC3	MCP3021A5T-E_OT	SOT95P280X145-5N	1
IC4	ESP32-WROOM-32E-N16	ESP32WROOM32EN16	1
S1	D3C-1210	D3C1210	1
J1, J2, J3, J4, J5, J8	Conn_01x03_Pin	PinHeader_1x03_P2.54mm_Vertical	6
J7	Conn_01x02_Socket	PinHeader_1x02_P2.54mm_Vertical	1
J9	USB_C_Receptacle_USB2.0	USB4110GFA	1

Tabelle 18: Betückungsliste Platine Ausgabesubsystem Version 2

References	Value	Footprint	Quantity
C1, C2, C7, C8, C9, C10, C11, C12, C14	0.1u	C_0402_1005Metric	9
C3, C5	22u	C_0402_1005Metric	2
C4	68n	C_0402_1005Metric	1
C6	10u	C_0402_1005Metric	1
D1	D_Power	D_0805_2012Metric_Pad1. 15x1.40mm_HandSolder	1
D2	D_prog1	D_0805_2012Metric_Pad1. 15x1.40mm_HandSolder	1
D3	D_Zener_Small	D_SOD-123	1
H1, H2, H3, H4	MoutingHole	MountingHole_3.2mm_M3 _DIN965_Pad_TopBottom	4
IC1	MUX508IDR	SOIC127P600X175-16N	1
IC2	TLV2322ID	SOIC127P600X175-8N	1
IC3	MCP3021A5T-E_OT	SOT95P280X145-5N	1
IC4	ESP32-WROOM-32E-N16	ESP32WROOM32EN16	1
J1	Conn_01x04_Socket	PinHeader_1x04_P2.54mm _Vertical	1
J9	USB_C_Receptacle_USB2.0	USB4110GFA	1
J3, J4, J5, J6, J7, J8, J9, J10	Conn_01x03_Pin	PinHeader_1x03_P2.54mm _Vertical	8
J11	Barrel_Jack	BarrelJack_CLIFF_FC681465 S_SMT_Horizontal	1
J12	Conn_01x03_Socket	PinSocket_1x03_P2.54mm _Vertical	1
Q1, Q2	BC846	SOT-23	2
Q3	PMOS	TO-252-3_TabPin2	1
R7, R8	5k1	R_0402_1005Metric	2
R9, R42	100k	R_0402_1005Metric	2
R10	6k2	R_0402_1005Metric	1
R11	120R	R_0402_1005Metric	1
R12, R14, R21, R22, R29, R31, R43, R47	330m	R_1812_4532Metric	8

Tabelle 19: Betückungsliste Platine Ausgabesubsystem Version 3

References	Value	Footprint	Quantity
R13, R15, R19, R20, R27, R28, R30, R44, R46	220R	R_0402_1005Metric	9
R18, R35, R36, R37	10k	R_0402_1005Metric	4
R25, R26	4k7	R_0402_1005Metric	2
R32	100R	R_0402_1005Metric	1
R38, R39	22R	R_0402_1005Metric	2
R40, R41	330R	R_0402_1005Metric	2
SW1, SW2, SW3	SW_Push_Dual	SW4 PTS815 SJM 250 SMTR LFS_CNK	3
U1	PCA9685PW	TSSOP-28_4.4x9.7mm _P0.65mm	1
U2	AMS1117-3.3	SOT-223-3_TabPin2	1
U3	CH340C	SOIC-16_3.9x9.9mm _P1.27mm	1

Tabelle 20: Betückungsliste Platine Ausgabesubsystem Version 3

## 16.3 Software

### 16.3.1 Skizzen und Konzepte

### 16.3.2 Diagramme

## 16.4 Software

### 16.4.1 Skizzen und Konzepte

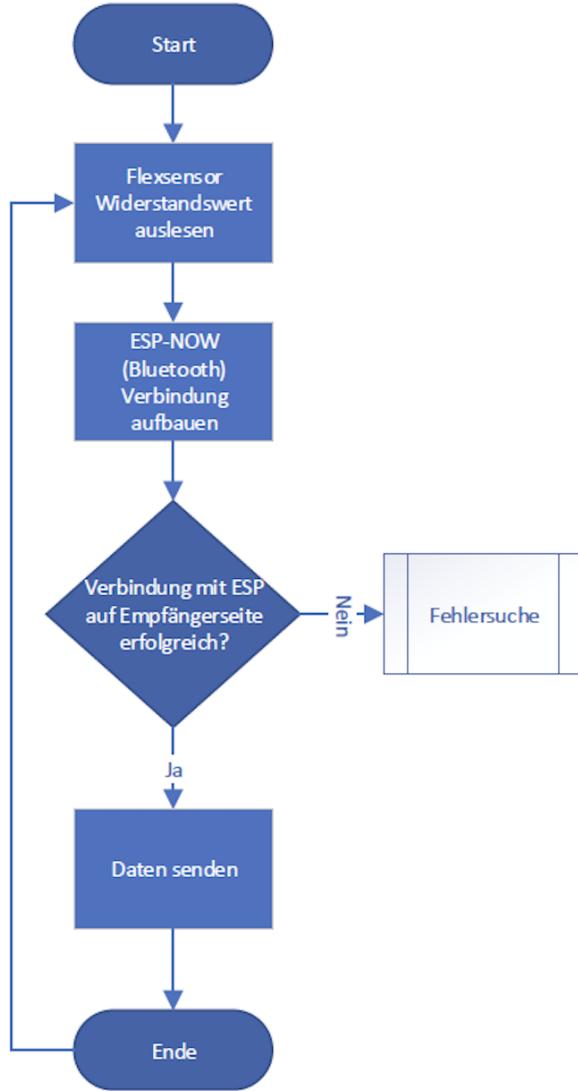


Abbildung 114: Softwarekonzept Eingabesubsystem

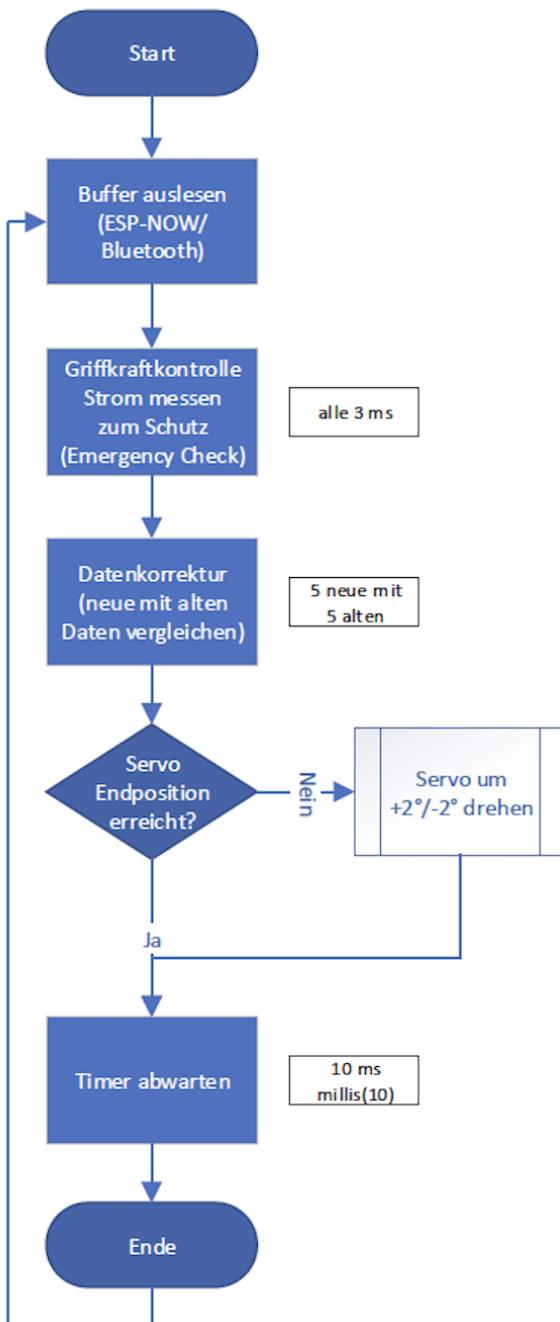


Abbildung 115: Softwarekonzept Ausgabesubsystem

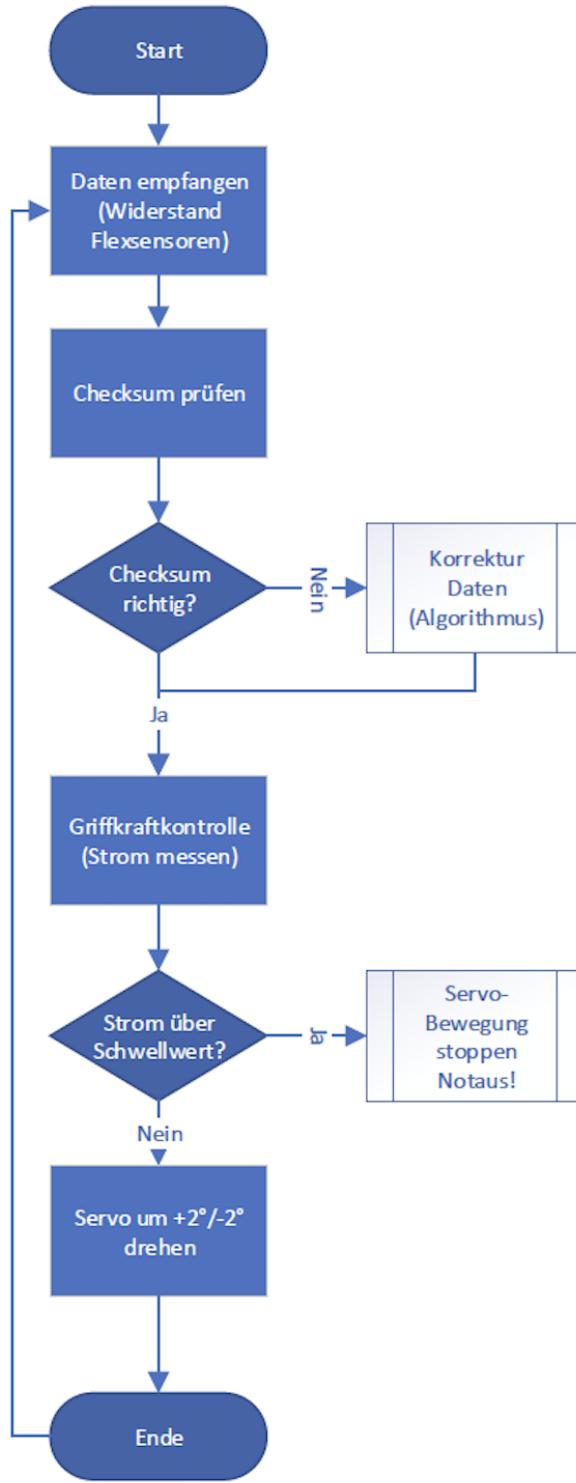


Abbildung 116: Softwarekonzept Teilbereich Servostellung

### 16.4.2 Code

#### 16.4.2.1 Eingabesubsystem

#### 16.4.2.2 Ausgabesubsystem

#### 16.4.2.3 User-Interface

---

## 17 Abbildungsverzeichnis

Abbildung 1	Flexsensor
2	Kostenkalkulationen fertig
3	Prototypen proof of concept
4	Hardwaredesign fertig
5	PCB-Design fertig
6	Softwareimplementierung für Integrationstest fertig
7	Integration aller Komponenten
8	User Interface fertiggestellt
9	Abnahme durch die Projektbetreuer

---

## 18 Tabellenverzeichnis

2	Kostenkalkulationen fertig
3	Prototypen proof of concept
4	Hardwaredesign fertig
5	PCB-Design fertig
6	Softwareimplementierung für Integrationstest fertig
7	Integration aller Komponenten
8	User Interface fertiggestellt
9	Abnahme durch die Projektbetreuer

---

## 19 Literaturverzeichnis