Diplomarbeit

31. Januar 2024

Gesamtprojekt

RoboGlove - Bionische Hand

3D-Druck, Mechanik und User-Interface-Programmierung
Amir Al-Maytah 5BHEL Betreuer: Dipl.-Ing. Christoph Diemberger
Fachlehrer: Robert Offner

Mikrokontroller-Promgrammierung, Testmanagement und Gesamtintegration
Fabian Schweitzer 5BHEL Betreuer: Dipl.-Ing. Christoph Diemberger
Fachlehrer: Robert Offner

Hardwareentwicklung, PCB-Design und Projektleitung
Ladislaus Szabo 5BHEL Betreuer: Dipl.-Ing. Christoph Diemberger
Fachlehrer: Robert Offner

Ausgeführt im Schuljahr 2023/24

Eidesstattliche 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.

Inhaltsverzeichnis

1 Einleitung

1.1 Kurzfassung

Aufgabenstellung: Im Rahmen eines Diplomprojekts soll eine Roboterhand mittels eines Handschuhs gesteuert werden. Dazu müssen verschiedene Sensoren auf Genauigkeit getestet und aanscließend 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.
Realisierung: 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.
Ergebnis: 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.

1.2 Abstract

Task: 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.
Realization: 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.
Result: 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.

1.3 Ausgangslage

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.

1.4 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)

1.5 Allgemeine Zielsetzung

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.

1.6 Terminplan (Milestones)

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

1.7 Geplantes Ergebnis der Prüfungskandidaten

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.

1.8 Entwicklungsteam

Folgende Schüler des TGMs haben, das in Punkt 1.1 zusammengefasste, 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

2 Anforderungen der einzelnen Projektteile

2.1 Mechanik

2.1.1 Handschuh Amir

Folgende machanische Anforderungen muss der Handschuh nach Abschluss des Projekts erfüllen:

2.1.2 Roboterhand Amir

Folgende mechanische Anforderungen muss der Handschuh nach Abschluss des Projekts erfüllen:

- Die Roboterhand soll mittels 3D-Druck gefertigt werden.
- Die Servomotoren der Elektronik werden mithilfe von Schnüren mit den jeweiligen Fingern verbunden.
- Die Finger sollen sich sowohl zum Handballen als auch vom Handballen weg kontrolliert bewegen können.
- Die Finger sollen sich zitterfrei bewegen können.

2.2 Hardware

2.2.1 Handschuh Laci

Die Hardware des Handschuhs muss einige Kriterien erfüllen, um schlussendlich voll funktionfähig in das Gesamtprojekt 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, darf die entgültige Platine auf dem Handschuhrücken nicht größer als 3cm x 3cm 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 ab einer Temperatur von ca. 60C° unangenehm für den Endbenutzer wird. Die Versorgung der Schaltung sollte möglichst über einen kleinen und portablen Akku geschehen, um dem Benutzer das bestmögliche Erlebnis zu bereiten. Wenn diese schaltungsbezogenen Anforderungen bedacht sind, stellt sich nun die Frage wie und womit die Fingerbewegungen des Anwenders erfasst werden sollen. Hier setzen wir auf Flexsensoren, wobei optional noch ein Gyro-Sensor, also ein Mikrochip mit einem eingebauten Gyroskop, zum Einsatz kommen kann. Diese Sensoren müssen durch einen Mikrokontroller gesteuert beziehungsweise ausgelesen werden. Für die Verwendung des Mikrokontrollers muss eine Art von Datendigitalisierung in der Schaltung vorhanden sein, die zumindest 30 verschiedene Sensorpositionen erkennen kann. Über einen USB Anschluss soll der Mirkokontroller programmierbar sein und die Schaltung auch für Test -und Wartungszwecke versorgt werden können. Das Maximalgewicht darf 500g nicht übersteigen.

2.2.2 Roboterhand Laci

Folgende elektronische Anforderungen muss die Roboterhand nach Abschluss des Projekts erfüllen:

- Die Bewegung der Finger muss mit Sensoren erfasst werden.
- Die Sensordaten müssen mit einem Mikrokontroller ausgewertet werden können.

- Der zur Datenerfassung gewählte Mikrokontroller muss die I2C-Kommunikation unterstützen, die Möglichkeit für eine kabellose Kommunikation aufweisen, analoge und digitale Eingänge haben und programmierbar sein.
- Es muss möglich sein, mindestens 30 verschiedene Sensorwerte, von minimaler bis maximaler Fingerbeugung, zu erfassen.
- Der RoboGlove muss die Möglichkeit zur Versorgung mit einem Akku oder einer Batterie und einem Netzteil aufweisen.
- Jeder Finger der „Bionic Hand“ soll durch einen eigenen Motor gesteuert werden.
- Jeder dieser Motoren muss individuell regelbar sein, um die Griffkraft der Hand zu kontrollieren.
- Jeder Finger soll sich zum Handballen beugen und wieder strecken lassen.
- Diese Bewegungen sollen störungsfrei und ohne zittern erfolgen.

2.3 Software

2.3.1 Handschuh Fabian

Folgende Software Anforderungen muss der Handschuh nach Abschluss des Projekts erfüllen:

- Der Widerstand jedes Flexsensors, der an jedem Handschuh jeweils über jedem Finger angebracht ist, soll einzeln ausgelesen werden können. Die Software soll damit feststellen, wie sehr der Flexsensor gebogen ist.
- Ein Multiplexer soll angesteuert werden, damit der Wert des jeweiligen Flexsensors ausgelesen werden kann. Ein ADC, der die Spannung misst, wandelt die analogen Werte in digitale um. Dieser muss ebenso zur Messung der einzelnen Werte ausgelesen werden.
- Diese Daten sollen erfolgreich eingelesen werden und direkt an die Empfängerplatine über den Funkstandard ESP-NOW gesendet werden.
- Die von der Handschuhplatine gesendeten Daten sollen erfolgreich empfangen und gespeichert werden. Eine Bestätigung soll im Serial Monitor gesehen werden können.
- Die Software muss bei jedem Finger etwas anders agieren, da die Widerstandswerte der verschiedenen Flexsensoren stark variieren. Es soll bei jedem Finger eine gleich gute Steuerung der Roboterhand möglich sein.
- Es muss immer wieder (alle 10ms) der Strom gemessen werden. Mit einem Timing Interrupt muss also zwischen dem Bewegen der Finger immer wieder dieser gemessen werden.
- Es sollen unterschiedliche Modi erstellt werden, je nachdem, was mit der Roboterhand gehalten werden soll. Die Griffkraft soll je nach Objekt variieren.

2.3.2 Roboterhand Fabian

Folgende Software Anforderungen muss der Handschuh nach Abschluss des Projekts erfüllen:

- Die von der Senderplatine, über ESP-NOW, gesendeten Daten sollen erfolgreich empfangen und verarbeitet werden.

- Eine Fehlerkorrektur sorgt dafür, dass die empfangenen Daten korrigiert werden können, um starke Werteabweichungen, die durch Fehlmessungen entstehen, zu reduzieren.
- Die Servos, die zur Steuerung der einzelnen Finger der Roboterhand nötig sind, sollen über die Software einzeln angesprochen werden können. Der jeweils richtige Wert (der Biegung) des jeweiligen Flexsensors soll dann die idente Bewegung des Fingers verursachen.
- Die empfangenen Werte sollen ebenso an das User Interface per UART übertragen werden, um sie dort grafisch darzustellen.
- Die Werte, die im User Interface eingestellt werden, sollen von der Empfängerplatine verarbeitet werden. Die Finger sollen sich dann exakt so viel bewegen, wie es im User Interface eingestellt wird.

2.3.3 User Interface Laci

3 Detailierte Beschreibung der Projektteile

3.1 Handschuh (Eingabe) Fabian

Der Handschuh ermöglicht dem Benutzer die Roboterhand zu steuern. Dies geschieht durch Flexsensoren, die an den Fingern des Handschuhs entweder durch annähen, ankleben, oder einer 3D gedruckten Motage 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.

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.

Folgende Richtlinien und Normen werden vom Handschuh erfüllt:

Die Software des Roboterhandschuhs ist dafür verantwortlich die sich verändernden Widerstandswerte der Flexsensoren jeweils einzeln einzulesen. Über die UART-Schnittstelle wird die Software auf den ESP32 hochgeladen. Dabei werden direkt aus dem in der Schaltung verbauten ADC-Werte, über

die I2C-Schnittstelle, ausgelesen. Man erhält bereits digitale Werte, die der ESP32 dann direkt in einer festgelegten Reihenfolge an die Empfängerplatine, bei der Roboterhand, per ESP-NOW Übertragung senden kann.

3.2 Roboterhand (Ausgabe) Amir

Die Roboterhand ist die Ausgabe des Projekts und wird vom Roboterarm 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:

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.

3.3 User Interface Laci

4 Mechanische Entwicklung

4.1 Handschuh Amir

4.2 Roboterhand Amir

5 Hardwareentwicklung

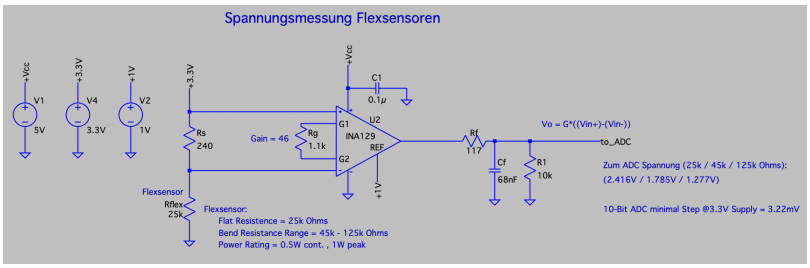
5.1 Handschuh

5.1.1 Überlegungen, Simulationen und Berechnungen Laci

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.

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.



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 korrerkte 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 ist dies +1V. Würde diese Refernzspannung 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 inder Simulation bestimmt.

Ein Tiefpassfilter (Rf und Cf) ist hinter den Ausgang des OPVs geschaltet, um mögliche Spannungsstörungen (Ripple), zusätzlich zu dem ohnehin schon sehr störungsarmen Ausgangssignal des INA129, herauszufiltern. Der zu GND geschaltete Widerstand (R1), entlastet den Eingang des folgenden ADCs. Der Tiefpassfilter wurde folgendermaßen dimensioniert.

Berechnung des Tiefpassfilters:

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

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

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

Berechnung der OPV Verstärkung und Dimensionierung des Shuntwiderstands:

Verstärkungsgleichung laut Datenblatt: $G = 1 + \frac{49.4k\Omega}{R_g}$
diemensionierung 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 240Ω gewählt, da dieser Widerstand bei sowohl voller, als auch geringer Biegung der Flexsensoren, eine gute Spannungsdifferenz für die Verstärkung mit dem OPV liefert.

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, Auflösungund Aussteuerbereich:

  $Aussteuerbereich = 0 - 3.3V$

bei 10Bit ADC: $LSB = 3.22mV$

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

$ADCAusgangsstufen = \frac{2.3V}{3.22mV} = 713$

Der reale Ausgangspegel des OPVs liegt wie bei der Simulation in Punkt Auslesen der Sensoren ermittelt zwischen 1.277V bis 2.416V. Das beudeutet, 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.

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.

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 Kompatiblität mit der restlichen Schaltung zu ermöglichen.

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

Mikrokontroller:
Bei der Auswahl des Mikrokontrollers 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 vonn dem verwendeten Mikrokontroller 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 Mikrokontroller 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 die wir bei der Schaltung eunheitlich auf das I2C Bussystem setzen, muss zumindest dieses von dem gewählten Mikrokontroller unterstützzt 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 Mikrokontroller 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 Mikrokontrollers 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.

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 Mikrokontrollers verwendet wird. Dazu ist allerdings eine relativ komplitzierte 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.

5.1.2 Versuchsaufbauten und Messungen Laci

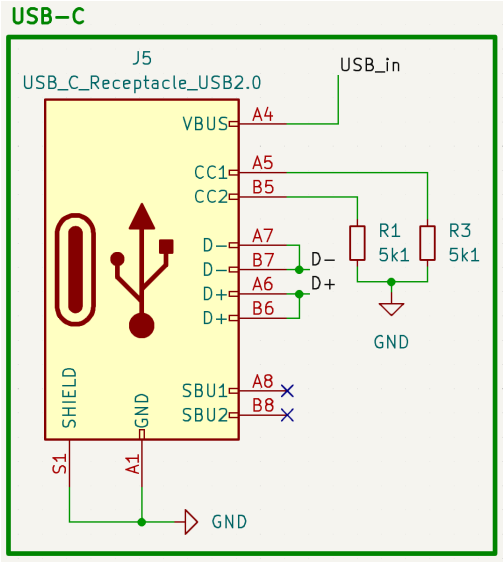
5.1.3 Schaltungsdesign Laci

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

Externe Anschlüsse:

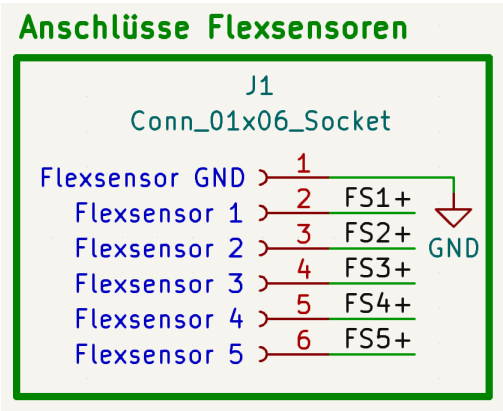
- Anschluss zur Programmierung des Mikrokontrollers:

  Für die Programmierung des Mikrokontrollers 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 Punkt Mikrokontroller erläutert.



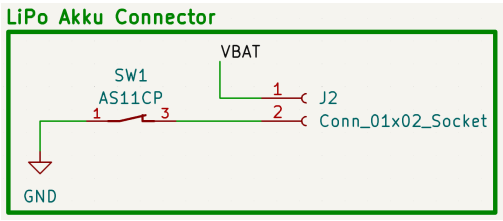
- Anschluss der Flexsensoren:

  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.

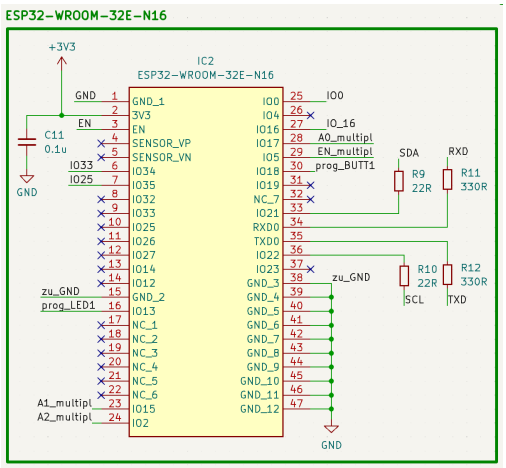


- Anschluss des Akkus:

  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.



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.



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.

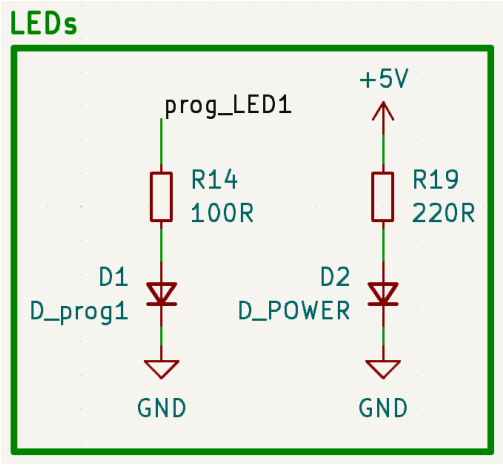
- Reset Button:

  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 abstürtzt oder ein anderweitiges Problem auftritt, durch das dieser nicht mehr korrekt funktioniert.

- Progrmmable Button:

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

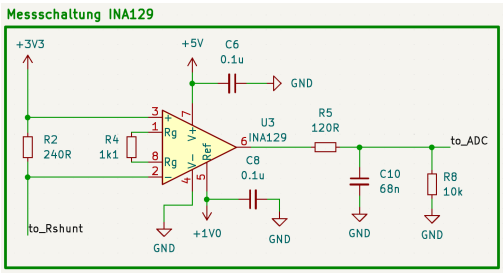
Status LEDs:
Die beiden Leuchtdioden sind als Statusanzeige gedacht. Eine POWER LED, die immer leuchtet wenn die Schaltung mit +5V versorgt wird. Die Funktion der anderen LED ist, sowie bei einem Button, frei wählbar und ist deswegen mit Pin IO13 des ESP32 verbunden.



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.

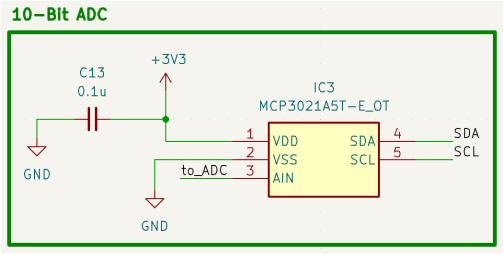
Operationsverstärker:
Für das Auslesen der Flexsensoren, wird eine Schaltung, wie in Punkt Äuslesen der Sensoren" beschrieben, benötigt. Da am Shuntwiderstand nur sehr wenig Spannungsabfall auftritt und daher die Spannungsdifferenz zwischen positivem und negativem Verstärkereingang sehr klein ist, muss 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 $125k\Omega$ 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.



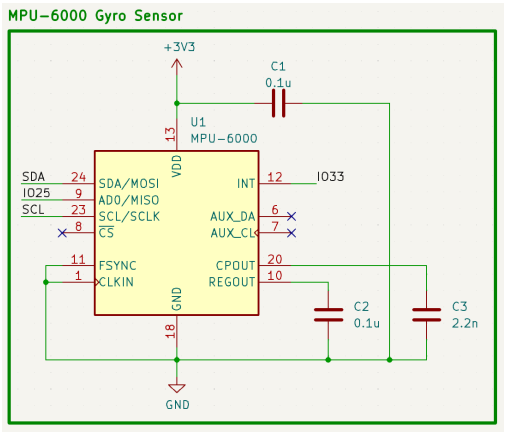
Analog-Digital-Wandler:

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 Diferenzwerte 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.



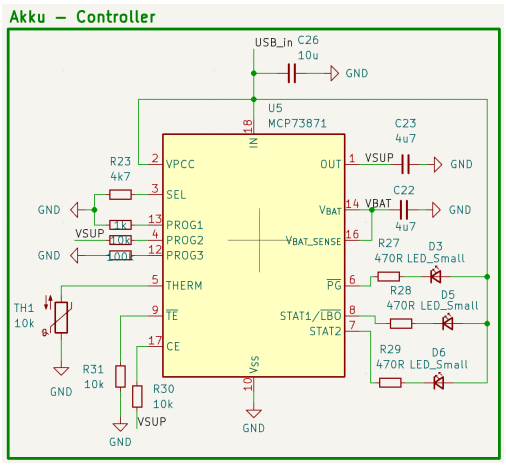
Gyroskop-Sensor:

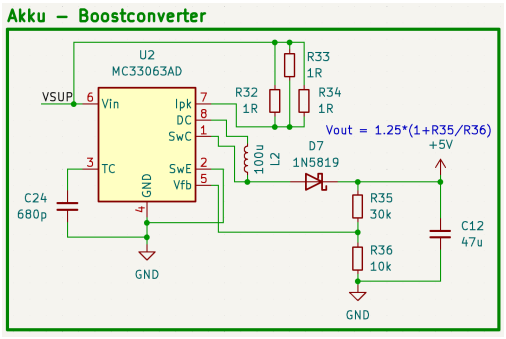
Der Gyroskopsensor ist dafür zuständig die Drehung des Handschuhs in X, Y -und Z Richtung warzunehmen. Die Versorgung basiert auf 3.3V und die Datenübertragung erneut mittels I2C-Bussystem. Der Unterschied bei diesem Chip ist allerdings, dass mehrere ADCs integriert sind, um die Positionsdaten der Achsen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

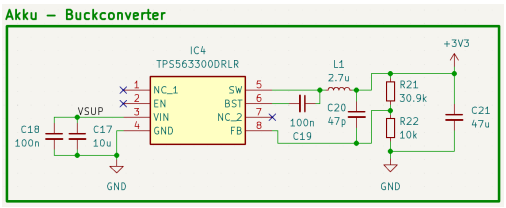


Am Pin CLKIN hätte man die Möglichkeit einen externen Referenztaktgeber anzuschließen. Bei unserer Anwendung ist der integrierte Taktgeber allerdings völlig ausreichend, weswegen der Anschluss mit GND verbunden und damit deaktiviert wurde. Der CS Pin wird nur für die SPI Kommunikation benötigt, weshalb 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.

Akku Versorgung:







5.1.4 Platinendesign Laci

5.2 Roboterhand

5.2.1 Simulationen und Versuchsaufbauten Laci

5.2.2 Überlegungen und Dimensionierung Laci

5.2.3 Schaltplandesign Laci

5.2.4 Platinendesign Laci

6 Softwareentwicklung

6.1 Handschuh

6.1.1 Konzepte und Überlegungen Fabian

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 unstabilen 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 funkt 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 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.

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.

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.

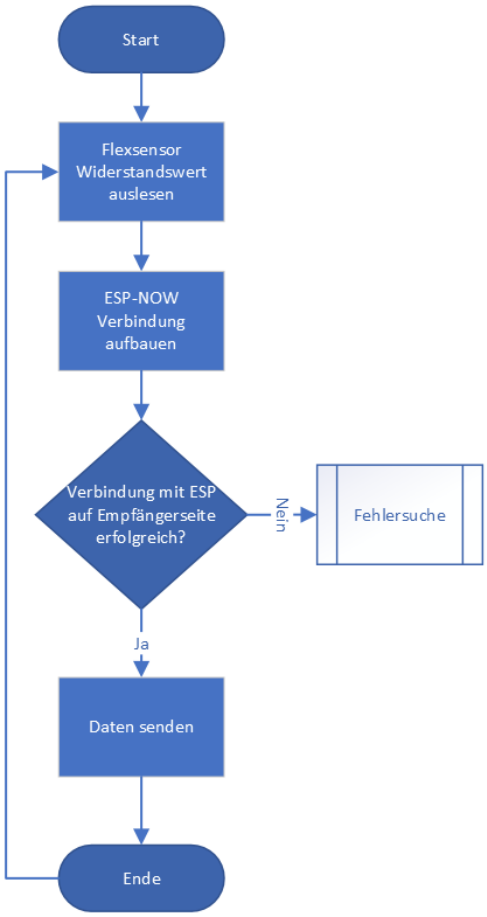
Testen:
Jedes Programm muss ausführlich getestet werden. Die Tests bei der Senderplatine (Handschuh) beschäftigen sich vor allem mit dem Auslesen der Widerstandswerte der Flexsensoren. Es soll überprüft werden, ob sich die Werte in dem von dem Benutzer freiwillig ausgesuchten Bereich der map Funktion 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 gemapt 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.

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: "$\$s : n : angepasster Wert$" (n... Multiplexer 0 – 4; angepasster Wert... gemapter 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.

6.1.2 Realisierung und Gliederung Fabian

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.

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 "$\$s : n : angepasstesWert$" 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.

6.2 Roboterhand

6.2.1 Konzepte und Überlegungen Fabian

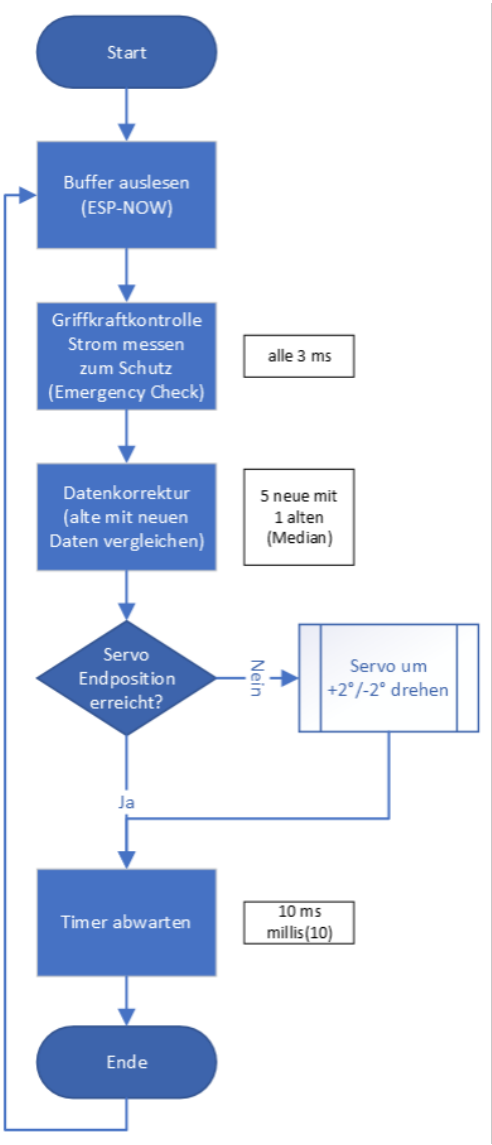
Datenübertragung, Programmiersprache/Entwicklungsumgebung und Benutzerfreundlichkeit siehe 6.1.1!

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.

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 einen bestimmten Winkel drehen, sodass der 3D-gedruckte Finger herangezogen oder entlastet wird.

6.2.2 Realisierung und Gliederung Fabian

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.

6.3 User Interface

6.3.1 Entwicklungsumgebung Laci

6.3.2 Konzepte und Überlegungen Laci

6.3.3 Realisierung und Gliederung Laci

7 Abschluss und Zusammenfassung

8 Anhang

9 Quellen -und Literaturverzeichnis

10 Abbildungsverzeichnis