

# 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:



## Eidesstaatliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzen Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

---

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.
----------------	---

Illustrative graph, photo  
(with explanation)



**Die Schule der Technik**  
Final construction of the System

Participation in competitions,  
Awards

Accesibility 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 . . . . .	1
1.4 Untersuchungsanliegen der individuellen Themenstellungen . . . . .	1
<b>2 Lastenheft</b>	<b>2</b>
<b>3 Vorwissenschaftlicher Theorieteil</b>	<b>2</b>
3.1 Platzhalter Thema Amir . . . . .	2
3.2 Platzhalter Thema Fabian . . . . .	2
3.3 Sicherheitsaspekte der Mensch-Roboter-Kollaboration . . . . .	2
<b>4 Theoretische Grundlagen des Diplomprojekts</b>	<b>2</b>
4.1 ESP-NOW Fabian . . . . .	2
4.1.1 OSI-Modell . . . . .	4
4.1.1.1 Physical Layer . . . . .	4
4.1.1.2 Data Link Layer . . . . .	5
4.1.1.3 Network Layer . . . . .	5
4.1.1.4 Transport Layer . . . . .	5
4.1.1.5 Session Layer . . . . .	5
4.1.1.6 Presentation Layer . . . . .	5
4.1.1.7 Application Layer . . . . .	6
4.1.1.8 ESP-NOW Schichten . . . . .	6
4.1.1.9 IEEE 802.15.4 Standard . . . . .	6
4.2 OPV Technologien Laci . . . . .	6
4.2.1 Single-Supply . . . . .	7
4.2.2 Dual-Supply . . . . .	7
4.2.3 Rail-to-Rail Technologie . . . . .	7
<b>5 Grundlegende Systemkonzepte</b>	<b>9</b>
5.1 Gesamtsystemkonzept . . . . .	9
5.1.1 Systembeschreibung . . . . .	9
5.2 Eingabesubsystem . . . . .	10
5.3 Ausgabesubsystem . . . . .	11
Al-Maytah, Schweitzer, Szabo	12

<b>6 Mechanische Realisierung</b>	<b>11</b>
6.1 Handschuh <i>Amir</i> . . . . .	11
6.1.1 Platzhalter für diverse Unterpunkte von <i>Amir</i> . . . . .	11
6.2 Roboterhand <i>Amir</i> . . . . .	11
6.2.1 Der Grundgedanke . . . . .	11
6.2.2 Versuchsaufbauten . . . . .	12
6.2.2.1 Aufbau . . . . .	12
6.2.2.2 Versuchsresultat . . . . .	12
6.2.3 Erste Testhand . . . . .	12
6.2.3.1 Mechanismus . . . . .	12
6.2.3.2 Realisierung und Zusammenbau . . . . .	13
6.2.3.3 Fazit und Ergebnisse . . . . .	13
6.2.4 Zweite Testhand (Inmoov) . . . . .	13
6.2.4.1 Einführung . . . . .	13
6.2.4.2 Realisierung und Zusammenbau . . . . .	13
6.2.4.3 Fazit und Ergebnisse . . . . .	13
6.2.5 Dritte Testhand (Projekt-Silikonhand) . . . . .	13
6.2.5.1 Konzepteklärung . . . . .	13
6.2.5.2 Erste Versuche einer Gussform und Aswertung . . . . .	13
6.2.5.3 Verbesserungen und Realisierung (Gelenke im Guss) . . . . .	13
6.2.5.4 Verbesserung der Gelenke . . . . .	13
6.2.5.5 Entwurf einer neuen Gussform . . . . .	13
6.2.5.6 Vorbereitung und Probleme beim Guss . . . . .	14
6.2.5.7 Konzeptverbesserung . . . . .	14
6.2.5.8 Fazit und Stellungnahme (Konzept verworfen) . . . . .	14
6.2.6 Vierte Hand . . . . .	14
6.2.6.1 Erfahrung und Konzepteklärung . . . . .	14
6.2.6.2 Erster Entwurf . . . . .	14
6.2.6.3 Verbesserungen – zweiter Entwurf . . . . .	14
6.2.6.4 Realisierung und Aufbau . . . . .	14
6.2.6.5 Versuche... . . . . .	14
<b>7 Hardware Realisierung</b>	<b>14</b>
7.1 Eingabesubsystem . . . . .	14
7.1.1 Grundlegende Voraussetzungen <i>Laci</i> . . . . .	14
7.1.2 Überlegungen, Simulationen und Berechnungen <i>Laci</i> . . . . .	15
7.1.2.1 Bewegungserfassung der Fingerbeugung . . . . .	15
7.1.2.2 Auslesen der Sensoren . . . . .	15
7.1.2.3 Berechnung des Tiefpassfilters . . . . .	17
7.1.2.4 Berechnung der OPV Verstärkung und Dimensionierung des Shuntwiderstands . . . . .	17
7.1.2.5 Umwandlung der Differenzwerte in ein geeignetes Format . .	17
7.1.2.6 Vervielfachung der Schaltung für alle Flexsensoren . . . .	18
7.1.2.7 Bewegungserfassung der Handgelenksdrehung . . . . .	19
7.1.2.8 Mikrokontroller . . . . .	19

7.1.2.9	Akkuversorgung . . . . .	20
7.1.3	Versuchsaufbauten und Messungen <b>Laci</b> . . . . .	21
7.1.3.1	Messschaltung der Flexsensoren . . . . .	21
7.1.3.2	Messergebnisse . . . . .	22
7.1.4	Schaltungsdesign <b>Laci</b> . . . . .	25
7.1.4.1	Externe Anschlüsse . . . . .	25
7.1.4.2	Mikrokontroller . . . . .	27
7.1.4.3	Mikrokontroller Buttons . . . . .	28
7.1.4.4	Status LEDs . . . . .	28
7.1.4.5	Multiplexer . . . . .	29
7.1.4.6	Operationsverstärker . . . . .	29
7.1.4.7	Analog-Digital-Wandler . . . . .	30
7.1.4.8	Gyroskop-Sensor . . . . .	30
7.1.4.9	Akku Versorgung . . . . .	31
7.1.5	Platinendesign <b>Laci</b> . . . . .	32
7.2	Roboterhand . . . . .	32
7.2.1	Grundlegende Voraussetzungen . . . . .	32
7.2.2	Simulationen und Versuchsaufbauten <b>Laci</b> . . . . .	33
7.2.3	Überlegungen und Dimensionierung <b>Laci</b> . . . . .	33
7.2.4	Schaltplandesign <b>Laci</b> . . . . .	33
7.2.5	Platinendesign <b>Laci</b> . . . . .	33
<b>8</b>	<b>Software Realisierung</b> . . . . .	<b>33</b>
8.1	Handschuh . . . . .	33
8.1.1	Konzepte und Überlegungen <b>Fabian</b> . . . . .	33
8.1.1.1	Datenübertragung . . . . .	33
8.1.1.2	Programmiersprache/Entwicklungsumgebung . . . . .	33
8.1.1.3	Benutzerfreundlichkeit . . . . .	34
8.1.1.4	Testen . . . . .	34
8.1.1.5	Allgemeines Konzept . . . . .	35
8.1.1.6	Minimaler & maximaler Widerstandswert . . . . .	35
8.1.1.7	Verworfene Ideen . . . . .	36
8.1.2	Realisierung und Gliederung <b>Fabian</b> . . . . .	36
8.1.2.1	Realisierung . . . . .	36
8.1.2.2	Gliederung . . . . .	37
8.2	Roboterhand . . . . .	38
8.2.1	Grundlegende Voraussetzungen . . . . .	38
8.2.2	Konzepte und Überlegungen <b>Fabian</b> . . . . .	38
8.2.2.1	Testen . . . . .	39
8.2.2.2	Allgemeines Konzept . . . . .	39
8.2.3	Realisierung und Gliederung <b>Fabian</b> . . . . .	39
8.2.3.1	Realisierung . . . . .	39
8.3	User Interface . . . . .	42
8.3.1	Grundlegende Voraussetzungen . . . . .	42
8.3.2	Entwicklungsumgebung <b>Laci</b> . . . . .	42

8.3.3 Konzepte und Überlegungen <b>Laci</b> . . . . .	42
8.3.3.1 Allgemeines Konzept . . . . .	42
8.3.4 Realisierung und Gliederung <b>Laci</b> . . . . .	44
<b>9 Tests und Messungen</b>	<b>44</b>
9.1 Platzhalter für diverse Unterpunkte . . . . .	44
<b>10 Projektmanagement</b>	<b>44</b>
10.1 Projektstrukturplan . . . . .	44
10.2 Milestoneplan . . . . .	44
10.3 Gantt-Diagramm . . . . .	45
10.4 Meeting Struktur . . . . .	45
<b>11 Fertigungsunterlagen</b>	<b>45</b>
11.1 Mechanik . . . . .	45
11.1.1 händische Skizzen . . . . .	45
11.1.2 CAD-Zeichnungen . . . . .	45
11.1.3 3D-Modelle . . . . .	45
11.2 Hardware . . . . .	45
11.2.1 Stromlaufpläne . . . . .	45
11.2.2 Platinen . . . . .	45
11.2.3 Bestückungslisten . . . . .	45
11.3 Software . . . . .	45
11.3.1 Diagramme . . . . .	45
11.3.2 Code . . . . .	45
<b>12 Ergebnisse und Erkenntnisse</b>	<b>45</b>
<b>13 Ausblick</b>	<b>45</b>
<b>14 Verzeichnisse</b>	<b>45</b>
14.1 Quellenverzeichnisse . . . . .	45
14.2 Abbildungsverzeichnis . . . . .	45
14.3 Abkürzungsverzeichnis . . . . .	45
<b>15 Anhang</b>	<b>45</b>

---

# 1 Einleitung

## 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 Fingergelenksensoren 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.

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 Punkt 1.1 zusammengefasste, Projekt entwickelt und gefertigt:

Projektleiter: Ladislau 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

## 3 Vorwissenschaftlicher Theorieteil

### 3.1 Platzhalter Thema Amir

### 3.2 Platzhalter Thema Fabian

### 3.3 Sicherheitsaspekte der Mensch-Roboter-Kollaboration

## 4 Theoretische Grundlagen des Diplomprojekts

### 4.1 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 Senden ausgetauscht werden. Dies ist auch bis zu einer maximalen Anzahl von zwanzig ESP32/ESP8266 Slaves möglich. Das Mischen der beiden genannten Boards ist 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 gespeichert werden, an die man dann etwas sendet. 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 Feedback zurückgeliefert, ob die Daten erfolgreich gesendet wurden oder nicht. Bei erfolgreichem Senden wird `ESP_NOW_SEND_SUCCESS` zurückgeliefert. Dies passiert, wenn sie erfolgreich am MAC-Layer 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_esnow_rate()` kann die ESP-NOW Rate 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 kann man dann einstellen, wann sich der ESP einschalten soll.

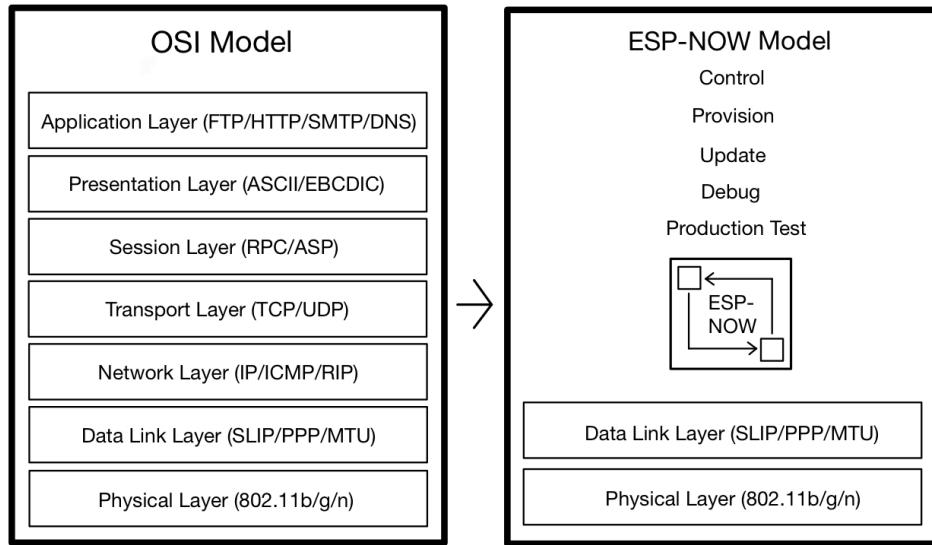
Im Freien, ohne störende Wände oder Möbel, kann ESP-NOW Daten auf jeden Fall von 150 bis 200 Metern weit übertragen. In geschlossenen Räumen funktioniert die Übertragung meist durch eine Wand, in seltenen Fällen sogar durch zwei, abhängig von der Dicke und anderen störenden Einflüssen.

Das OSI-Modell wird bei dem ESP-NOW Modell von fünf Schichten auf eine einzige reduziert. Daten müssen somit nicht den Network layer, nicht den Transport layer, nicht den Session layer, nicht den Presentation layer und nicht den Application layer durchlaufen. 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 und die

## 4.1 ESP-NOW Fabian

---

Paketverluste werden dadurch in überlasteten Netzen reduziert. Dies führt zu einer deutlichen Verbesserung der möglichen Verzögerung, indem diese sehr niedrig gehalten werden kann.



### 4.1.1 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 Schicht eine andere Aufgabe erfüllt. Durch das System der 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.

#### 4.1.1.1 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.

#### 4.1.1.2 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.

#### 4.1.1.3 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.

#### 4.1.1.4 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.

#### 4.1.1.5 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.

#### 4.1.1.6 Presentation Layer

## 4.2 OPV Technologien **Laci**

---

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.

### 4.1.1.7 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.

### 4.1.1.8 ESP-NOW Schichten

Es gibt bei ESP-NOW zwei Schichten, den Data Link Layer und den Physical Layer (siehe OSI-Modell). Dabei beruht der Physical Layer auf dem IEEE 802.15.4 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.

### 4.1.1.9 IEEE 802.15.4 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. Es werden lizenzenfreie Frequenzbänder im 2,4GHz-, 915MHz- und 868MHz-Bereich verwendet.

## 4.2 OPV Technologien **Laci**

Ein Operationsverstärker, kurz OPV, ist ein analoger Gleichspannungsverstärker, der sich durch seine üblicherweise sehr hohe Verstärkung auszeichnet. Der elektrische 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.

### 4.2.1 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.

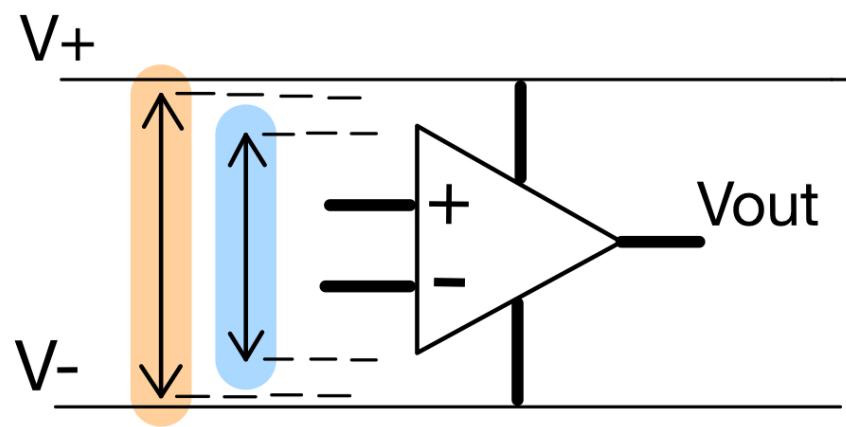
### 4.2.2 Dual-Supply

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.

### 4.2.3 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.

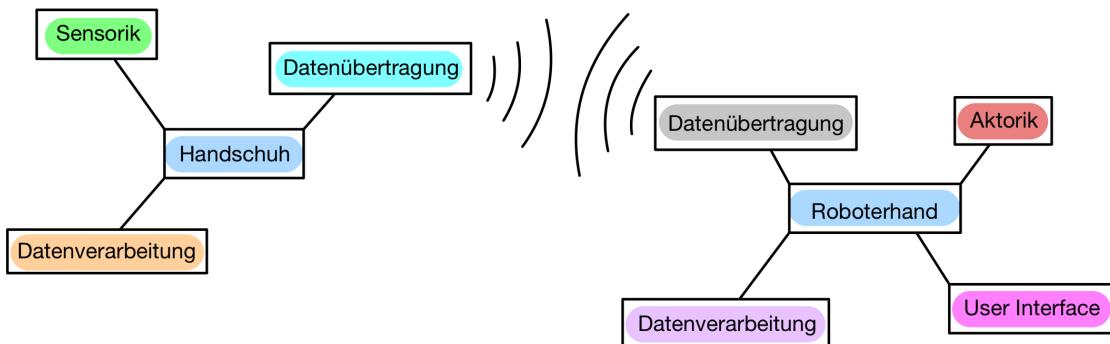
In der folgenden Grafik sind die Ausgangsspannungsbereiche eines normalen OPVs, in blau, und eines OPVs mit Rail-to-Rail Technologie, in Orange, dargestellt.



# 5 Grundlegende Systemkonzepte

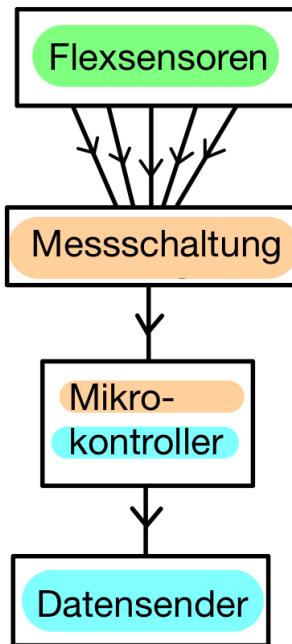
## 5.1 Gesamtsystemkonzept

### 5.1.1 Systembeschreibung



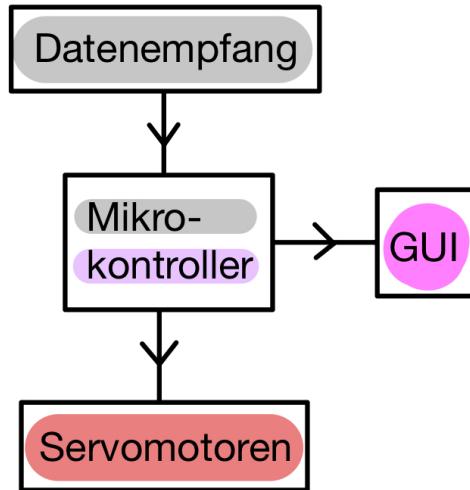
Das Gesamtsystem ist im oberhalb abgebildeten Blockschaltbild grafisch veranschaulicht. Die linke Hälfte zeigt das Konzept des Handschuhs (Eingabe) und die rechte Hälfte die Roboterhand (Ausgabe). Die grundlegende Konzeptionierung der Bewegungserfassung besteht darin, dass mithilfe von Sensoren die Fingerbeugung aller fünf Finger gemessen wird. Anschließend werden die Sensordaten ausgewertet und verarbeitet. Die resultierenden Informationen werden folglich 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



Der Sensorhandschuh 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 der korrespondierenden Sensoren. Diese Änderungen werden von einer Messschaltung erfasst, die die Sensorwerte interpretiert, digitalisiert und anschließlich an den Mikrokontroller über ein Kommunikationsprotokoll weitergibt.

### 5.3 Ausgabesubsystem



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 Punkt 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 gesteuertem Zugmechanismus.

### 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 Mechanismus

### **6.2.3.2 Realisierung und Zusammenbau**

### **6.2.3.3 Fazit und Ergebnisse**

### **6.2.4 Zweite Testhand (Inmoov)**

#### **6.2.4.1 Einführung**

#### **6.2.4.2 Realisierung und Zusammenbau**

#### **6.2.4.3 Fazit und Ergebnisse**

### **6.2.5 Dritte Testhand (Projekt-Silikonhand)**

#### **6.2.5.1 Konzepterläuterung**

#### **6.2.5.2 Erste Versuche einer Gussform und Asuwertung**

#### **6.2.5.3 Verbesserungen und Realisierung (Gelenke im Guss)**

#### **6.2.5.4 Verbesserung der Gelenke**

#### **6.2.5.5 Entwurf einer neuen Gussform**

---

#### **6.2.5.6 Vorbereitung und Probleme beim Guss**

#### **6.2.5.7 Konzeptverbesserung**

#### **6.2.5.8 Fazit und Stellungnahme (Konzept verworfen)**

### **6.2.6 Vierte Hand**

#### **6.2.6.1 Erfahrung und Konzepterläuterung**

#### **6.2.6.2 Erster Entwurf**

#### **6.2.6.3 Verbesserungen – zweiter Entwurf**

#### **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

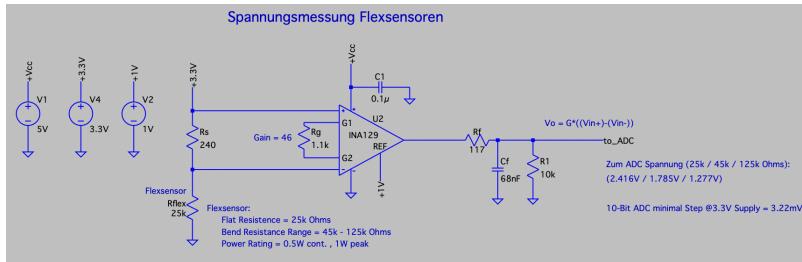
Die Bewegungen des Benutzers müssen gemessen werden können. Das bedeutet, dass eine Form von Messschaltung notwendig ist um die Beugung der Finger interpretieren zu können. Dies könnte man durch das Messen des Beugungswinkels realisieren. Allerdings hat jeder Finger drei Gelenke, wodurch man diese auch 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. 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.

## 7.1 Eingabesubsystem

---



Die Flexsensoren (Rflex) beziehen ihre Versorgung über einen Shunt-Widerstand (Rs). 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 (Rf und Cf) ist hinter dem Ausgang des OPVs geschaltet, um mögliche Spannungsstörungen (Ripple), zusätzlich zu dem ohnehin schon sehr störungsfreien Ausgangssignal des INA129, herauszufiltern. Der zu GND geschaltete Widerstand (R1), 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

Verstärkungsgleichung laut Datenblatt:  $G = 1 + \frac{49.4k\Omega}{R_g}$   
dimensionierung shuntwiderstand noch einfügen

Gain gewählt mit 46.  $R_g = 1.1k\Omega$

Die Verstärkung wurde so gewählt, dass diese mit dem Shunt-Widerstand für den ADC optimal geeignet ist. (Simulation)

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  $25k\Omega - 125k\Omega$  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 Spannungsdifferenzen am Flexsensor für den Mikrokontroller möglichst effizient und brauchbar zu machen, ist eine Umwandlung in ein digitales Signal notwendig. Dies Funktion wird mit 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 Punkt Auslesen der Sensoren 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 Punkt (externe Anschlüsse) erläutert.

- Möglichkeiten der drahtlosen Übertragung:

Da die Flexsensorwerte drahtlos übertragen werden müssen, muss der Mikrocontroller eine Form dieser Übertragung unterstützen. Vorzüglicherweise 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**.

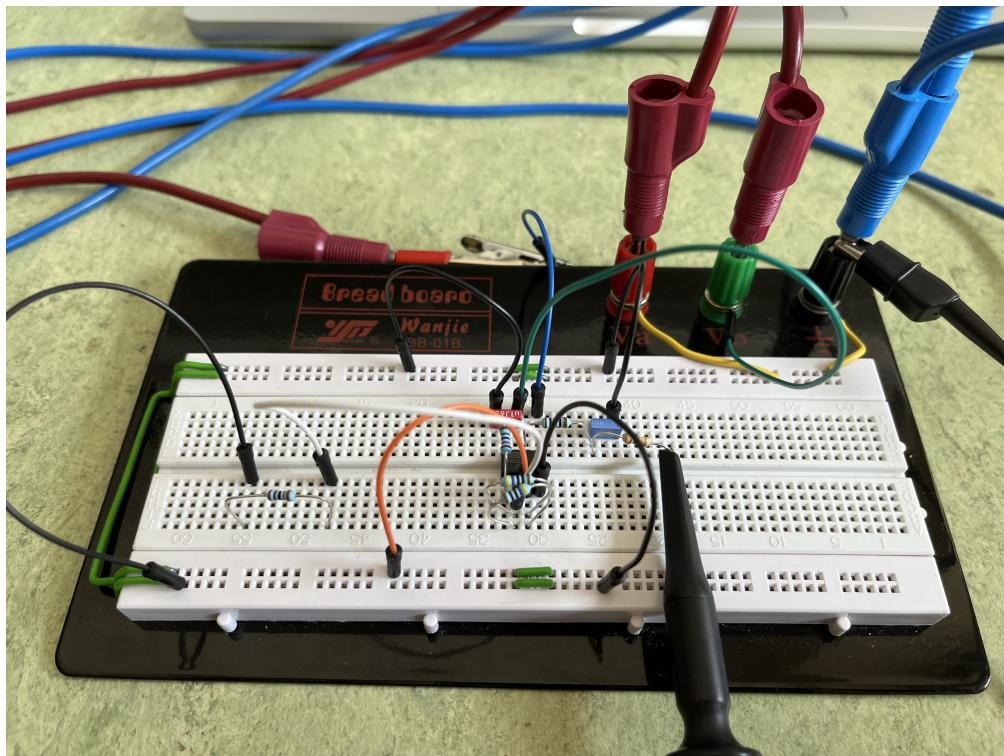
### 7.1.3 Versuchsaufbauten und Messungen **Laci**

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

#### 7.1.3.1 Messschaltung der Flexsensoren

Zunächst wurde die Schaltung aus Punkt 7.1.2.2 auf einem Steckbrett aufgebaut. Das Ziel der Messung ist es, den Widerstand bei jeder Stellung eines Flexsensoren 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. Im nächsten Punkt werden die Messergebnisse dargestellt.



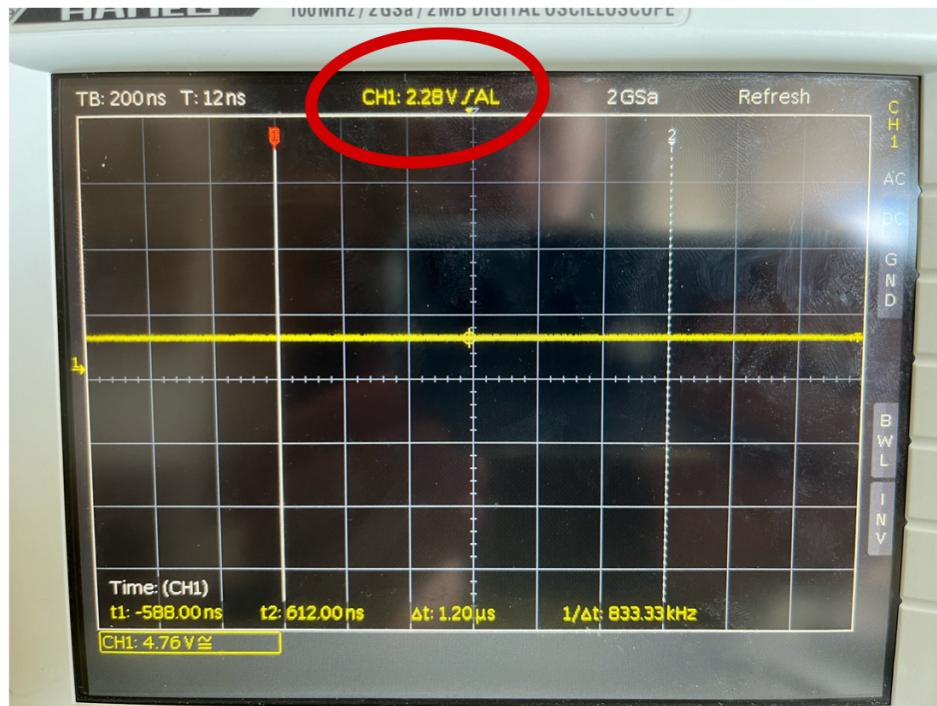


+3.3V für die Versorgung des Flexsensors (Last), simuliert mit drei normalen Widerständen.

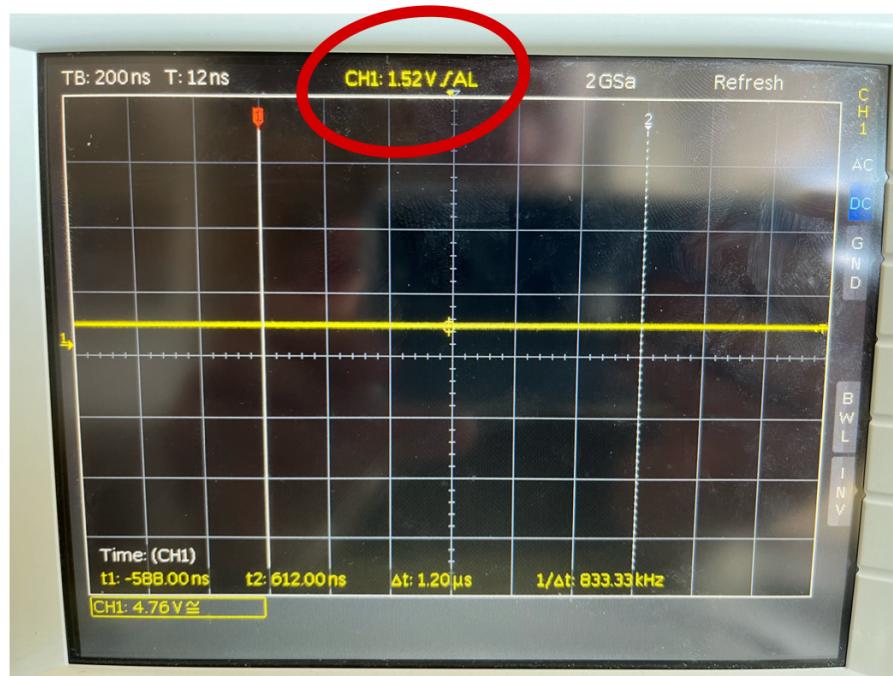
+1V als Referenzspannung für den INA-129.

+5V zur Versorgung des INA129.

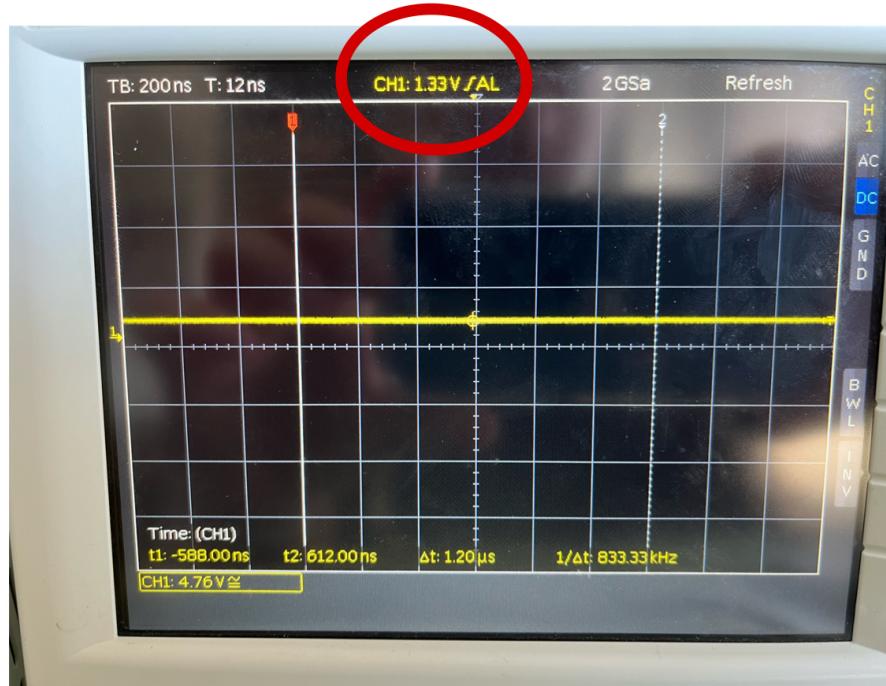
### 7.1.3.2 Messergebnisse



Das obige Oszilloskopbild zeigt die, mit einem Tastkopf, gemessene Ausgangsspannung des Operationsverstärkers, bei einem Flexsensorwiderstand von  $27\text{k}\Omega$ . Die gemessene Spannung ist rot eingekreist und beträgt 2.28V. Da bei der Simulation in LTspice mit einem Widerstand von  $25\text{k}\Omega$  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.



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.



Schlussendlich wurde noch ein Flexsensorwiderstand von  $118\text{k}\Omega$  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.

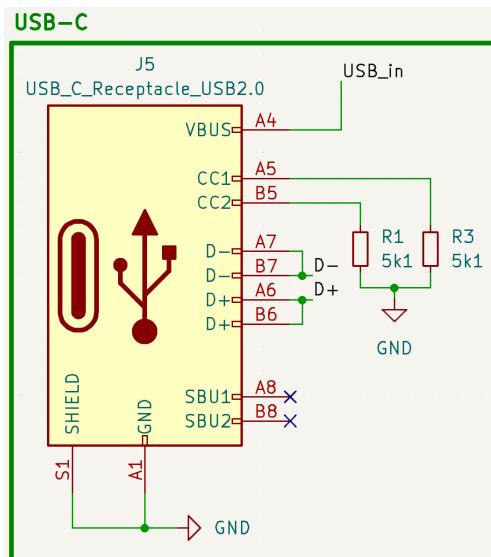
#### 7.1.4 Schaltungsdesign Laci

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

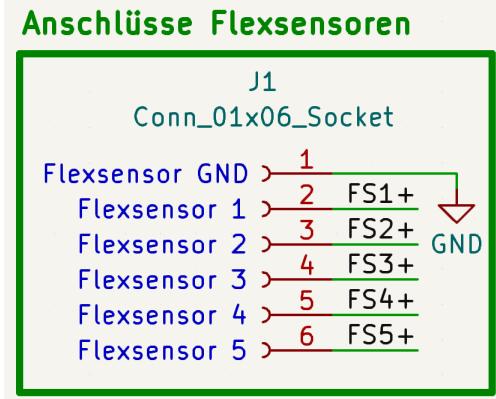
- Anschluss zur Programmierung des Mikrocontrollers:

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 Mikrocontroller lesbar zu machen, muss dieses für die UART-Kommunikation umgewandelt werden. Hierzu muss der Mikrocontroller 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 Mikrocontroller übertragen. Wichtig zu beachten ist hierbei, dass die UART-Leitungen ausgekreuzt sein müssen. Die Funktion der Anschlüsse RTS und DTR wird in Punkt [Mikrocontroller](#) erläutert.



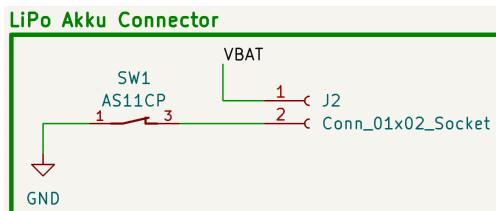
- Anschluss der Flexsensoren:

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.



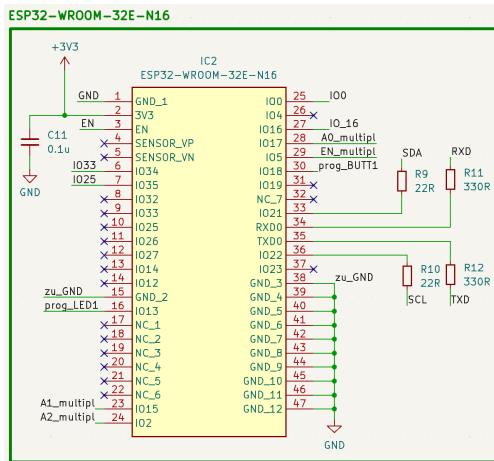
- Anschluss des Akkus:

Der Akku wird ebenfalls mit einer JST-Buchse mit der Schaltung verbunden anstatt diesen anzulöten. Da der Akku einen Versorgungs -und GND-Anschluss hat, wird eine 2-Pin-JST-Buchse verwendet.



#### 7.1.4.2 Mikrokontroller

Der ESP32-Chip wird mit +3.3V versorgt. Wichtig zu beachten ist dadurch, das 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.



### 7.1.4.3 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:

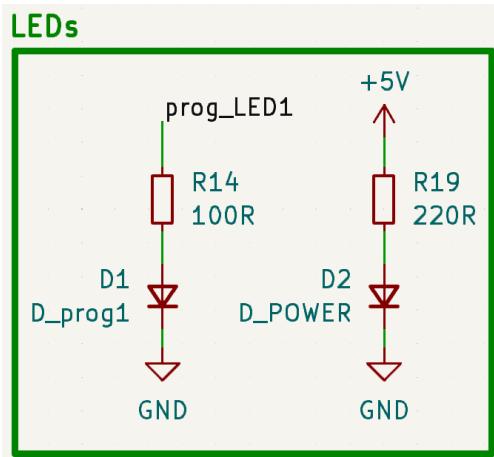
Dieser Button ist mit dem EN (Enable) Anschluss des ESP32 verbunden. Die Funktion dieses Tasters ist es, den ESP32 jederzeit zurücksetzen zu können falls dieser abstürzt oder ein anderweitiges Problem auftritt, durch das dieser nicht mehr korrekt funktioniert.

- 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.



#### 7.1.4.5 Multiplexer

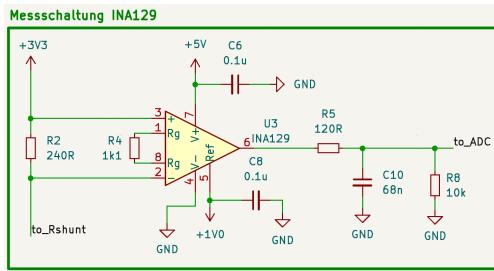
Der Multiplexer schaltet zwischen allen Flexsensoren durch und vermeidet somit die Messschaltung fünf mal bauen zu müssen. Die Verbindung zum ESP32 erfolgt über drei digitale Addresspins und einen Enable Pin. Je nachdem welche Bitkombination übermittelt wird, ändert sich die interne Schalterposition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

#### 7.1.4.6 Operationsverstärker

Für das Auslesen der Flexsensoren, wird eine Schaltung, wie in Punkt „Äuslesen der Sensoren“ 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.

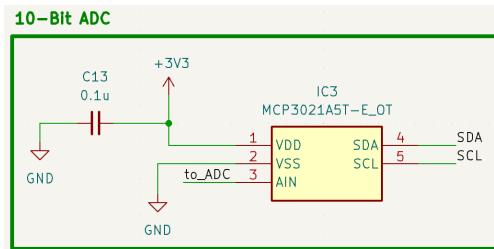
Die +1V Spannungsreferenz ist notwendig, da der ausgewählte Operationsverstärker nicht über Rail-to-Rail Technologie besitzt. Die kleinstmögliche Spannungsdifferenz am Eingang des OPV, wenn der Flexsensor seinen maximalen Widerstand von  $125\text{k}\Omega$  erreicht, beträgt nur 6.3mV. Aufgrund der 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.

## 7.1 Eingabesubsystem



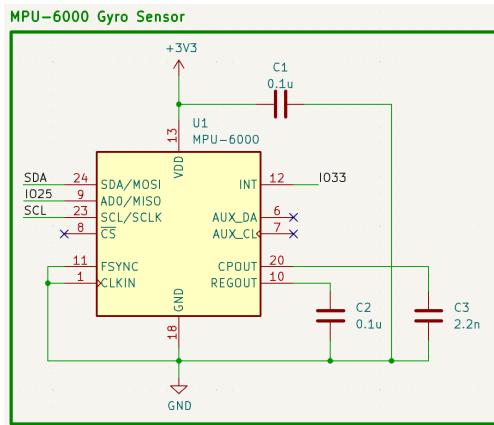
### 7.1.4.7 Analog-Digital-Wandler

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 Punkt Umwandlung der Differenzwerte in ein geeignetes Format 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.



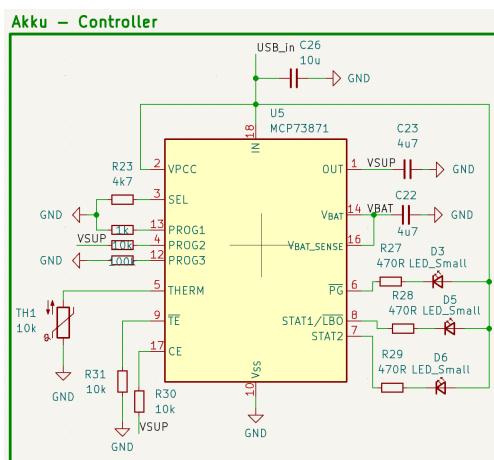
### 7.1.4.8 Gyroskop-Sensor

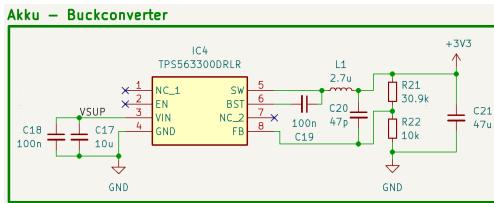
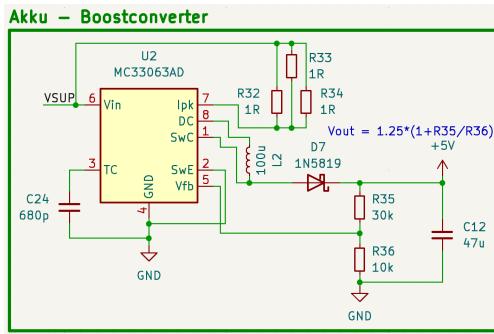
Der Gyroskopsensor ist dafür zuständig die Drehung des Handschuhs in X, Y -und Z Richtung zu erkennen. Die Versorgung basiert auf 3.3V und die Datenübertragung erneut mittels I2C-Bussystem. Der Unterschied bei diesem Chip ist allerdings, dass mehrere ADCs integriert sind, um die Positionsdaten der Achsen 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.



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.

#### 7.1.4.9 Akku Versorgung





### 7.1.5 Platinendesign Laci

## 7.2 Roboterhand

### 7.2.1 Grundlegende Voraussetzungen

Die Roboterhand ist die Ausgabe des Projekts und wird vom Eingabesubsystem, dem Handschuh, gesteuert. Jeder Finge wird von einem Servomotor bewegt. Dieser ist durch zwei dünne Schnüre mit dem korrespondierenden Finger verbunden. Die vom Handschuh kommenden Daten werden von einem Mikrokontroller ausgewertet und in ein PWM-Signal (Pulsweitenmodulation) zur Ansteuerung der Servomotoren umgewandelt. Jeder Finger hat drei Gelenke und kann daher kontrolliert gebeugt und wieder gestreckt werden. Die komplette Roboterhand wird mittels 3D-Druck gefertigt. Silikonprofile auf den Fingern versichern das Objekte nicht mehr aus dem Griff der Hand rutschen können.

Folgende Richtlinien und Normen werden von der Roboterhand erfüllt:

---

## 7.2.2 Simulationen und Versuchsaufbauten **Laci**

## 7.2.3 Überlegungen und Dimensionierung **Laci**

## 7.2.4 Schaltplandesign **Laci**

## 7.2.5 Platinendesign **Laci**

# 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 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 instabilen System sehr hohe Schwankungen der Werte andauern wahrnehmen, was zu einer durchgängigen Belastung der Servos 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 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 umgerechnet werden, nun versendet werden. Deshalb haben wir nach einer neuen Möglichkeit gesucht, die einfacher zu realisieren ist. Schlussendlich wurde es dann ESP-NOW, 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, 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 Verbindungsauflauf. 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) 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 bestenfalls 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, 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 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 Verworfene Ideen

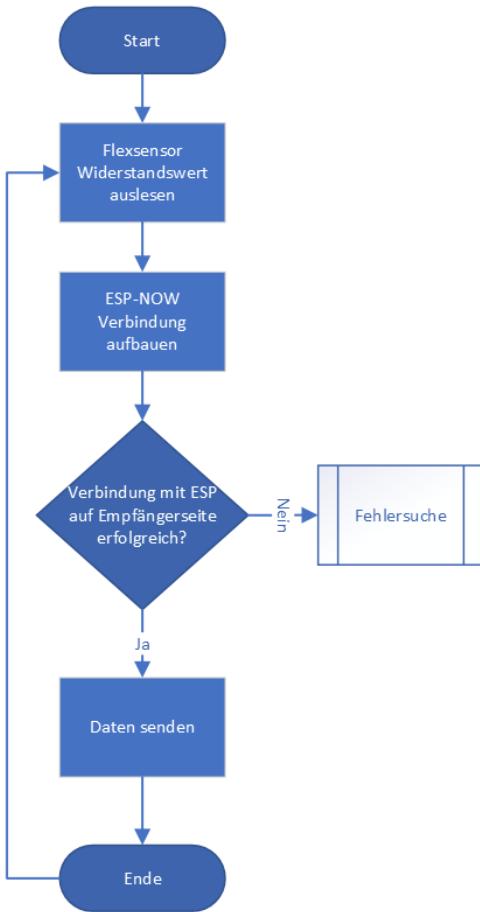
Datenkorrektur auf Senderseite Ursprünglich wurde mit dem Gedanken gespielt, dass die eingelesenen Werte bereits auf der Senderseite korrigiert werden. Allerdings haben wir uns dann überlegt, dass es aus energietechnischer Sicht sinnvoller ist, wenn man es auf der Empfängerseite macht. Die Datenkorrektur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

## 8.1.2 Realisierung und Gliederung **Fabian**

### 8.1.2.1 Realisierung



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 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.

## 8.2 Roboterhand

### 8.2.1 Grundlegende Voraussetzungen

Die Software des Roboterarmes ist dafür verantwortlich, dass die von der Handschuhplatine, via ESP32, gesendeten Daten erfolgreich empfangen und gespeichert werden. Die Software wird hierzu per UART-Schnittstelle auf den ESP32 hochgeladen. Die Daten der Flexsensoren, die man bereits erhalten hat, sollen dann verarbeitet werden. Anhand des Widerstandwertes und der jeweiligen Veränderung, wird dann der Winkel des Servos eingestellt. Eine Fehlerkorrektur soll sicherstellen, dass die Griffkraft passend eingestellt wird. Jeder Flexsensor hat unterschiedliche Widerstandswerte, es muss also über jeden Finger eine gezielte Software geschrieben werden, damit man schlussendlich durch das Zusammenspiel aller Finger die Griffkraft richtig einstellen kann. Falsche Messungen sollen bestmöglich korrigiert werden, sodass die Servos jeweils die richtigen Winkeleinstellungen erhalten. Der ESP32 stellt hierbei die Zentrale der Steuerung dar.

### 8.2.2 Konzepte und Überlegungen **Fabian**

Datenübertragung, Programmiersprache/Entwicklungsumgebung und Benutzerfreundlichkeit siehe 6.1.1!

### 8.2.2.1 Testen

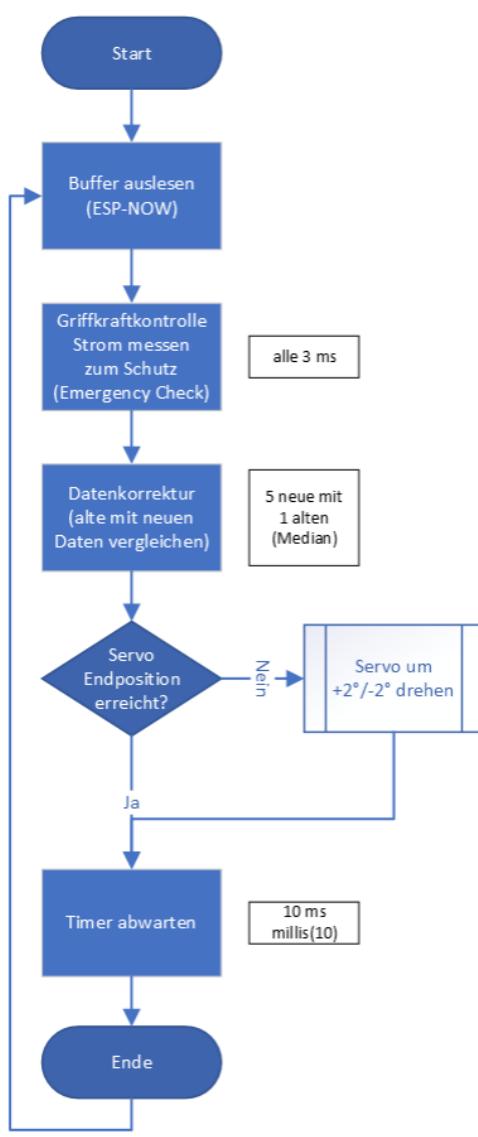
Die von der Senderplatine (Handschuh) gesendeten Daten sollen ausgelesen werden. Sie wurden nach einem bestimmten Format (Schema) gesendet. Diese Daten sollen nun ausgewertet werden. Diese Daten werden nun dazu verwendet, um die einzelnen Servos anzusteuern. Dabei wird zuerst die Servo Nummer der empfangenen Daten ausgelesen und dann der Wert. Je nachdem wie groß der Wert ist, dreht sich der Servo. Es kann dann getestet werden, ob sich der Servo tatsächlich dreht und somit, ob die Kommunikation mit dem Servo funktioniert. Wenn man die 3D-gedruckten Finger dann per Angelschnur mit den Servos verbindet, dann kann man am besten feststellen, ob die Software richtig läuft oder nicht.

### 8.2.2.2 Allgemeines Konzept

Die über ESP-NOW empfangenen Daten (Widerstandswerte samt Nummer des Flexsensors) sollen ausgelesen werden. Dadurch kann dann bestimmt werden, welcher Servo sich um wie viel drehen soll. Es muss festgestellt werden, dass die Daten über eine Datenkorrektur (Median der letzten 5 Werte) korrigiert wird. Das soll dazu beitragen, dass Fehlmessungen nicht in die Werte, die an die Servos übergeben werden, einfließen. Dadurch wird sichergestellt, dass die korrigierten Werte weniger anfällig für große Schwankungen gegenüber der möglicherweise fehlgemessenen Werten (Wackelkontakt zwischen Flexsensor und Sendeplatine) sind. Durch die Strommessung kann dann noch festgestellt werden, ob ein Finger blockiert. Dies ist dann der Fall, wenn der Strom zu lange einen gewissen festgelegten Schwellwert übersteigt. Die Servos sollen am Ende dann mit den korrigierten Werten angesteuert werden und sich ja nach Wert dann um eine bestimmten Winkel drehen, sodass der 3D-gedruckte Finger herangezogen oder entlastet wird.

## 8.2.3 Realisierung und Gliederung **Fabian**

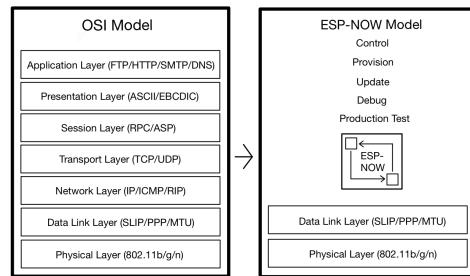
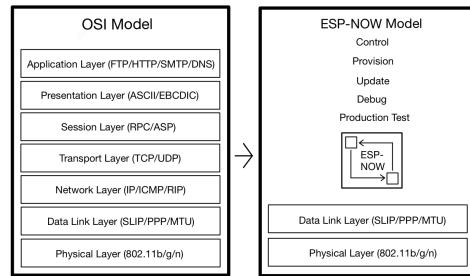
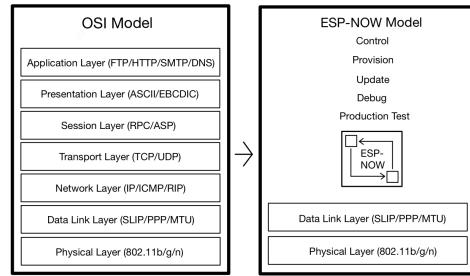
### 8.2.3.1 Realisierung



Der Code der Programmierung des ESP32 wurde in C/C++ geschrieben.

Am Anfang wird der Buffer des ESP32 ausgelesen. Die von der Senderplatine (Handschuh) empfangenen Daten werden verarbeitet. Der Strom, der eine mögliche Blockierung eines Fingers erkennen kann, muss alle 3ms gemessen werden. Wenn dieser je Finger zu hoch ist, dann muss dieser Finger vorerst auf dem momentanen Wert belassen werden oder auf den maximalen Wert gesetzt werden. Dieser beschreibt die Stellung, wenn ein Finger nicht gebeugt wurde. Da die empfangenen Daten möglicherweise eine Fehlmessung beinhalten, muss eine Datenkorrektur diese entfernen. Der Median der letzten 5 Werte wird zu diesem Zweck verwendet. Dies soll dazu führen, dass ein Wert, der sehr anders als die weiteren 4 Werte ist, nicht an die Servos gesendet wird, sondern aussortiert wird. Nachdem dann alle Werte gesendet werden, wird überprüft, ob die Endposition des jeweiligen Servos erreicht wurde. Falls nein, dann darf sich der Servo weiter zu dem gesendeten Wert hinbewegen. Falls dann die gewünschte Endposition erreicht ist oder der Servo sich in eine beliebige Richtung

dreht, dann wird noch der Timer von 10ms abgewartet und das Prozedere beginnt wieder von vorne.



## 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 Verbindungsauflbau 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.

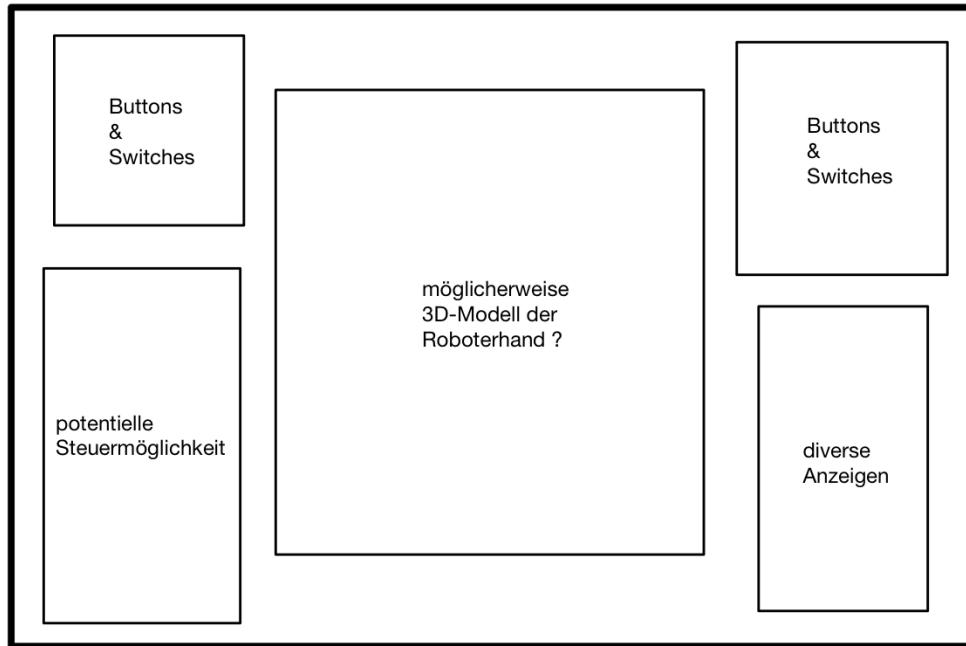
Weitere Eigenschaften von Qt sind:

- 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 grafische Interface wurde in der Beschreibungssprache QML erstellt, welche vielseitige Möglichkeiten zur individuellen Gestaltung bietet. Das Erstkonzept sieht folgendermaßen aus:



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 und Gliederung **Laci**

## 9 Tests und Messungen

### 9.1 Platzhalter für diverse Unterpunkte

## 10 Projektmanagement

### 10.1 Projektstrukturplan

### 10.2 Milestoneplan

Lastenheft fertig – 10.10.2023

Kostenkalkulationen fertig – 24.10.2023

Prototypen proof of concept – 21.11.2023

Hardwaredesign fertig – 05.12.2023

PCB-Design fertig – 09.01.2024

Softwareimplementierung für Integrationstest fertig - 30.01.2024

Integration aller Komponenten – 06.02.2024

User Interface fertiggestellt – 13.02.2024

Abnahme durch die Projektbetreuer – 12.03.2024

### **10.3 Gantt-Diagramm**

### **10.4 Meeting Struktur**

## **11 Fertigungsunterlagen**

### **11.1 Mechanik**

#### **11.1.1 händische Skizzen**

#### **11.1.2 CAD-Zeichnungen**

#### **11.1.3 3D-Modelle**

### **11.2 Hardware**

#### **11.2.1 Stromlaufpläne**

#### **11.2.2 Platinen**

#### **11.2.3 Bestückungslisten**

### **11.3 Software**

#### **11.3.1 Diagramme**

#### **11.3.2 Code**

## **12 Ergebnisse und Erkenntnisse**

## **13 Ausblick**

## **14 Verzeichnisse**

### **14.1 Quellenverzeichnisse**

### **14.2 Abbildungsverzeichnis**

### **14.3 Abkürzungsverzeichnis**

## **15 Anhang**