Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

Bachelorarbeit

Prototyp und Echtzeit-Software-Architektur für einen IMU-Datenhandschuh mit haptischem Feedback

Prüfer: Dr. Norman Hendrich

Betreuer: Dr. Andreas Mäder

Vorgelegt von:
Jan Metzing, 6256221
Depenkamp 29, 22549 Hamburg
0metzing@informatik.uni-hamburg.de
B.Sc. Informatik

Abgabe:
Hamburg, 14.11.2024

Abstract

This bachelor's thesis focuses on the development and evaluation of a prototype and real-time software architecture for an IMU-based data glove with haptic feedback. The primary goal was to design an open source system capable of accurately tracking hand and finger movements using Inertial Measurement Units (IMUs) while integrating haptic feedback to enhance the user experience.

The system combines IMUs for precise movement detection with vibration motors to provide tactile feedback to the user. Building on the foundational work of Can Bagdas, this thesis further develops the prototype by improving data accuracy, integrating visualization functions into the Robot Operating System (ROS), and optimizing the system's response time.

The evaluation of the data quality revealed that the system's performance is influenced by factors such as data acquisition frequency, sensor drift, and the interaction between the IMUs and the vibration motors. These challenges were addressed through calibration methods like Ferraris calibration and the use of the Madgwick filter, which minimized errors caused by sensor drift and vibration effects. Additionally, the system's movement data was visualized in real-time using ROS, providing a comprehensive view of hand and finger positions.

The results showed that the system is capable of accurately tracking hand movements, with potential for further improvements in fine motor accuracy and long-term performance.

Inhaltsverzeichnis

1 Einleitung

Die fortschreitende Entwicklung in den Bereichen Robotik, Virtual Reality (VR) und Augmented Reality (AR) hat den Bedarf an präzisen und flexiblen Eingabegeräten deutlich erhöht. Insbesondere in der Mensch-Maschine-Interaktion, bei der Benutzer durch ihre Bewegungen oder Gesten mit virtuellen oder physischen Systemen interagieren, werden hochgradig sensitive und realistische Eingabegeräte benötigt. Ein zentrales Element dieser Interaktion sind Hand- und Fingerbewegungen, da diese in vielen Anwendungsszenarien – von der Steuerung von Robotern bis hin zu immersiven VR-Erfahrungen – eine entscheidende Rolle spielen.

Traditionelle Eingabegeräte wie Tastaturen, Mäuse oder Joysticks bieten nur eingeschränkte Möglichkeiten, um die komplexen und nuancierten Bewegungen der menschlichen Hand präzise zu erfassen. Hier setzt die Entwicklung von Datenhandschuhen an, die eine direkte und intuitive Erfassung von Handbewegungen ermöglichen. Diese Geräte können in verschiedenen Anwendungen eingesetzt werden, um eine realistischere und benutzerfreundlichere Steuerung zu bieten.

Im Fokus steht die Weiterentwicklung eines kostengünstigen und effektiven Datenhandschuhs, der mithilfe von Inertialsensoren (IMUs) die Bewegungen der Hand und der Finger in Echtzeit erfasst. Besonderer Wert wird auf die Präzision und Zuverlässigkeit der Bewegungserfassung sowie auf die Integration von haptischem Feedback gelegt, um eine möglichst genaue und benutzerfreundliche Steuerung zu ermöglichen.

Die Arbeit befasst sich mit der Auswahl und Implementierung geeigneter Hardwarekomponenten, der Entwicklung der Firmware zur Verarbeitung der Sensordaten sowie der Integration des Datenhandschuhs in das Robot Operating System (ROS). Ein besonderes Augenmerk liegt auf der Kalibrierung der Sensoren und der Optimierung der Datenübertragung, um eine präzise Erfassung und Darstellung der Handbewegungen zu gewährleisten. Zur praktischen Anwendung der Handbewegungsdaten werden die berechneten Orientierungen in RViz visualisiert.

Zunächst wird die theoretische Grundlage zu Inertialsensoren und deren Anwendung in der Handbewegungserfassung erläutert. Anschließend folgt eine detaillierte Beschreibung des entwickelten Systems, beginnend bei der Hardware-Entwicklung bis hin zur Software-Integration. Abschließend werden die Ergebnisse der Tests präsentiert und die gewonnenen Erkenntnisse diskutiert, um den Erfolg des entwickelten Systems zu evaluieren.

2 Problemstellung

Ein zentrales Problem bei der Entwicklung von Datenhandschuhen ist die Präzision der eingesetzten Sensoren. Insbesondere Gyroskop- und Beschleunigungssensoren sind anfällig für Driftphänomene, die mit der Zeit und unter variierenden Betriebsbedingungen zu Ungenauigkeiten führen können. Diese Ungenauigkeiten beeinträchtigen nicht nur die Qualität der erfassten Daten, sondern führen auch zu fehlerhaften Rückmeldungen, die die Benutzererfahrung negativ beeinflussen.

Ein weiteres Problem stellt die unzureichende Kalibrierung und Driftkompensation bestehender Systeme dar, wodurch Nutzer langfristig mit ungenauen Feedbacks konfrontiert werden. Insbesondere die Berechnung relativer Winkel zwischen den verschiedenen IMUs, die in einem Datenhandschuh verwendet werden, ist eine Herausforderung. Diese Berechnung setzt voraus, dass die Sensordaten korrekt ausgerichtet sind und Fehler durch die fehlende Nutzung von Magnetometern kompensiert werden. Ohne Magnetometer ist die Bestimmung des Gierwinkels undefiniert und führt zu Inkonsistenzen in den Winkeln zwischen den Sensoren, was eine präzise Finger- und Handbewegungserfassung erschwert.

Darüber hinaus kann das haptische Feedback, das durch Vibrationsmotoren erzeugt wird, die Funktionalität der IMUs beeinträchtigen. Die Vibrationen können die Sensordaten verfälschen und somit die Genauigkeit der Bewegungsmessungen weiter verringern.

Trotz der vielversprechenden Fortschritte in der Technologie von Datenhandschuhen bestehen weiterhin Herausforderungen, die die Präzision, Kalibrierung, Integration und Benutzererfahrung betreffen. Insbesondere die Berechnung und Korrektur der relativen Winkel zwischen den Sensoren und die Driftkompensation für eine langfristig stabile Leistung stellen zentrale Punkte dar. In dieser Arbeit werden diese Probleme analysiert und Lösungen entwickelt, die zu einem robusteren und benutzerfreundlicheren System führen sollen.

3 Theoretische Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen behandelt, die für das Verständnis und die Umsetzung des Projekts von entscheidender Bedeutung sind. Zunächst wird ein Überblick über die Funktionsweise und die Anwendung von Inertial Measurement Units (IMUs) gegeben, die eine zentrale Rolle bei der Erfassung von Bewegungs- und Lagedaten spielen. Es wird das Konzept des haptischen Feedbacks erläutert, das eine wesentliche Komponente der Benutzerinteraktion darstellt. Es folgt eine Erklärung der Pulsweitenmodulationssteuerung (PWM-Steuerung) von Wechselstrommotoren. Das Robot Operating System (ROS) wird als Software-Framework zur Steuerung und Kommunikation von Sensoren und Aktuatoren vorgestellt, gefolgt von einer detaillierten Beschreibung des Madgwick-Filters, einem Algorithmus zur präzisen Bestimmung der Orientierung anhand von IMU-Daten. Die Ferraris-Kalibrierung wird als Methode zur Verbesserung der Genauigkeit von IMUs behandelt, bevor abschließend die Berechnung relativer Winkel zwischen zwei IMUs erläutert wird, um Fingerbewegungen in Datenhandschuhen präzise zu erfassen. Dieses Kapitel bildet somit die theoretische Grundlage für die folgenden praktischen Implementierungen und Experimente.

3.1 Einführung in Inertial Measurement Units (IMUs)

Inertial Measurement Units (IMUs) sind wesentliche Komponenten der modernen Sensortechnik und spielen eine zentrale Rolle bei der Erfassung von Bewegungs- und Lagedaten. IMUs kombinieren typischerweise mehrere Sensoren, darunter Beschleunigungssensoren, Gyroskope und manchmal Magnetometer [1]. Beschleunigungssensoren innerhalb der IMU messen lineare Beschleunigungen entlang der X-, Y- und Z-Achse und liefern so wichtige Informationen über die Richtung und Stärke von Bewegungen. Gyroskope ergänzen diese Daten durch die Erfassung der Winkelgeschwindigkeit um die drei Raumachsen, was eine präzise Bestimmung der Orientierung eines Objekts ermöglicht. Magnetometer können zur Korrektur der Orientierung verwendet werden, indem sie das Erdmagnetfeld messen und als Referenz nutzen. Dies ist besonders wichtig, um die Drift der Gyroskope zu korrigieren und eine langfristige Stabilität der Orientierungsdaten zu gewährleisten [1]. Datenhandschuhe mit IMUs bieten folgende Vorteile:

- **Genauigkeit**: Die Kombination von Beschleunigungssensoren, Gyroskopen und Magnetometern in einer IMU ermöglicht eine präzise Erfassung der Bewegungen und Orientierungen der Finger und Hände. Dies ist entscheidend für Anwendungen in der virtuellen Realität und Robotik, wo eine exakte Bewegungsübertragung erforderlich ist [1].
- **Echtzeitverarbeitung**: IMUs können Bewegungsdaten in Echtzeit erfassen und verarbeiten, was eine unmittelbare Rückmeldung und Interaktion ermöglicht. Dies ist besonders wichtig für immersive Anwendungen, bei denen Verzögerungen die Benutzererfahrung negativ beeinflussen könnten [1].
- **Kompaktheit**: IMUs sind kompakt und leicht, was ihre Integration in tragbare Geräte wie Datenhandschuhe erleichtert. Ihre geringe Größe und Gewicht tragen dazu bei, dass die Handschuhe komfortabel und unauffällig sind, ohne die Bewegungsfreiheit des Benutzers zu stark zu beeinträchtigen [1].

IMUs sind aufgrund dieser Eigenschaften hervorragend für die Integration in Datenhandschuhe geeignet. Ihre Verwendung ermöglicht eine detaillierte und genaue Erfassung von Hand- und Fingerbewegungen, was sie zu einem unverzichtbaren Werkzeug in vielen Anwendungsbereichen wie der virtuellen Realität, der Robotik und der medizinischen Rehabilitation macht [1].

3.2 Haptisches Feedback

Haptisches Feedback bezeichnet Rückmeldungen eines Systems, welche der Nutzer über seinen Tastsinn wahrnimmt.

Die haptische Wahrnehmung umfasst das aktive Erfassen von Eigenschaften wie Größe, Kontur, Textur, Temperatur und Gewicht eines Objekts. Diese Wahrnehmung wird durch Mechanorezeptoren in der Haut und in tieferliegenden Geweben ermöglicht. Grunwald und Beyer (2001) betonen, dass die haptische Wahrnehmung für die Planung, Steuerung und Ausführung alltäglicher Handlungen von grundlegender Bedeutung ist. Mechanorezeptoren sind entscheidend für diese sensorischen Rückmeldungen, da sie Druck, Vibration und Dehnung wahrnehmen, welche dann vom Nervensystem in fühlbare Empfindungen umgewandelt werden [2].

Vibrationsmotoren sind ein zentraler Bestandteil haptischer Systeme, da sie mechanische Rückmeldungen erzeugen. Diese Motoren können in unterschiedlichen Intensitäten vibrieren, um verschiedene haptische Effekte zu erzeugen. Ein bekanntes Anwendungsbeispiel ist die Verwendung in Touchscreens von Smartphones und Tablets, wo sie beim Tippen oder Berühren Rückmeldungen geben, die das Gefühl mechanischer Tasten simulieren.

Durch die Fähigkeit, komplexe haptische Rückmeldungen zu erzeugen, tragen Vibrationsmotoren dazu bei, die Benutzererfahrung zu verbessern und realitätsnahe Interaktionen zu ermöglichen.

3.3 PWM-Steuerung von Wechselstrommotoren

Wechselstrommotoren können mit Gleichstrom betrieben werden, indem der Strom nicht wie bei klassischen Wechselstrommotoren periodisch seine Polarität ändert, sondern in einer bestimmten Frequenz ein- und ausgeschaltet wird. Dieser Prozess, bekannt als Pulsweitenmodulation (PWM), ermöglicht auch die Steuerung der Motorleistung.

Bei der Steuerung von Motoren mittels PWM hat die Frequenz des ein- und ausgeschalteten Stroms einen entscheidenden Einfluss auf die entstehenden Vibrationen. Die stärksten Vibrationen treten bei der resonanten Frequenz des Motors auf, also bei dem Punkt, an dem der Motor die maximale Vibrationsstärke erreicht. Abweichungen von dieser resonanten Frequenz führen zu einer Reduktion der erzeugten Vibrationen. Auf diese Weise lässt sich ein Wechselstrommotor mit Gleichstrom betreiben und gleichzeitig die Intensität der Vibrationen regulieren.

Wenn ein Motor zum Beispiel normalerweise mit 235 Hz Wechselstrom betrieben wird, ist die stärkste Vibration auch mit PWM bei 235 Hz zu erwarten. Abweichende Frequenzen führen zu einer Verringerung der Vibrationstärke (siehe Abbildung 1).

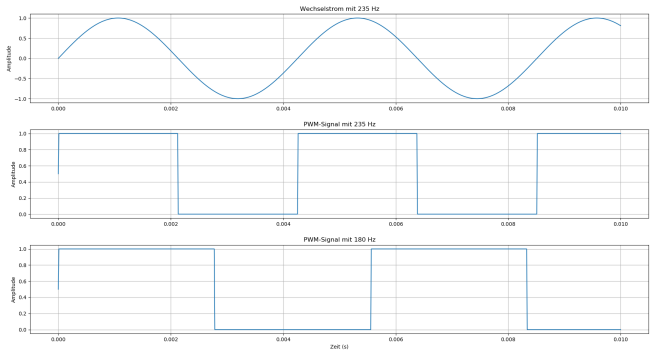

Abbildung 1: Wechselstrom- und PWM-Signale bei unterschiedlichen Frequenzen

3.4 Robot Operating System (ROS)

Das Robot Operating System (ROS) ist ein flexibles Framework zur Entwicklung von Software für Robotersysteme. Es bietet eine Vielzahl von Tools und Bibliotheken, die für die Entwicklung von Roboteranwendungen geeignet sind, angefangen bei kleinen Forschungsprojekten bis hin zu großen industriellen Systemen. ROS wurde konzipiert, um die Softwareentwicklung für Roboter zu vereinfachen, indem es modulare und wiederverwendbare Softwarekomponenten bereitstellt. Dies ermöglicht Entwicklern die Integration und Steuerung verschiedener Sensoren, Aktuatoren und Algorithmen. Die von mir verwendeten Kernelemente von Ros stelle ich in der Folge kurz vor.

Nodes In ROS werden Programme als **Nodes** bezeichnet. Eine Node fungiert als unabhängige Ausführungseinheit, die eine spezifische Aufgabe innerhalb des Robotersystems übernimmt, wie beispielsweise die Verarbeitung von Sensordaten oder die Steuerung eines Aktuators. Die Kommunikation zwischen den Nodes erfolgt über das ROS-Kommunikationssystem.

Topics sind Kommunikationskanäle, die es Nodes ermöglichen, Nachrichten auszutauschen. Eine Node kann Nachrichten an ein Topic veröffentlichen (Publisher) oder von einem Topic abonnieren (Subscriber). Diese Publish-Subscribe-Architektur ermöglicht eine flexible und skalierbare Kommunikation zwischen den Nodes.

Launch Files sind XML-Dateien, die verwendet werden, um mehrere Nodes und deren Konfigurationen gleichzeitig zu starten. Sie bieten eine benutzerfreundliche Möglichkeit, komplexe Roboteranwendungen zu verwalten und zu betreiben.

3.4.1 Rviz: Visualisierungswerkzeug für ROS

Rviz (ROS Visualization) ist ein leistungsstarkes Visualisierungswerkzeug, das in der ROS-Umgebung weit verbreitet ist [3]. Es dient dazu, Sensordaten, Zustände und andere relevante Informationen von Robotersystemen in einer grafischen Benutzeroberfläche darzustellen. Rviz bietet eine intuitive Benutzeroberfläche, in der verschiedene Datenquellen und deren Darstellungen konfiguriert werden können. Zu den wesentlichen Funktionen gehören: die Anzeige von Sensordaten, die Visualisierung der Robotergeometrie, interaktive Markierungen und die Darstellung von Frames und Koordinatensystemen [4].

Rviz kann Echtzeitdaten von Sensoren wie Kameras, LiDARs, IMUs und mehr visualisieren [4]. Die Visualisierung der Robotergeometrie basiert auf URDF (Unified Robot Description Format)-Dateien, einschließlich der Gelenkzustände, Positionen und Orientierungen der Roboterteile [3].

3.5 Der Madgwick-Filter

Der Madgwick-Filter ist ein algorithmisches Verfahren zur Fusionsverarbeitung von Sensordaten aus Inertial Measurement Units (IMUs), insbesondere Beschleunigungssensoren, Gyroskopen und Magnetometern. Dieser Filter wurde von Sebastian Madgwick im Rahmen seiner Doktorarbeit entwickelt und stellt eine effiziente Methode zur Bestimmung von Orientierung und Lage eines Objekts in Echtzeit dar [5].

Grundprinzip Der Madgwick-Filter verwendet eine quaternionsbasierte Methode zur Berechnung der Orientierung, was die Nachteile der klassischen Euler-Winkel-Methode, wie Gimbal Lock, umgeht [5].

Mathematische Grundlagen Der Filter kombiniert die Daten von Beschleunigungssensoren, Gyroskopen und Magnetometern, um eine präzise und stabile Orientierungsschätzung zu liefern. Die Gyroskopdaten liefern Informationen über die Winkelgeschwindigkeit des Objekts, während die Beschleunigungsdaten Aufschluss über die Richtung der Schwerkraft geben. Zudem liefern die Magnetometerdaten Informationen über das Erdmagnetfeld und helfen bei der Bestimmung der absoluten Ausrichtung.

Die aktualisierte Quaternion-Orientierung wird durch die Integration der Gyroskopdaten und die Korrektur mit den Beschleunigungs- und Magnetometerdaten berechnet. Die entsprechende Gleichung lautet:

$$\dot{q} = \frac{1}{2} q \otimes \omega - \beta e \tag{1}$$

wobei $\dot{q}$ die Änderungsrate der Quaternion, q die aktuelle Quaternion, ω die gemessene Winkelgeschwindigkeit, β eine Anpassungsrate und e der Fehler ist [5].

Vorteile des Madgwick-Filters Der Madgwick-Filter bietet mehrere Vorteile gegenüber anderen Fusionsalgorithmen. Zum einen ist der Algorithmus recheneffizient und eignet sich daher für den Einsatz in Echtzeitanwendungen [6]. Darüber hinaus liefert er durch die Kombination von Gyroskop-, Beschleunigungs- und Magnetometerdaten eine hohe Genauigkeit bei der Orientierungsschätzung [6]. Ein weiterer Vorteil ist, dass der Algorithmus robust gegenüber Messrauschen ist und stabile Ergebnisse liefert [6].

Aufgrund dieser Eigenschaften ist der Madgwick-Filter für die Verwendung in Datenhandschuhen gut geeignet [5].

3.6 Die Ferraris-Kalibrierung

Die Ferraris-Kalibrierung ist ein Verfahren zur präzisen Kalibrierung von Inertial Measurement Units (IMUs) [7]. Das Verfahren ermöglicht die genaue Bestimmung und Korrektur von Sensorfehlern, was zu einer signifikanten Verbesserung der Messgenauigkeit führt.

Grundprinzip Die Ferraris-Kalibrierung basiert auf der Verwendung einer mechanischen Vorrichtung, die die IMUs präzise und kontrolliert bewegt. Durch die exakte Kenntnis dieser Bewegungen und der damit verbundenen Sensordaten können die Fehlerparameter der IMUs ermittelt und korrigiert werden. Zu den typischen Fehlerparametern gehören Bias, Skalenfaktorfehler, Nichtlinearitäten und Kreuzkopplungen zwischen den Achsen [7].

Mathematische Grundlagen Die Kalibrierung erfolgt in mehreren Schritten. Zunächst wird die IMU kontrollierten Bewegungen unterzogen, wie zum Beispiel Rotationen und Translationsbewegungen. Während dieser Bewegungen werden die Rohdaten der Beschleunigungs- und Gyroskopsensoren aufgezeichnet. Anschließend werden die aufgezeichneten Rohdaten mit den erwarteten theoretischen Werten verglichen, um die Fehlerparameter zu ermitteln. Diese Fehlerparameter umfassen unter anderem Bias-Werte und Skalenfaktorfehler. Die berechneten Fehlerparameter werden dann auf die Rohdaten angewendet, um die Messgenauigkeit zu verbessern.

Mathematisch lässt sich dieser Prozess folgendermaßen darstellen:

$$\mathbf{y} = \mathbf{S}(\mathbf{x} - \mathbf{b}) + \mathbf{n} \tag{2}$$

Hierbei steht $\mathbf{y}$ für die kalibrierten Sensordaten, $\mathbf{S}$ für die Sensormatrix, $\mathbf{x}$ für die Rohdaten, $\mathbf{b}$ für die Bias-Werte und $\mathbf{n}$ für das Rauschen [7].

Vorteile der Ferraris-Kalibrierung Die Ferraris-Kalibrierung bietet mehrere Vorteile gegenüber anderen Kalibrierungsmethoden. Zum einen führt die Anwendung der berechneten Fehlerparameter zu einer signifikanten Reduzierung der Messfehler und einer Verbesserung der Datenqualität. Darüber hinaus ist die Ferraris-Kalibrierung in einer breiten Palette von Anwendungen einsetzbar und eignet sich für verschiedene Arten von IMUs.

Die Ferraris-Kalibrierung stellt somit eine äußerst effektive Methode zur Verbesserung der Messgenauigkeit von IMUs in Datenhandschuhen dar.

3.7 Berechnung von relativen Winkeln

Die Berechnung der relativen Winkel zwischen zwei IMUs ist grundlegend erforderlich für die Anwendung von IMUs in Datenhandschuhen. Die Ausrichtung eines Sensors am Finger wird dabei mit der Ausrichtung des Referenzsensors in der Handfläche verglichen. Dies ermöglicht es Rückschlüsse auf die Bewegungen des Fingers und seine derzeitige Position zu ziehen.

Grundlagen von Quaternions Quaternionen stellen eine effiziente Methode zur Beschreibung von Rotationen im dreidimensionalen Raum dar. Ein Quaternion $\mathbf{q}$ wird allgemein wie folgt dargestellt:

$$\mathbf{q} = \begin{pmatrix} q_w \\ q_x \\ q_y \\ q_z \end{pmatrix}$$

Hierbei ist q_w der Skalaranteil und (q_x, q_y, q_z) der Vektoranteil des Quaternions. Im Vergleich zu anderen Methoden der Rotationsdarstellung, wie z. B. Euler-Winkeln oder Rotationsmatrizen, bieten Quaternionen den Vorteil, das Gimbal-Lock-Problem zu vermeiden.

Berechnung eines relativen Winkels zwischen zwei Quaternions Wenn beide Quaternionen die Orientierung korrekt repräsentieren, gestaltet sich die Berechnung relativ einfach. Die relative Rotation zwischen zwei Quaternionen $\mathbf{q}_1$ und $\mathbf{q}_2$ wird durch die Inverse des ersten Quaternions und die Multiplikation mit dem zweiten Quaternion berechnet:

$$\mathbf{q}_{\text{rel}} = \mathbf{q}_1^{-1} \cdot \mathbf{q}_2$$

Die resultierende Quaternion $\mathbf{q}_{\text{rel}}$ beschreibt die relative Orientierung von $\mathbf{q}_2$ gegenüber $\mathbf{q}_1$. Diese kann anschließend in Euler-Winkel umgewandelt werden, um die relativen Winkel darzustellen.

Komplikationen durch einen undefinierten Gierwinkel Ohne Magnetsensoren ist der Gierwinkel der Sensoren undefiniert, was zu Inkonsistenzen in der initialen Orientierung der Sensoren führen kann, was in der Folge zu falschen relativen Winkeln führt. Durch das Mitteln der ersten Positionen der Sensoren und die umgekehrte Rotation dieser auf die derzeitige Orientierung des Sensors werden die Sensoren in ein gemeinsames Referenzkoordinatensystem transformiert.

$$\mathbf{q}_{\text{transformed}} = \mathbf{q}_{\text{initial}}^{-1} \cdot \mathbf{q}$$

Diese Transformation stellt sicher, dass alle Quaternionen relativ zur Anfangsposition der Sensoren betrachtet werden.

Komplikationen durch unterschiedlich ausgerichtete Sensoren Die Änderung von der anfänglichen Ausrichtung zur tatsächlichen Position der Sensoren muss ebenfalls berücksichtigt werden. Da die Sensoren in der Realität an den Fingern montiert sind und nicht alle genau gleich

ausgerichtet sind, muss die anfängliche Fehlstellung erfasst und korrigiert werden. Dies geschieht durch Multiplikation des aktuellen Quaternions mit dem inversen Quaternion, das die anfängliche Fehlstellung beschreibt:

$$\mathbf{q}_{\text{finger\_corrected}} = \mathbf{q}_{\text{finger}} \cdot \mathbf{q}_{\text{reference}}^{-1}$$

Hierbei ist $\mathbf{q}_{\text{finger}}$ die gemessene Quaternion des Fingersensors und $\mathbf{q}_{\text{reference}}$ die Quaternion, die die anfängliche Fehlstellung des Fingersensors beschreibt. Die resultierende Quaternion $\mathbf{q}_{\text{finger\_corrected}}$ berücksichtigt die anfängliche Ausrichtung und kann verwendet werden, um korrekte relative Winkel zu berechnen.

4 Stand der Technik

In diesem Kapitel werden die aktuellen Technologien und Methoden untersucht, die für das Verständnis und die Umsetzung des Projekts relevant sind. Zunächst erfolgt eine Kategorisierung der verschiedenen Bewegungserfassungssysteme. Hierbei werden optische, inertiale, Ultraschall- und hybride Systeme beschrieben, die zur präzisen Messung von Bewegungen und Positionen eingesetzt werden.

Anschließend wird die Kategorisierung haptischer Geräte behandelt. Es werden die unterschiedlichen Technologien zur Erzeugung von taktilem Feedback, Kraft-Feedback und thermischem Feedback erläutert, sowie multimodale haptische Geräte, die mehrere Feedback-Arten kombinieren.

Im Abschnitt über bestehende Lösungen und Systeme wird der SenseGlove Nova vorgestellt, ein fortschrittlicher haptischer Handschuh. Dabei werden seine technischen Merkmale, Anwendungsmöglichkeiten und die Herausforderungen bei der Nutzung kommerzieller Produkte, wie hohe Kosten und fehlende Open-Source-Verfügbarkeit, diskutiert.

4.1 Kategorisierung von Bewegungserfassungssystemen

Bewegungserfassungssysteme messen präzise die Bewegungen und Positionen von Objekten und Menschen. Die Hauptkategorien sind optische, inertiale, Ultraschall- und hybride Systeme.

Optische Erfassungssysteme nutzen Licht, um Bewegungen und Positionen zu detektieren. Verfahren wie das Lichtschnittverfahren, Streifenprojektion und Photogrammetrie ermöglichen präzise 3D-Messungen [8]. Weitere Methoden wie Weißlicht-Interferometrie und Lasermesstechnik bieten sehr genaue Oberflächen- und Distanzmessungen, besonders in der industriellen Messtechnik [8].

Inertiale Erfassungssysteme verwenden Gyroskope und Beschleunigungssensoren zur Messung von Bewegung und Rotation. Diese Systeme sind besonders nützlich in tragbaren Geräten und der VR. Die Sensorfusion kombiniert Sensordaten, um die Messgenauigkeit zu erhöhen und langfristige Driftprobleme zu minimieren [9].

Ultraschall-Erfassungssysteme nutzen Schallwellen, die von Objekten reflektiert werden, um Positionen und Bewegungen zu messen. Sie arbeiten unabhängig von Lichtverhältnissen und sind besonders für Abstandsmessungen in dunklen oder transparenten Umgebungen geeignet [10].

Hybride Systeme kombinieren verschiedene Erfassungsmethoden, wie optische und inertiale Sensoren, um die Präzision zu erhöhen und die Schwächen der einzelnen Technologien auszugleichen. Sie werden in Bereichen wie VR, Robotik und Medizintechnik eingesetzt [11].

Diese Technologien bieten in ihren jeweiligen Anwendungsbereichen bedeutende Vorteile und tragen zur Entwicklung präziserer und robusterer Systeme bei.

4.2 Kategorisierung haptischer Geräte

Haptische Geräte erzeugen physikalische Empfindungen, die den Tastsinn ansprechen, und lassen sich in verschiedene Kategorien unterteilen: taktiles Feedback, Kraft-Feedback, thermisches Feedback und multimodale haptische Geräte.

Taktiles Feedback umfasst Technologien, die Berührungs- und Druckempfindungen simulieren. Die bekannteste Form sind vibrotaktile Aktuatoren, die Vibrationen erzeugen. Diese Geräte nutzen kleine Motoren, die kostengünstig und einfach in verschiedene Technologien integrierbar sind. Die Anpassung der Intensität und Frequenz ermöglicht unterschiedliche taktile Empfindungen [12]. Eine weitere Technik ist elektrotaktiles Feedback, bei dem elektrische Impulse Kribbeln oder Vibrationen erzeugen. Thermische Stimulation nutzt Hitze oder Kälte, um realistische Erlebnisse zu bieten, etwa in der virtuellen Realität und Medizintechnik [12]. Kontaktlose Systeme, die Ultraschallwellen verwenden, erzeugen Druckreize ohne direkten Hautkontakt.

Kraft-Feedback-Systeme simulieren physikalischen Widerstand oder direkte Kräfte bei der Interaktion mit virtuellen Objekten. Diese lassen sich in passives Kraft-Feedback, bei dem mechanische Vorrichtungen Widerstände erzeugen, und aktives Kraft-Feedback, das direkte Kräfte auf den Körper ausübt, unterteilen. Aktive Systeme, wie Exoskelette und robotische Systeme, bieten präzisere Simulationen der Interaktion mit Objekten und sind teurer, aber auch technisch komplexer [12]. Seilzugsysteme wie das SPIDAR-System bieten realistisches Bewegungsfeedback.

Thermisches Feedback simuliert Temperaturveränderungen an der Haut, etwa durch Peltier-Elemente oder Widerstandsheizungen, und wird oft in der virtuellen Realität genutzt, um Umgebungen realistischer zu gestalten. Ein Beispiel ist ThermoReal, das sowohl Hitze als auch Kälte simulieren kann. In der Medizintechnik und Rehabilitation hilft thermisches Feedback, sensorische und motorische Fähigkeiten zu fördern und das Benutzererlebnis zu verbessern.

Multimodale haptische Geräte kombinieren taktiles, Kraft- und thermisches Feedback für ein umfassenderes Erlebnis. Ein Beispiel sind die HaptX Gloves, die vibrotaktile Aktuatoren und pneumatische Systeme kombinieren, um Druck- und Kraftempfindungen zu erzeugen [12]. Auch der SenseGlove kombiniert mechanischen Widerstand und taktiles Feedback. Diese Geräte bieten ein immersiveres Benutzererlebnis, erfordern jedoch fortschrittliche Hardware und Software, um die verschiedenen Feedback-Arten zu synchronisieren.

4.3 Bestehende Lösungen und Systeme

Der SenseGlove Nova [13] ist ein fortschrittlicher haptischer Handschuh, der entwickelt wurde, um ein realistisches und immersives haptisches Erlebnis zu bieten. Der Handschuh ist mit mehreren Sensoren und Aktuatoren ausgestattet, die präzise Bewegungen und taktiles Feedback ermöglichen (Abbildung 2).

Abbildung 2: Senseglove [13]

Über die Finger und die Handfläche sind vibrotaktile Aktuatoren verteilt, die Vibrationen erzeugen können, um taktile Rückmeldungen zu simulieren. Diese Aktuatoren sind in der Lage, Vibrationen mit verschiedenen Frequenzen zu erzeugen, wodurch unterschiedliche haptische Empfindungen vermittelt werden.

Das Kraft-Feedback-System des SenseGlove verwendet magnetische Reibungsbremsen, die einen mechanischen Widerstand von bis zu 20 N erzeugen können. Dies ermöglicht es den Benutzern, das Gewicht und die Steifheit von virtuellen Objekten realistisch zu erfahren. Zusätzlich zu den taktilen und Kraft-Aktuatoren ist der SenseGlove mit präzisen Sensoren ausgestattet, die die Bewegungen der Hände und Finger millimetergenau erfassen und verfolgen.

Durch die Integration dieser Technologien kann der SenseGlove in einer Vielzahl von Anwendungen eingesetzt werden, darunter VR, die Robotik und die medizinische Simulation. Diese Vielseitigkeit macht ihn zu einem wertvollen Werkzeug für die Forschung und Entwicklung in verschiedenen Bereichen.

Das Problem mit der Verwendung von kommerziellen Produkten ist hierbei der oftmals hohe Preis, sowie dass es sich im Generellen um proprietäre Software handelt. Insbesondere zweiteres ist ein große Problem für die wissenschaftliche Forschung an einer Universität.

5 Hardware-Entwicklung

In diesem Kapitel wird die Entwicklung der Hardware des Datenhandschuhs behandelt, einschließlich der verwendeten Sensoren, Mikrocontroller und Erweiterungen wie die Vibrationsmotoren. Der vollständige Schaltplan des Systems ist in Unterabschnitt A.1 dieser Arbeit zu finden und zeigt die Verbindungen zwischen den verschiedenen Komponenten.

5.1 Übergabestand des Systems von Can Bagdas

Dieser Abschnitt behandelt den Übergabestand des Systems, das von Can Bagdas in seiner Bachelorarbeit "Entwicklung eines Prototypen für einen Datenhandschuh basierend auf IMUs - Version 2" (2022) entwickelt wurde [14]. Das System besteht aus zwei individuell gestalteten Datenhandschuhen, die jeweils mit einem ESP32-Mikrocontroller ausgestattet sind.

Die sechs IMU-Sensoren sind so angeordnet, dass sie die Hand- und Fingerbewegungen erfassen können. Ein Sensor in der Handfläche dient als Referenz (MPU9050), während die anderen fünf in der Nähe der Fingerspitzen platziert sind.

Die verwendeten IMU-Sensoren in den Fingerspitzen umfassen die Modelle MPU-6050 (linke Hand) und BMI160 (rechte Hand), die zu Testzwecken aufgrund ihrer unterschiedlichen Eigenschaften in Bezug auf Datenrate und Rauschverhalten ausgewählt wurden. In seiner Arbeit stellte Bagdas fest, dass der MPU-6050 eine höhere Datenrate und dadurch eine bessere Eignung für den Einsatz in einem Datenhandschuh zeigt [14].

Die grundlegenden Entscheidungen in Bezug auf die Sensorauswahl und die Systemarchitektur legen den Grundstein für die Weiterentwicklung und Optimierung des Datenhandschuhs in dieser Bachelorarbeit.

Can Bagdas Arbeit hat eine hervorragende Basis geschaffen haben, auf der weiter aufgebaut werden kann. Eine der Herausforderungen bestand darin, dass noch nicht alle Funktionen in das System integriert waren. Insbesondere fehlten einfache Möglichkeiten zur Visualisierung der Daten in ROS. Außerdem gab es im ursprünglichen System noch kein haptisches Feedback, was eine wertvolle Ergänzung für die Interaktionseffizienz des Handschuhs darstellen würde.

Der Schaltplan von Can Bagdas zeigt die ursprüngliche Hardwarearchitektur des linken Handschuhs und ist in Abbildung 3 dargestellt. In den folgenden Unterkapiteln werden die verwendeten Komponenten vorgestellt.

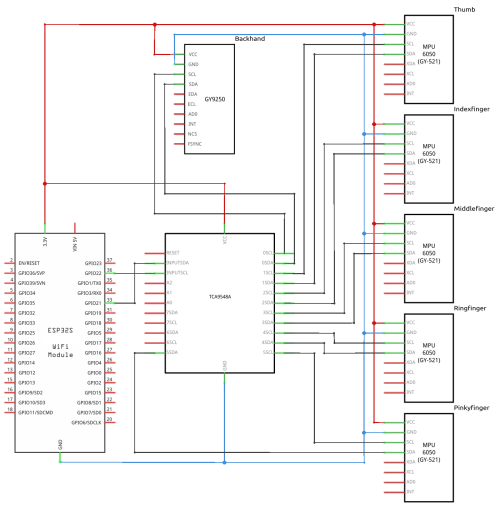

Abbildung 3: Schaltplan des ursprünglichen Systems von Can Bagdas [14]

5.1.1 IMUs

MPU9050: Der MPU9050 ist ein fortschrittlicher IMU-Sensor, der ein Gyroskop, einen Beschleunigungssensor sowie ein Magnetometer enthält. Zu den Hauptmerkmalen gehören ein 3-Achsen-Gyroskop und ein 3-Achsen-Beschleunigungssensor, die eine präzise Erfassung von Bewegungen in verschiedenen Dimensionen ermöglichen. Der Sensor ist mit einer Digital Motion Processing (DMP) Einheit ausgestattet, die die Berechnung der Orientierung direkt im Sensor durchführt. Dies ermöglicht eine effiziente und genaue Bewegungsverfolgung. Der MPU9050 verfügt sowohl über I2C- als auch über SPI-Schnittstellen, die die Kommunikation mit Mikrocontrollern erleichtern. Zu den technischen Spezifikationen des Sensors gehören ein Gyroskop-Rauschen von 0,005 $°/s/\sqrt{Hz}$ und ein Beschleunigungssensor-Rauschen von 400 $\mu g/\sqrt{Hz}$. Der Gyroskopbereich ist in vier Stufen einstellbar: ±250°/s, ±500°/s, ±1000°/s und ±2000°/s. Ebenso ist der Beschleunigungsbereich in vier Stufen einstellbar: ±2g, ±4g, ±8g und ±16g. Der MPU9050 enthält außerdem ein 3-Achsen-Magnetometer (AK8963) mit einer Auflösung von 14 Bit und einem Messbereich von ±4800 µT. [15]

MPU6050: Der MPU6050 ist einer der am häufigsten verwendeten IMU-Sensoren in der Industrie. Er integriert ein 3-Achsen-Gyroskop und einen 3-Achsen-Beschleunigungssensor auf einem Chip. Die Hauptmerkmale des MPU6050 sind ein 3-Achsen-Gyroskop und ein 3-Achsen-Beschleunigungssensor, die eine präzise Erfassung von Bewegungen in verschiedenen Dimensionen ermöglichen. Der Sensor verfügt über eingebaute digitale Temperatursensoren und einen integrierten FIFO-Puffer zur Reduzierung der Arbeitslast des Mikrocontrollers. Der MPU6050 unterstützt I2C-Schnittstellen. Das Gyroskop-Rauschen beträgt 0,03 $°/s/\sqrt{Hz}$ und das Beschleunigungssensor-Rauschen 400 $\mu g/\sqrt{Hz}$. Der Gyroskopbereich ist in vier Stufen einstellbar: ±250°/s, ±500°/s, ±1000°/s und ±2000°/s. Der Beschleunigungsbereich ist ebenfalls in vier Stufen einstellbar: ±2g, ±4g, ±8g und ±16g. [16]

BMI160: Der BMI160 ist ein hocheffizienter IMU-Sensor von Bosch, der sowohl ein 3-Achsen-Beschleunigungssensor als auch ein 3-Achsen-Gyroskop in einem kleinen Gehäuse kombiniert. Seine Merkmale umfassen hohe Empfindlichkeit und geringe Rauschwerte für präzise Messungen, einen integrierten FIFO-Puffer zur Entlastung des Hostprozessors sowie die Unterstützung von I2C- und SPI-Schnittstellen. Das Gyroskop-Rauschen beträgt 0,007 $°/s/\sqrt{Hz}$ und das Beschleunigungssensor-Rauschen 180 $\mu g/\sqrt{Hz}$. Der Gyroskopbereich ist in fünf Stufen einstellbar: ±125°/s, ±250°/s, ±500°/s, ±1000°/s und ±2000°/s. Der Beschleunigungsbereich ist ebenfalls in vier Stufen einstellbar: ±2g, ±4g, ±8g und ±16g. [17]

5.1.2 Der I²C-Bus

Der TCA9548A ist ein serieller Datenbus, der zur Kommunikation zwischen Mikrocontrollern und Peripheriegeräten wie Sensoren, Aktuatoren und Speicherchips verwendet wird. Er benötigt nur zwei Signalleitungen: eine für die Datenübertragung (SDA) und eine für den Takt (SCL). Diese einfache Struktur ermöglicht die Verbindung mehrerer Geräte über einen einzigen Mikrocontroller-Pin, was die Verkabelung und den Platzbedarf reduziert. Ein Beispiel für einen I²C-Multiplexer ist der TCA9548A, der es ermöglicht, mehrere I²C-Geräte mit demselben Mikrocontroller zu verbinden, selbst wenn diese Geräte dieselbe I²C-Adresse verwenden. Dies wird erreicht, indem der Multiplexer den I²C-Bus in bis zu acht separate Kanäle aufteilt, die individuell angesprochen werden können. Zu den technischen Spezifikationen des TCA9548A gehören eine Betriebsspannung von 1,65 V bis 5,5 V und ein konfigurierbarer I²C-Adressenbereich von 0x70 bis 0x77. Der Multiplexer unterstützt Kommunikationsgeschwindigkeiten von bis zu 400 kHz im I²C-Standard- und Fast-Mode und hat eine typische Leistungsaufnahme von 1,5 mA im Betriebsmodus. Diese Ei-

genschaften machen den TCA9548A zu einem vielseitigen und leistungsfähigen Baustein für die Erweiterung von I²C-Systemen. [18]

5.1.3 Mikrocontroller

Der ESP32 Mikrocontroller bietet eine leistungsstarke Plattform für die Entwicklung von IoT-Anwendungen und die Steuerung von Geräten. Der ESP32 verfügt über einen Dual-Core Xtensa LX6 CPU, der mit bis zu 240 MHz getaktet werden kann. Der Mikrocontroller besitzt 520 KB SRAM und 448 KB ROM, was ausreichend Speicher für viele Anwendungen bietet. Zu den unterstützten Schnittstellen gehören SPI, I2C, UART, ADC, DAC, PWM und GPIO, was eine vielseitige Einbindung in verschiedene Projekte ermöglicht.

Ein Vorteil des ESP32 ist sein niedriger Energieverbrauch, was ihn ideal für batteriebetriebene Geräte macht. Darüber hinaus bietet der Mikrocontroller umfangreiche Sicherheitsfunktionen wie Secure Boot, Flash Encryption und eine Trusted Execution Environment, um die Sicherheit der Anwendungen zu gewährleisten. [19]

5.2 Nötige Erweiterungen

5.2.1 Vibrationsmotoren

Vibrationsmotoren sind kleine, leistungsstarke Geräte, die Vibrationen erzeugen, um haptische Rückmeldungen zu liefern. Sie werden häufig in Smartphones und anderen tragbaren Geräten verwendet. Der VG0832022D Vibrationsmotor (siehe Abbildung 4) ist ein Beispiel für einen solchen Motor, der in verschiedenen Anwendungen eingesetzt werden kann. Der VG0832022D Vibrationsmotor ist ein LRA (Linear Resonant Actuator) mit einer Betriebsspannung von 1,8 Vrms und einer Resonanzfrequenz von 235 Hz. Der Motor hat einen Durchmesser von 8 mm und eine Dicke von 3,2 mm. Der maximale Nennstrom beträgt 80 mA, während der typische Strom bei 58 mA liegt. Die Vibrationskraft beträgt 1,00 Grms bei 50% der maximalen G-Kraft. Dieser Motor wird häufig in Smartphones verwendet, um Benachrichtigungsvibrationen zu erzeugen, sowie in Wearables, um haptisches Feedback in Fitness-Trackern und Smartwatches zu übermitteln. [20]

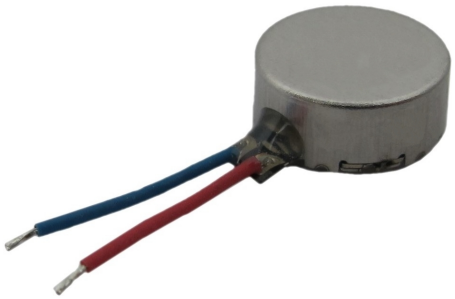

Abbildung 4: VG0832022D-Vibrationsmotor [21]

5.2.2 Motorkontroller

Der ULN2803A ist ein weit verbreiteter Motorkontroller, der häufig in Projekten eingesetzt wird, die mehrere Motoren steuern müssen. Er ist ein Hochspannungs- und Hochstrom-Darlington-Transistor-Array, das eine einfache und effektive Lösung für die Ansteuerung von Motoren, Relais und anderen Lasten bietet. Der ULN2803A verfügt über acht Kanäle und kann einen maximalen Ausgangsstrom von 500 mA pro Kanal liefern. Die maximale Kollektor-Emitter-Spannung beträgt 50 V, und die Eingangsspannung ist 5 V TTL- und CMOS-kompatibel. Das Bauteil ist in einem 18-poligen DIP-Gehäuse untergebracht und verfügt über integrierte Freilaufdioden für induktive Lasten. [22]

5.2.3 Platzierung der Motoren an der Hand

Der vorgestellte Motorkontroller ermöglicht die Steuerung von acht Vibrationsmotoren. Die Entscheidung, auf den Einsatz von zwei Motorkontrollern zu verzichten, die insgesamt sechzehn Motoren versorgen könnten, wurde bewusst getroffen, um die Komplexität zu reduzieren und die Benutzerfreundlichkeit der Handschuhe zu gewährleisten. In Abbildung 6 ist die minimale Anzahl notwendiger Aktuatoren sowie deren Platzierung dargestellt, um eine haptische Rückmeldung für jeden Finger sowie die Handfläche zu ermöglichen. Die rot markierten Motoren stellen das absolute Minimum dar. Von den insgesamt acht verfügbaren Motoren sind somit zwei (grün markiert) übrig.

Abbildung 5: Handschuh mit installierten Vibrationsmotoren

Es wird vorgeschlagen, diese beiden verbleibenden Motoren zu nutzen, um zusätzliche haptische Rückmeldungen zu ermöglichen. In Abbildung 5 sowie Abbildung 7 sind diese Motoren zusätzlich in der Mitte des Zeige- und Mittelfingers positioniert, da diese Finger aufgrund ihrer zentralen Rolle bei Greif- und Feinmotorikbewegungen besonders hervorgehoben werden sollten. Abbildung 8 zeigt die vorgeschlagene Positionierung der Motoren bei Tätigkeiten wie dem Schreiben. Die zwei verbleibenden Motoren bieten somit eine gewisse Flexibilität, wobei ihre genaue Platzierung je nach Anwendungsfall angepasst werden muss.

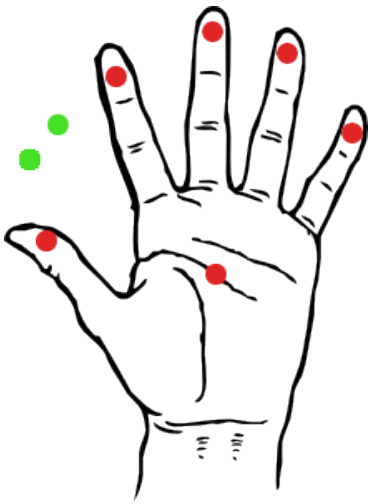

Abbildung 6: Minimale Anzahl notwendiger Aktuatoren

Abbildung 7: Aktuatoren in Standardposition

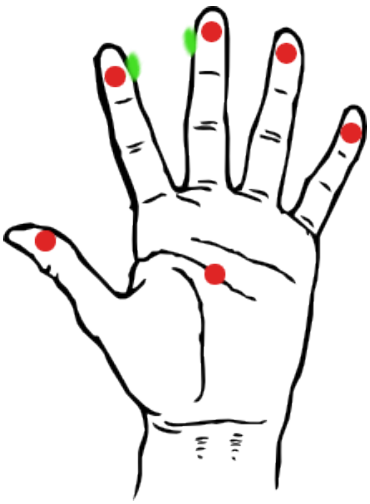

Abbildung 8: Aktuatoren in Schreibposition

6 Firmware-Entwicklung

In diesem Kapitel wird die Firmware des ESP32-Mikrocontrollers beschrieben, der für die Steuerung von Motoren und das Auslesen von Sensoren zuständig ist. Der Mikrocontroller verarbeitet Sensordaten und überträgt diese über Wi-Fi an eine Kontrolleinheit. Darüber hinaus empfängt er Steuerbefehle zur Motorsteuerung und verarbeitet diese zur Ausgabe von haptischem Feedback.

6.1 Programmaufbau

In Abbildung 9 ist der Firmware Programmaufbau gezeigt.

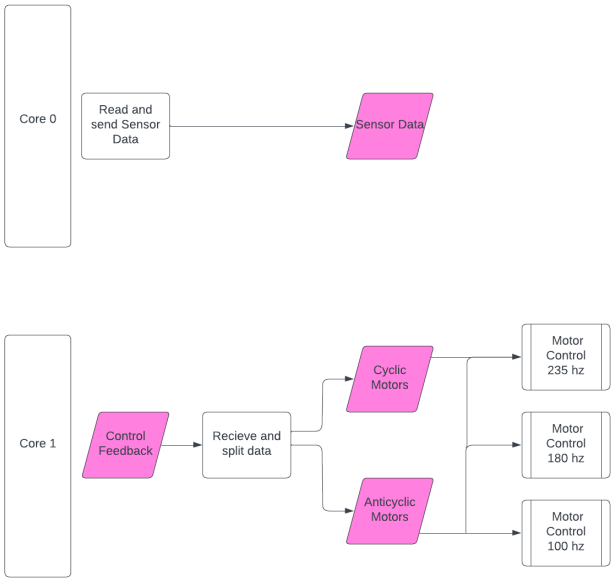

Abbildung 9: Schema des Programmaufbaus der Firmware auf dem ESP32-Mikrokontroller.

Auf ”Kern 0” werden kontinuierlich die 6 IMUs ausgelesen und deren Rohdaten über WiFi an die Kontrolleinheit gesendet. Die Datenrate ist hierbei von entscheidender Bedeutung und auf zusätzliche Aufgaben auf diesem Kern wurde verzichtet. Ursprünglich waren 6 Threads (einer für jeden Sensor) vorgesehen, um z.B. dem wichtigeren Referenzsensor in der Handfläche eine höhere Prorität geben zu können. Auf diesen Aufbau wurde jedoch verzichtet, da der zusätzliche Overhead und die unregelmäßigere Datenrate zu einer stark gesteigerten Komplexität, bei der weiteren Datenverarbeitung auf der Kontrolleinheit geführt hat.

”Kern 1” übernimmt das Empfangen des ”Control Feedbacks” also den Daten, welche dem Programm mitteilen welche Motoren in welcher Stärke vibrieren sollen. 3 Threads steuern diese Motoren dann in der entsprechenden Frequenz an.

6.1.1 Bibliotheken und Konstanten

Der Code beginnt mit dem Einbinden der notwendigen Bibliotheken und der Definition von Konstanten:

Hier können wesentliche Parameter leicht verändert werden, um die Systemleistung oder die Kommunikation zu optimieren.

WiFi-Verbindung: Die WiFi-Verbindung muss konfiguriert werden, damit der Mikrocontroller korrekt mit der Kontrolleinheit kommunizieren kann. Änderungen an den WiFi-Verbindungsparametern (SSID, Passwort, IP-Adresse und Ports) sollten vorgenommen werden, wenn eine neue Netzwerkumgebung erforderlich ist. Ebenso müssen diese Parameter angepasst werden, wenn sich die

IP-Adresse der Kontrolleinheit oder der Kommunikationsport ändern, um mögliche Konflikte mit anderen Geräten zu vermeiden. Weitere Änderungen sind erforderlich, wenn eine Sicherheitsaktualisierung notwendig ist, etwa durch das Ändern des WLAN-Passworts.

```
const char* ssid = "glovecontrol";
const char* password = "password";

const char* udpAddress = "10.42.0.1";
const uint16_t udpPort = 8080;
const uint16_t listenPort = 8082;
```

Motorengeschwindigkeiten: Die maximal erreichbare Vibration wird durch eine Frequenz von 235 Hz bestimmt, und abweichende Werte führen zu schwächerer Vibration. Diese Parameter können angepasst werden, um unterschiedliche Vibrationsstärken zu erzeugen, die für spezifische Anwendungen erforderlich sind. Eine Anpassung der Frequenzen kann beispielsweise notwendig sein, um die Vibrationen an die Bedürfnisse der Nutzer anzupassen. Weitere Details zur Frequenzmodulation und deren Auswirkungen auf die Motorsteuerung, insbesondere bei der Steuerung von Wechselstrommotoren, sind in Abschnitt 3.3 zu finden.

```
const int lowMotorFreq = 100;   // Frequenz für niedrige Motorsteuerung
const int midMotorFreq = 160;   // Frequenz für mittlere Motorsteuerung
const int highMotorFreq = 235;  // Frequenz für hohe Motorsteuerung
```

Sensorbereiche: Die Messbereiche für den Beschleunigungs- und Gyroskopsensor können angepasst werden, um entweder eine höhere Präzision oder einen erweiterten Messbereich zu ermöglichen. Änderungen an diesen Parametern sollten vorgenommen werden, wenn die Anforderungen an die Sensordaten sich ändern. So kann der Sensorbereich vergrößert werden, um höhere Beschleunigungen oder größere Drehgeschwindigkeiten zu messen, was besonders in Anwendungen mit schnellen Bewegungen von Vorteil sein kann. Umgekehrt kann der Bereich auch verringert werden, wenn feinere Messungen benötigt werden oder wenn die Genauigkeit in einem kleineren Bereich optimiert werden soll.

```
const int ACCEL_RANGE = 8;
const int GYRO_RANGE = 500;
```

6.1.2 Motorsteuerung

Die Motoren werden mithilfe von Pulsweitenmodulation gesteuert, um die Leistung der Wechselstrommotoren effektiv zu regulieren. Hierbei werden verschiedene Frequenzen verwendet, um die Vibrationen zu kontrollieren, die durch die Motoren erzeugt werden.

Zusätzlich wird die Motorsteuerung so implementiert, dass zyklische und antizyklische Betriebsarten verwendet werden. Diese Technik reduziert die maximale Last auf die Motoren, insbesondere beim gleichzeitigem Starten aller Motoren. Bei zyklischen Motoren werden die Motoren in regelmäßigen Intervallen aktiviert, antizyklische Motoren werden alternierend betrieben, um die Gesamtbelastung zu minimieren. Dies reduziert das Risiko von Überlastungen und Ausfällen, die durch plötzliche Laständerungen verursacht werden können.

Die Funktion `processIncomingData()` verwaltet die empfangenen Befehle zur Steuerung der Motoren:

```
void processIncomingData(const char* packet) {
int receivedStates[8] = {0};
sscanf(packet, "%d,%d,%d,%d,%d,%d,%d,%d",
&receivedStates[0], &receivedStates[1], &receivedStates[2], &receivedStates[3],
&receivedStates[4], &receivedStates[5], &receivedStates[6], &receivedStates[7]);

// Resetting the motorStatus array
for (int i = 0; i < 8; i++) {
if(i%2 == 0){
motorCyclic[i] = receivedStates[i];
}
else{
motorAntiCyclic[i] = receivedStates[i];
}
if(receivedStates[i] == 0){
digitalWrite(motorPins[i],LOW);
}
}
}
```

Die `motorControl`-Funktion wird in drei Instanzen ausgeführt, um unterschiedliche Frequenzen der Motorsteuerung zu ermöglichen. Sie verwendet ein zyklisches und antizyklisches Steuerungsschema, um eine Überlappung der Motoraktivitäten zu vermeiden. Die Methode schaltet zwischen zwei Zuständen hin und her: In einem Zustand werden die antizyklischen Motoren ausgeschaltet und die zyklischen Motoren eingeschaltet, und im anderen Zustand umgekehrt. Die Verzögerungszeiten (`delayTime` und `switchDelay`) werden sorgfältig berechnet, um sicherzustellen, dass die Motoren nicht gleichzeitig schalten, was die Belastung verringert und die Systemstabilität erhöht. Die Verwendung von `vTaskDelay((delayTime - switchDelay) / portTICK_PERIOD_MS)` am Ende des Schleifenzyklus dient dazu, die tatsächliche Verzögerung an die zuvor eingeführte Schaltverzögerung (`switchDelay`) anzupassen. Dies stellt sicher, dass die gesamte Verzögerungszeit korrekt eingehalten wird, wodurch eine gleichmäßige und präzise Steuerung der Motoren gewährleistet ist. Der `vTaskDelay`-Aufruf hilft dabei, die PWM-Frequenz einzuhalten, da er die Frequenz des Motorzyklus bestimmt und so eine präzise Regelung der Motorleistung ermöglicht.

```
1  void motorControl(void* pvParameters, int frequency, int group) {
2    bool state = LOW;
3    int delayTime = (1000 / (frequency * 2)); // Delay time in milliseconds
4    int switchDelay = 1; // Minimal delay in milliseconds to avoid simultaneous
         switching
5
6    while (true) {
7      // Turn off motors of the opposite type first
8      for (int i = 0; i < 8; i++) {
9        if ((state && motorAntiCyclic[i] == group) || (!state && motorCyclic[i] == group
             )) {
10         digitalWrite(motorPins[i], LOW);
11       }
12     }
13     vTaskDelay(switchDelay / portTICK_PERIOD_MS); // Delay to avoid overlap
14
15     // Turn on motors of the current type
16     for (int i = 0; i < 8; i++) {
17       if ((state && motorCyclic[i] == group) || (!state && motorAntiCyclic[i] == group
             )) {
18         digitalWrite(motorPins[i], HIGH);
19       }
20     }
21
22     state = !state;
23     //wait for next cycle, while adjusting for switch delay
24     vTaskDelay((delayTime - switchDelay) / portTICK_PERIOD_MS);
25   }
26 }
```

7 Software-Entwicklung in ROS

In diesem Kapitel wird die Software-Entwicklung im Kontext des Robot Operating Systems (ROS) beschrieben, die für die Erfassung und Verarbeitung von Handbewegungen mit Hilfe eines Datenhandschuhs erforderlich ist. Der Workflow dieser Softwarelösung wird im Detail erläutert, beginnend bei der Einrichtung des Systems über die Kalibrierung der Sensoren bis hin zur tatsächlichen Nutzung des Handschuhs für die Visualisierung und Interaktion mit externen Systemen.

Im Unterkapitel *Workflow* wird der gesamte Prozess von der Datenerfassung bis zur Verwendung des Handschuhs zur Hand- und Fingerbewegungserkennung beschrieben. Dabei werden die einzelnen Schritte wie Setup, Kalibrierung, Handabdruckgenerierung und die Visualisierung der Handposition mit ROS behandelt.

Das Unterkapitel *Beschreibung der Klasse* `IMUEulerPublisher` stellt eine spezifische ROS-Klasse vor, die dafür verantwortlich ist, die von den IMU-Sensoren gelieferten Rotationsdaten in Euler-Winkel umzuwandeln und in ROS zu publizieren. Der Aufbau und die Methoden dieser Klasse werden detailliert beschrieben.

Zusätzlich wird das Konzept des haptischen Feedbacks erläutert, das mittels UDP-Nachrichten an die Handschuhe gesendet wird, um den Benutzern taktiles Feedback zu den erfassten Handbewegungen zu bieten.

7.1 Workflow

Der Workflow im Robot Operating System umfasst mehrere Schritte, die in Abbildung 10 dargestellt sind. Die IMU-Daten werden über Wi-Fi empfangen und müssen bereinigt sowie kalibriert werden. Anschließend werden Orientierungen berechnet, die in Form von Quaternions dargestellt werden. Danach werden die relativen Winkel der Sensoren, welche an den Fingern befestigt sind, zum Referenzsensor in der Handfläche berechnet. Diese Winkel können zur Visualisierung der Handposition in Rviz verwendet oder von einem externen System, wie zum Beispiel der Steuerung einer Roboterhand, genutzt werden. Zusätzlich besteht die Möglichkeit, ”Control Feedback” an den Arduino zu senden, um die Vibrationsmotoren zu steuern. In den folgenden Kapiteln werden die einzelnen Stationen genauer betrachtet.

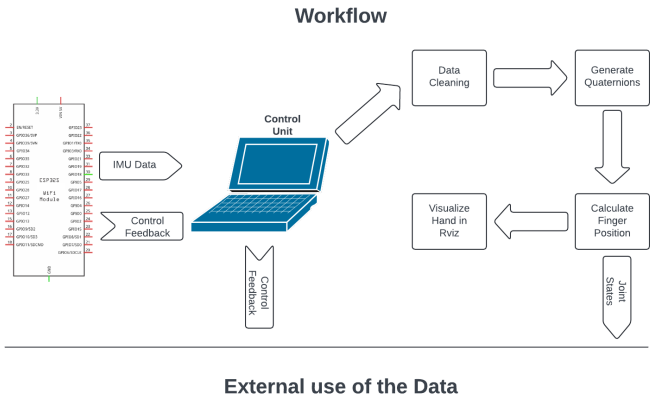

Abbildung 10: Workflow in ROS

1. Setup Der erste Schritt zur Verwendung der Datenhandschuhe ist das Einrichten eines Hotspots mit dem Namen ”glovecontrol” und dem Passwort ”password” von der Kontrolleinheit (zum Beispiel einem Laptop). Zusätzlich sollte mindestens einer der Datenhandschuhe eingeschaltet werden und ein ROS Master mittels des Kommandozeilenbefehls gestartet werden:

```
roscore
```

2. Kalibrierung Es wird davon ausgegangen, dass bislang entweder keine Kalibrierung durchgeführt wurde oder die letzte Kalibrierung nicht mehr ausreichend ist. Um eine Kalibrierung durchzuführen, müssen die Rohdaten zuerst vom System empfangen werden. Die einfachste Möglichkeit hierfür ist die Verwendung des folgenden Befehls:

```
roslaunch udp_datatransfer start_IMUgloves.launch
```

Die Kalibrierung wird dann mit folgendem Befehl gestartet:

```
rosrun udp_datatransfer imuCalibration.py --hand R
```

Das Programm führt Schritt für Schritt durch den Kalibrierungsprozess und kalibriert die mittels des Parameters –hand ausgewählte Hand (R für rechts, L für links). In der Praxis hat sich bewährt, die Sensoren in einer Box, deren Seiten mit den Achsen beschriftet sind, festzukleben (siehe Abbildungen 11 und 12).

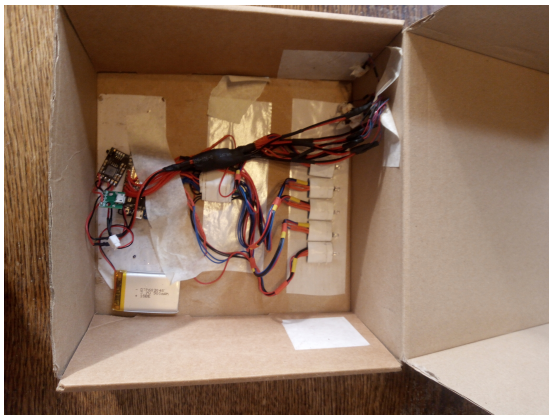

Abbildung 11: Kalibrierung der Sensoren - Sensoren festgeklebt

Abbildung 12: Kalibrierung der Sensoren - Box mit beschrifteten Achsen

Ist die Datenaquisition abgeschlossen, öffnet sich automatisch ein Plot (siehe Abbildung 13), der es ermöglicht Sektionen auszuwählen.Hier sollten Sektionen ausgewählt werden, in denen die Datenqualität gut erscheint.

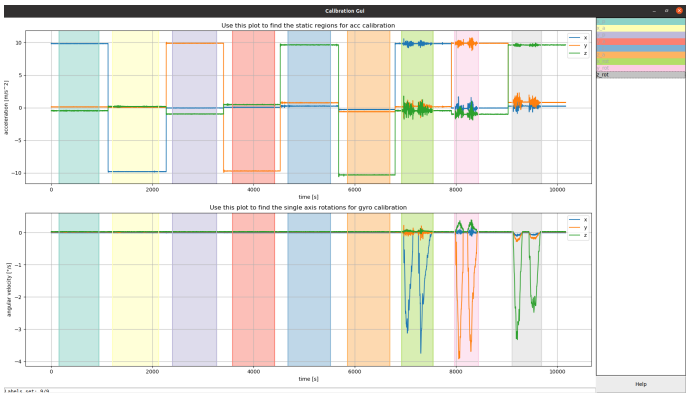

Abbildung 13: Sektionsauswahl während der Kalibrierung

Es ist darauf zu achten, dass die im Plot erkennbaren Rotationswerte im negativen Bereich liegen. Dies entspricht einer Rotation mit dem Uhrzeigersinn um die positiven Achsen [7]. Die Rotation sollte zudem möglichst exakt 360° abbilden, was in Abbildung 13 durch zweimaliges rotieren um jeweils 180° erfolgte. Damit die neue Kalibrierung verwendet wird ist an dieser Stelle ein Systemneustart erforderlich:

```
roslaunch udp_datatransfer start_IMUgloves.launch
```

3. Generierung des Handabdrucks Der Handabdruck wird generiert, um die relative Orientierung der Sensoren an der Hand nach der Kalibrierung zu speichern. Dies ist wichtig, da die genaue Ausrichtung der Sensoren für die präzise Erfassung der Handbewegungen entscheidend ist. Nachdem die Kalibrierung abgeschlossen ist, muss der Handschuh aus der Kalibrierungsbox genommen und angezogen werden. Um den Handabdruck zu generieren, muss der folgende Befehl verwendet werden:

```
rosrun joint_manipulator calculate_alignment_offset.py --hand R
```

Hierbei steht "R"für die rechte Hand und "L"für die linke Hand. Die gespeicherten Offset-Informationen ermöglichen es, auch nach einem Neustart des Systems, genaue und konsistente Messungen zu erhalten, indem die Sensoren wieder in ihr Referenzkoordinatensystem überführt werden. Weitere Details zur Notwendigkeit und Durchführung der Handabdruckgenerierung sind in Abschnitt 8 zu finden.

4. Verwendung des Datenhandschuhs Der Datenhandschuh ist nun einsatzbereit. Die relative Orientierung der Fingersensoren zum Referenzsensor in der Handfläche wird in den folgenden Topics veröffentlicht: `/euler_L,R_1 bis 5`. Diese Daten können entweder direkt in einem externen Programm verwendet oder zur Visualisierung der Hand in RViz genutzt werden.

Um die Hand und ihre Bewegungen in RViz zu visualisieren, muss folgender Befehl ausgeführt werden:

```
roslaunch human_hand_description two_gloves.launch mapping1:=P1 mapping2:=P1
```

Damit die relativen Eulerwinkel auf die von `human_hand_description` geforderten Joint States umgerechnet werden, ist zudem das Starten des entsprechenden Programms notwendig:

```
rosrun joint_manipulator joint_state_handler.py
```

5. Visualisierung mit Plottern Anstelle oder zusätzlich zu der Visualisierung in RViz können auch die bereitgestellten Plotter verwendet werden, um die Daten zu visualisieren:

Der Plotter `plot_angular` visualisiert die Gyroskop-Daten der einzelnen Sensoren. Diese Daten repräsentieren die Winkelgeschwindigkeit, also wie schnell und in welche Richtung sich die Sensoren drehen. Dies kann hilfreich sein, um die Dynamik der Fingerbewegungen zu analysieren. Um den `plot_angular`-Plotter zu starten, muss folgender Befehl verwendet werden:

```
rosrun imu_plotter plot_angular.py {L,R}_{0 to 5}
```

Der Plotter `plot_linear` zeigt die lineare Beschleunigung der Sensoren an. Diese Daten geben Auskunft darüber, wie sich die Finger beschleunigen oder abbremsen, was nützlich sein kann, um plötzliche Bewegungen oder Kollisionen zu erkennen. Um den `plot_linear`-Plotter zu starten, muss folgender Befehl verwendet werden:

```
rosrun imu_plotter plot_linear.py {L,R}_{0 to 5}
```

Mit dem Plotter `plot_orientation` werden die Orientierungsschätzungen visualisiert, die durch den Madgwick-Filter berechnet werden. Diese Schätzungen zeigen die absolute Ausrichtung der Sensoren im Raum und helfen dabei, die genaue Position und Bewegung der Finger zu bestimmen. Um den `plot_orientation`-Plotter zu starten, muss folgender Befehl verwendet werden:

```
rosrun imu_plotter plot_orientation.py {L,R}_{0 to 5}
```

Der Plotter `plot_euler` visualisiert die relativen Winkel zwischen dem Referenzsensor (Sensor 0) und den jeweiligen anderen Sensoren. Diese Eulerwinkel sind besonders nützlich, um die relative Bewegung der Finger im Vergleich zur Handfläche zu analysieren. Um den `plot_euler`-Plotter zu starten, muss folgender Befehl verwendet werden:

```
rosrun imu_plotter plot_euler.py {L,R}_{1 to 5}
```

7.2 Beschreibung der Klasse `IMUEulerPublisher`

Die Klasse `IMUEulerPublisher` ist darauf ausgelegt, die von IMU-Sensoren erfassten Rotationsdaten in Euler-Winkel umzuwandeln und zu publizieren. Sie dient dazu, die durch den Madgwick-Filter erzeugten Rotationsdaten zu verarbeiten und die berechneten relativen Euler-Winkel zu veröffentlichen. Diese Klasse ist ein wesentlicher Bestandteil des Systems zur Erfassung und Verarbeitung von Handbewegungen und ist in das ROS-Framework integriert, wodurch mittels Topics eine einfache Kommunikation mit anderen Klassen ermöglicht wird.

Der Konstruktor `__init__()` initialisiert den ROS-Knoten und die notwendigen Attribute der Klasse:

- `sensor_orientations`: `dict` - Speichert aktuelle Orientierungen der Sensoren.
- `rotation_buffers`: `dict` - Puffer zum Speichern der initialen Rotationen.
- `average_initial_rotation`: `dict` - Durchschnitt der ersten 100 Rotationen.
- `buffers_filled`: `dict` - Flags zur Anzeige gefüllter Puffer.
- `sensor_misalignment`: `dict` - Misalignment-Daten der Sensoren.
- `publishers`: `dict` - ROS-Publisher zur Veröffentlichung der Euler-Winkel.
- `subscribers`: `list` - ROS-Subscriber zur Entgegennahme der IMU-Daten.

Methoden der Klasse

`load_sensor_misalignment()` Lädt die Kalibrierungsdaten für die Sensoren aus JSON-Dateien und speichert sie als Rotationsobjekte.

`sensor_callback(msg, sensor_id)` Callback-Funktion für die ROS-Subscriber. Diese Funktion verarbeitet die empfangenen IMU-Daten und aktualisiert die Sensororientierungen.

`update_all_angles(hand)` Aktualisiert die Euler-Winkel aller Sensoren einer Hand, wenn der Referenzsensor (Sensor 0) aktualisiert wird.

`average_rotations(rotations)` Berechnet die Durchschnittsrotation aus einer Liste von Rotationen. Diese Methode verwendet die Eigenwertzerlegung, um das durchschnittliche Quaternion zu berechnen.

`transform_to_reference_frame(rotation, rotation_initial)` Diese Methode transformiert eine gegebene Rotation relativ zu einer anfänglichen Referenzrotation. Dies ist wichtig, um sicherzustellen, dass die Rotationsdaten konsistent in Bezug auf eine festgelegte Startposition sind.

```
def transform_to_reference_frame(self, rotation, rotation_initial):
# Transform to reference frame
transformed_rotation = rotation_initial.inv() * rotation
return transformed_rotation
```

`calculate_relative_angles_quaternion(rotation_hand, rotation_finger, rotation_han`
`rotation_finger_initial)` Diese Methode berechnet die relativen Euler-Winkel zwischen zwei Rotationen, indem die Rotationen relativ zu ihren initialen Referenzrahmen transformiert werden. Dies ermöglicht es, die relative Bewegung zwischen zwei Sensoren zu bestimmen.

```
def calculate_relative_angles_quaternion(self, rotation_hand, rotation_finger,
    rotation_hand_initial, rotation_finger_initial):
# Transform rotations to their initial reference frames
rotation_hand_transformed = self.transform_to_reference_frame(rotation_hand,
    rotation_hand_initial)
rotation_finger_transformed = self.transform_to_reference_frame(rotation_finger,
    rotation_finger_initial)

# Calculate the relative rotation
relative_rotation = rotation_hand_transformed.inv() * rotation_finger_transformed

# Convert the relative rotation to Euler angles
relative_euler_angles = relative_rotation.as_euler('xyz', degrees=True)

# Return the relative Euler angles
return relative_euler_angles
```

`publish_angles(hand, sensor_index)` Diese Methode veröffentlicht die berechneten Euler-Winkel für einen spezifischen Sensor. Hierbei werden die Sensororientierungen und die Misalignment-Daten verwendet, um die korrekten Winkel zu berechnen und über ROS zu publizieren.

```
def publish_angles(self, hand, sensor_index):
# Apply the sensor misalignment
q_finger_reference = self.sensor_misalignment[hand][sensor_index-1]

rotation_hand = self.sensor_orientations[hand][0]
rotation_hand_initial = self.average_initial_rotation[hand][0]

rotation_finger = self.sensor_orientations[hand][sensor_index] * q_finger_reference.
    inv()
rotation_finger_initial = self.average_initial_rotation[hand][sensor_index] *
    q_finger_reference.inv()

publisher = self.publishers[hand][sensor_index - 1]

# Ensure publisher connection
if publisher.get_num_connections() > 0:
euler_angles = self.calculate_relative_angles_quaternion(rotation_hand,
    rotation_finger,
rotation_hand_initial, rotation_finger_initial)

# Publish the Euler angles
msg = Float64MultiArray()
msg.data = euler_angles.tolist()
publisher.publish(msg)
```

7.3 Haptisches Feedback

Das haptische Feedback wird über das Pub/Sub-System von ROS gesteuert und an die Arduino-Boards der Handschuhe über UDP gesendet. Wenn ein Befehl auf die ROS-Topics `/imu/L_Hand/tactile_feedback` für die linke Hand und `/imu/R_Hand/tactile_feedback` für die rechte Hand veröffentlicht wird, übernimmt die ROS-Node `udp_command_sender_node.py` die Aufgabe, diese Daten zu empfangen und über UDP an die jeweiligen Arduino-Boards weiterzuleiten.

Die Nachrichten, die auf den genannten Topics veröffentlicht werden, enthalten Steuerbefehle in Form von Arrays, die die Zustände der Vibrationsmotoren beschreiben. Ein Wert von ‘0‘ bedeutet ausgeschaltet, ‘1‘ für schwache Vibrationen, ‘2‘ für mittlere Vibrationen und ‘3‘ für starke Vibrationen. Zum Beispiel:

$$[0, 0, 0, 0, 0, 1, 2, 3]$$

In diesem Beispiel würden die letzten drei Motoren der Handschuhe mit zunehmender Intensität vibrieren, während die ersten fünf ausgeschaltet sind.

Die Node `udp_command_sender_node.py` wird automatisch gestartet, wenn der ROS-Befehl `roslaunch udp_datatransfer start_application.launch` ausgeführt wird. Sobald die Node läuft, empfängt sie die auf den genannten Topics veröffentlichten Nachrichten und sendet diese direkt an die Arduino-Boards. Dadurch wird es möglich, das haptische Feedback in Echtzeit zu steuern und den Benutzern ein präzises taktiles Feedback während der Interaktion mit dem System zu bieten.

8 Evaluation der Datenqualität

Dieses Kapitel beschäftigt sich mit der Untersuchung der Qualität der erfassten Sensordaten und der Genauigkeit der verwendeten Sensoren im Datenhandschuh. Die verschiedenen Unterkapitel analysieren mehrere Schlüsselparameter, die die Leistungsfähigkeit und Zuverlässigkeit des Systems beeinflussen. Im Fokus stehen dabei die Datenerhebungsfrequenz, die Sensordrift im Ruhezustand sowie während des Betriebs der Vibrationsmotoren und die relative Abweichung der Eulerwinkel bei Handbewegungen. Ziel dieser Evaluation ist es, die Stabilität und Präzision der erfassten Bewegungsdaten zu überprüfen und potenzielle Fehlerquellen zu identifizieren, die die Systemleistung beeinträchtigen könnten. Zudem wird untersucht, wie unterschiedliche Faktoren wie Kalibrierung und externe Störungen die Sensorleistung und die Qualität der Messwerte beeinflussen.

Frequenz der Datenerhebung Die Frequenz, mit der Daten von den Sensoren erfasst und verarbeitet werden, ist ein kritischer Faktor für die Genauigkeit und Reaktionsfähigkeit des Systems. In diesem Abschnitt wird die tatsächliche Datenerhebungsfrequenz gemessen und mit den spezifizierten Werten verglichen. Eine hohe Frequenz ist erforderlich, um schnelle Bewegungen präzise zu erfassen und Verzögerungen zu minimieren.

In der Bachelorarbeit von Can Bagdas wird angegeben, dass die Datenerhebungsfrequenz des BMI160 signifikant niedrieger sei 5.1. Allerdings stellte sich während meiner Arbeit heraus, dass die ausgewählte Baudrate hier wahrscheinlich der Flaschenhals war.

Bei der Analyse der beiden Handschuhe wurde festgestellt, dass der linke Handschuh über ein funktionierendes Wi-Fi-Modul verfügte, während das Modul des rechten Handschuhs defekt war. Es ist anzunehmen, dass Herr Bagdas beim linken Handschuh die Wi-Fi-Methode verwendet hat, um Daten auf den Rechner zu übertragen, und beim rechten Handschuh die serielle Verbindung. Die serielle Verbindung erreichte lediglich etwa 30 Hz, was direkt auf die Konfiguration der Baudrate zurückzuführen war.

Nach dem Austausch des Mikrocontrollers im rechten Handschuh und der korrekten Einstellung der Übertragungsparameter konnte ich die Datenübertragungsfrequenz für beide Handschuhe angleichen. Beide Handschuhe senden nun mit einer Frequenz von ca. 60 Hz über Wi-Fi und etwa 100 Hz über die serielle Verbindung. Diese Frequenzen sind für die präzise Erfassung schneller Handbewegungen wesentlich, da sie eine reduzierte Latenz und verbesserte Reaktionszeiten gewährleisten. Die Anpassungen der Übertragungsparameter haben damit signifikant zur Verbesserung der Systemleistung beigetragen.

Drift der verschiedenen Sensoren ohne externe Einflüsse Die Drift der Sensoren beschreibt die Abweichung ihrer Messwerte über die Zeit, selbst wenn keine externen Bewegungen oder Einflüsse vorhanden sind. Hier wird untersucht, wie stabil die Sensoren des Datenhandschuhs sind, wenn sie in einer ruhigen Umgebung ohne äußere Einflüsse betrieben werden. Langzeitmessungen werden durchgeführt, um die Drift über verschiedene Zeiträume zu analysieren.

Im folgenden Plot eines Sensors des Typs MPU9250 bleiben Roll und Pitch durch den stabilisierenden Einfluss der Gravitation relativ konstant, jedoch zeigt sich, dass der Gierwinkel über einen Zeitraum von 300 Sekunden um etwa 527° driftet.

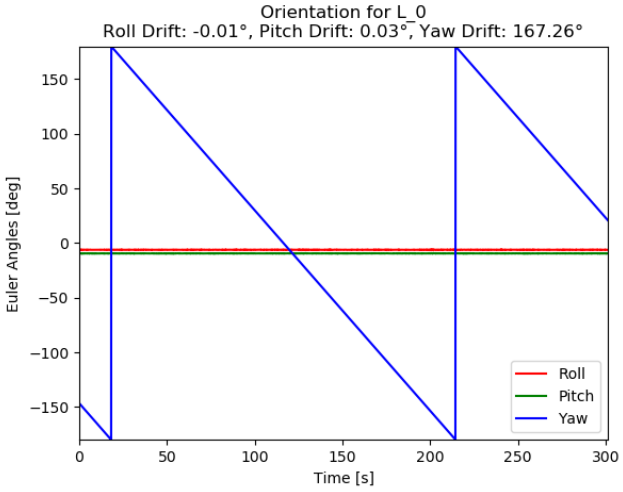

Abbildung 14: Drift der Sensoren ohne externe Einflüsse - MPU9250 ohne Ferraris-Kalibrierung

Dieser Drift ist auf das Rauschen und die Eigenheiten des Gyroskopsensors zurückzuführen. Die Nutzung unkalibrierter Daten reicht daher selbst bei Verwendung des Madgwick-Filters, der Orientierungsdaten berechnet, nicht aus, um stabile Ergebnisse zu erzielen. Daher ist die Anwendung einer Kalibrierung, wie z.B. der Ferraris-Kalibrierung (siehe Kapitel Theoretische Grundlagen), notwendig. Der nächste Plot zeigt die Drift desselben Sensors nach Anwendung dieser Kalibrierung.

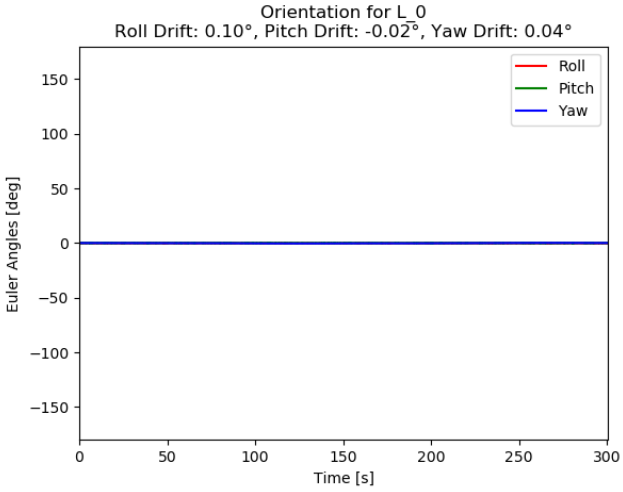

Abbildung 15: Drift der Sensoren ohne externe Einflüsse - MPU9250 mit überdurchschnittlich guter Ferraris-Kalibrierung

Für diese Messung wurde eine zufällig überdurchschnittlich präzise Kalibrierung verwendet, um die Wirksamkeit der Driftkompensation zu verdeutlichen. Im selben Zeitraum beträgt die Drift des Gierwinkels hier nur 0,04°, was für unseren Anwendungsfall vernachlässigbar ist. Leider lässt sich ein so gutes Ergebnis im praktischen Einsatz nicht dauerhaft erreichen. Ein realistisch erreichbares Ergebnis zeigt der folgende Plot eines Sensors des Typs MPU6050. Über denselben Zeitraum von 300 Sekunden driftet der Gierwinkel hier um etwa 3,06°.

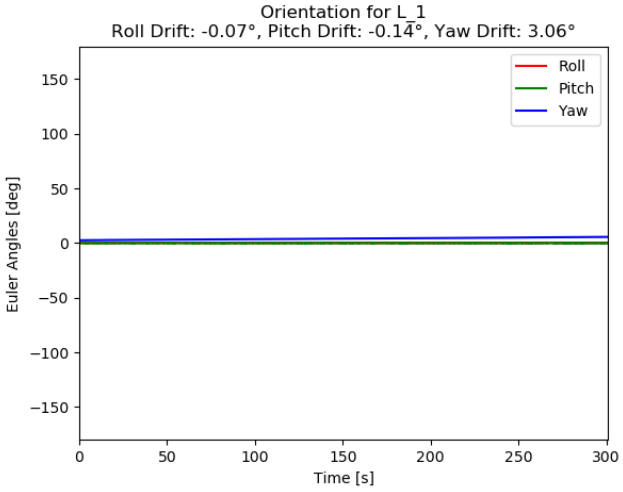

Abbildung 16: Drift der Sensoren ohne externe Einflüsse - MPU6050 mit Ferraris-Kalibrierung

Drift während die Motoren laufen Der Betrieb der Vibrationsmotoren kann zusätzliche Einflüsse auf die Sensoren haben und deren Drift verstärken. In diesem Abschnitt wird untersucht, wie sich die Drift der Sensoren verändert, wenn die Vibrationsmotoren aktiv sind. Die Ergebnisse werden mit den Messungen ohne Motoraktivität verglichen, um den Einfluss der Motoren auf die Sensordaten zu quantifizieren. Das zugrunde liegende Problem wird in Abbildung 17 und Abbildung 18 veranschaulicht. Der Zeitpunkt, zu dem der Vibrationsmotor aktiviert wurde, ist sowohl in den linearen Beschleunigungsdaten als auch in den Werten der Winkelgeschwindigkeit deutlich erkennbar.

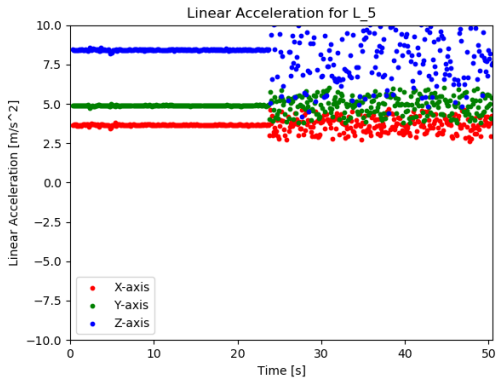

Abbildung 17: Rohdaten der Beschleunigungssensoren + Einfluss von Vibrationsmotoren

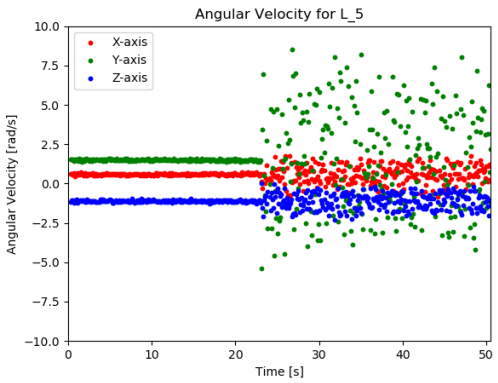

Abbildung 18: Rohdaten der Gyroskope + Einfluss von Vibrationsmotoren

Beide Sensortypen zeigen eine ausgeprägte Wechselwirkung mit den in der Nähe befindlichen Vibrationsmotoren. Wenn dieses Rauschen trotz der Anwendung der Kalibrierung und des Madgwick-Filters zu einer signifikanten Sensordrift führt, wäre eine Kombination aus IMUs und Vibrationsmotoren in Datenhandschuhen in dieser Form nicht praktikabel. In Abbildung 19 wird die Drift der Sensoren ohne aktive Motoren dargestellt, während Abbildung 20 die Drift bei laufenden Motoren zeigt.

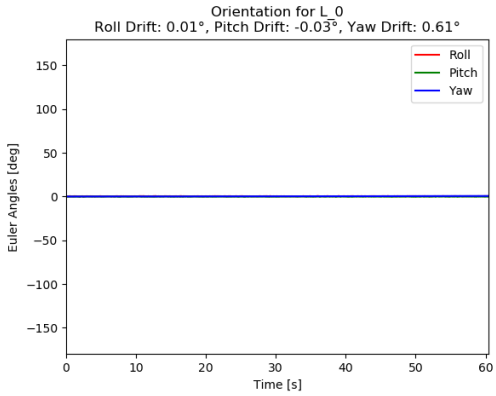

Abbildung 19: Drift eines MPU9250 ohne aktive Motoren

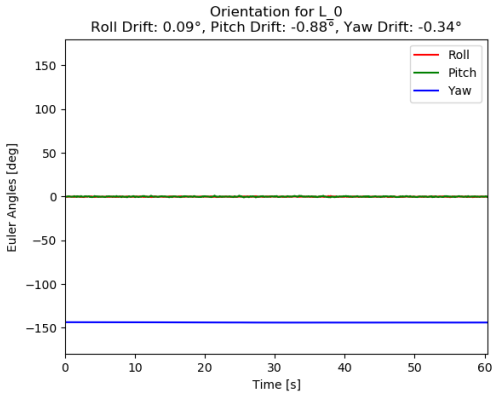

Abbildung 20: Drift eines MPU9250 während aktive Motoren laufen

Beide Plots verdeutlichen, dass der Einfluss der Vibrationsmotoren auf die Orientierungsdaten minimal ist. Die Drift ist in diesem speziellen Fall bei laufenden Motoren sogar leicht geringer, was jedoch als Varianz innerhalb der Messgenauigkeit betrachtet werden kann. Dieses erfreuliche Ergebnis ist vermutlich der Robustheit des Madgwick-Filters gegenüber Messrauschen zu verdanken. In Abbildung 21 sind die Beschleunigung, die Winkelgeschwindigkeit und die Orientierung eines Sensors des Typs MPU6050 im gleichen Zeitraum dargestellt. Etwa 25 Sekunden nach Beginn der Aufzeichnung wurde der Vibrationsmotor aktiviert. Es ist zu erkennen, dass die Vibrationen auch in diesem Fall nur einen vernachlässigbaren Einfluss auf die vom Madgwick-Filter berechnete Orientierung haben.

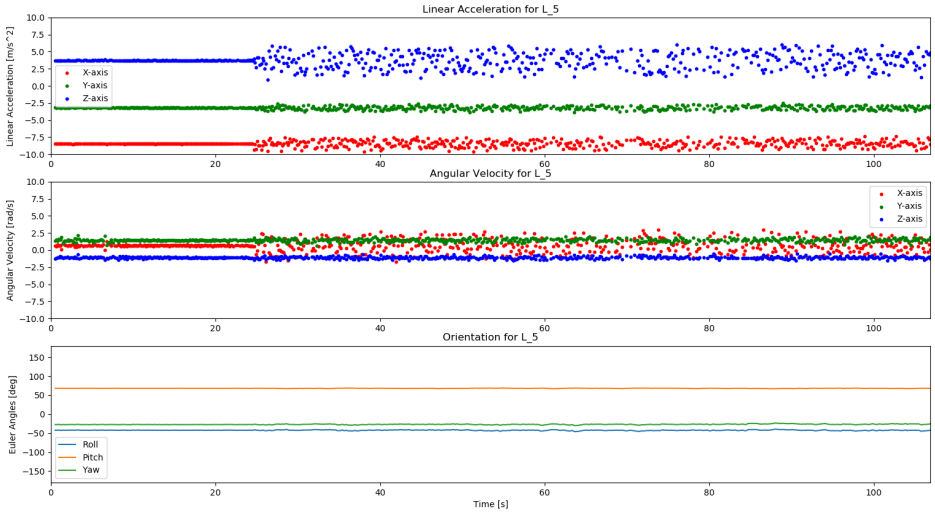

Abbildung 21: Drift eines MPU6050 mit laufendem Motor

Relative Eulerwinkel-Abweichung Die Abweichung der relativen Eulerwinkel bei Bewegungen gibt Aufschluss darüber, wie genau die Sensoren die Hand- und Fingerbewegungen erfassen. In dieser Analyse wird die Genauigkeit der berechneten Eulerwinkel während verschiedener Bewegungsabläufe untersucht. Da die Sensoren bei dieser Analyse fest auf einer Platte montiert sind und stets die gleiche Bewegung ausführen, sollte die Abweichung in der relativen Orientierung konstant und idealerweise nahe null liegen.

Es werden zwei Szenarien betrachtet, die die relativen Winkel eines Referenzsensors in der Handfläche im Vergleich zu einem Fingersensor darstellen.

Die Plots Abbildung 22 und Abbildung 23 zeigen das Verhalten der Sensoren direkt nach der Kalibrierung sowie nachdem ein Sensor nach der Kalibrierung an einer neuen Position auf der Platte angebracht wurde.

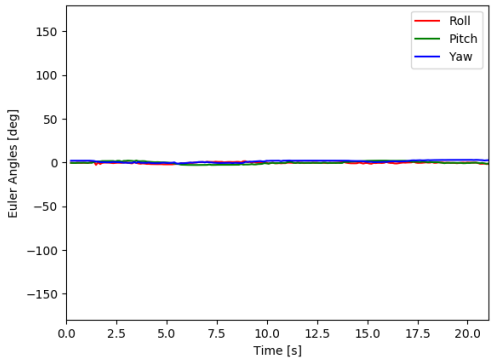

Abbildung 22: Relative Eulerwinkel-Abweichung direkt nach der Kalibrierung

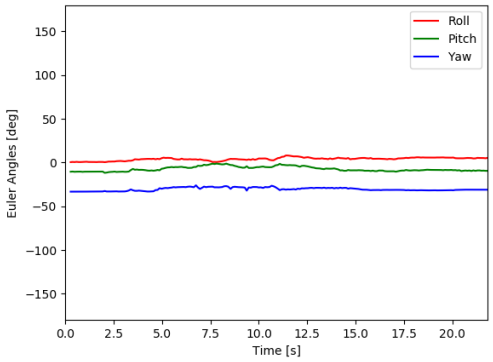

Abbildung 23: Relative Eulerwinkel-Abweichung nach Versatz eines Sensors

Klar ersichtlich ist, dass die relativen Eulerwinkel beim gleichmäßigen Bewegen beider Sensoren, wie gewünscht, vergleichsweise stabil bleiben. Die Abweichungen liegen nur im Bereich von wenigen Grad, sowohl direkt nach der Kalibrierung als auch nachdem einer der Sensoren relativ zum anderen bewegt wurde. Dies zeigt, dass die Position der Finger relativ zum Referenzsensor in diesen Fällen gut mit der von der Kalibrierung abhängigen Genauigkeit berechnet werden kann.

Die Plots Abbildung 24 und Abbildung 25 zeigen das Verhalten der Sensoren nach einem Neustart des Systems, einmal ohne und einmal mit Verwendung der Offset-Informationen.

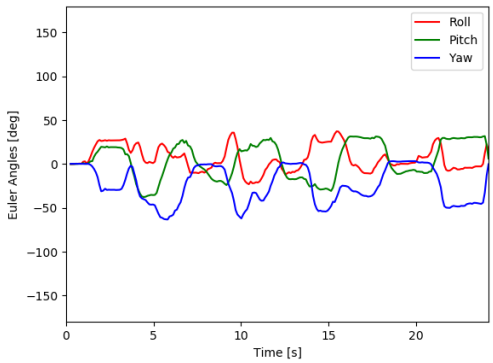

Abbildung 24: Relative Eulerwinkel-Abweichung nach Systemneustart ohne Sensor-Offset

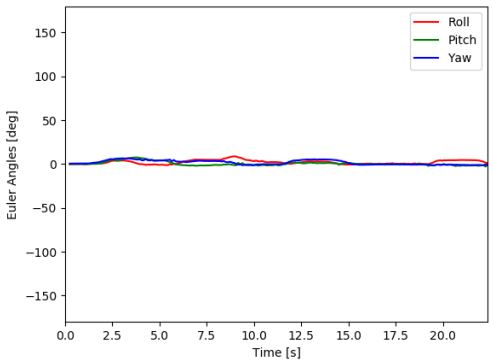

Abbildung 25: Relative Eulerwinkel-Abweichung nach Systemneustart mit Sensor-Offset

Im Gegensatz zum Verhalten ohne Systemneustart zeigt der Plot Abbildung 24 signifikante Veränderungen in den Eulerwinkeln. Da die Fingersensoren, sowohl der MPU6050 als auch der BMI160, keine Magnetsensoren enthalten, ist eine präzise Bestimmung des Gierwinkels nicht möglich. Wenn die Sensoren bewegt werden, ohne die relativen Ausrichtungen mithilfe der Sensor-Offset-Daten zu berücksichtigen, resultiert dies in erheblichen Abweichungen bei der Berechnung der relativen Winkel, da der Fingersensor nicht in sein Referenz-Koordinatensystem überführt wird.

Das Überführen der Sensoren in ihr Referenzsystem führt jedoch zu Ergebnissen, die denen direkt nach der Kalibrierung ähneln (siehe Abbildung Abbildung 25).

Diese Ergebnisse verdeutlichen, dass zur langfristigen Nutzung der Handschuhe nach einer Kalibrierung der Handschuh zügig angezogen werden sollte, um eine signifikante Sensordrift zu vermeiden. Anschließend sollte umgehend der folgende Befehl in der Kommandozeile ausgeführt werden:

```
rosrun joint_manipulator calculate_alignment_offset.py
```

Damit wird ein "Handabdruck", also die relative Orientierung der Sensoren an der Hand, gespeichert. So bleiben diese Informationen auch nach einem Systemneustart erhalten.

9 Visualisierung der Daten

In diesem Kapitel wird die Visualisierung der erfassten Sensordaten behandelt. Es werden verschiedene Methoden vorgestellt, um die Bewegungsdaten des Datenhandschuhs anschaulich darzustellen. Dies umfasst sowohl die Verwendung von Plots, die detaillierte Einblicke in die Orientierungen der Sensoren bieten, als auch die Anwendung von Rviz zur Visualisierung der Hand- und Fingerbewegungen in einer 3D-Umgebung. Ziel dieser Visualisierungen ist es, die erfassten Daten verständlich und intuitiv darzustellen, sodass die Benutzer sowohl die allgemeine Bewegung als auch feinere Details der Handbewegungen nachverfolgen können.

9.1 Visualisierung über Plots

Abbildung 26 zeigt einen Plot, der die Orientierungen aller Sensoren während eines bestimmten Zeitraums darstellt. In diesem Experiment wurden beide Hände unabhängig voneinander bewegt, um die Fähigkeit des Systems zu demonstrieren, Daten von allen Sensoren gleichzeitig zu empfangen und zu verarbeiten. Die Plot-Darstellung verdeutlicht, wie die unterschiedlichen Sensoren auf die Bewegungen reagieren und welche Datenmuster dabei entstehen.

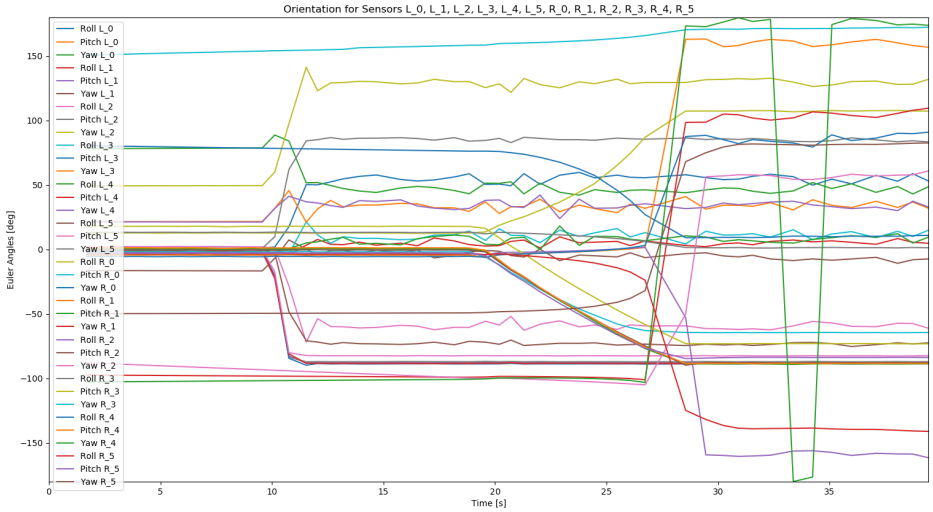

Abbildung 26: Orientierungen aller Sensoren über den gleichen Zeitraum

Obwohl diese Art der Darstellung detaillierte Informationen liefert, ist sie für den menschlichen Betrachter kaum intuitiv. Dieses Problem wird im folgenden Kapitel adressiert, wo eine Lösung zur Verbesserung der Verständlichkeit und Benutzerfreundlichkeit vorgestellt wird.

9.2 Visualisierung in Rviz

Die Handdarstellung erfolgt unter Verwendung der Bibliothek aus dem ”Human Hand” GitHub Repository [23], welche auf Rviz basiert. Durch die Nutzung dieser Bibliothek können die Bewegungen der Hand und Finger visualisiert werden. Abbildung 27 zeigt die Hand in der Ausgangsposition. In Abbildung 28 wurde der Zeigefinger gebeugt, indem der Sensor des Datenhandschuhs bewegt wurde. In Unterabschnitt A.2 ist ein kurzes Video, welches die rechte Hand in Bewegung zeigt.

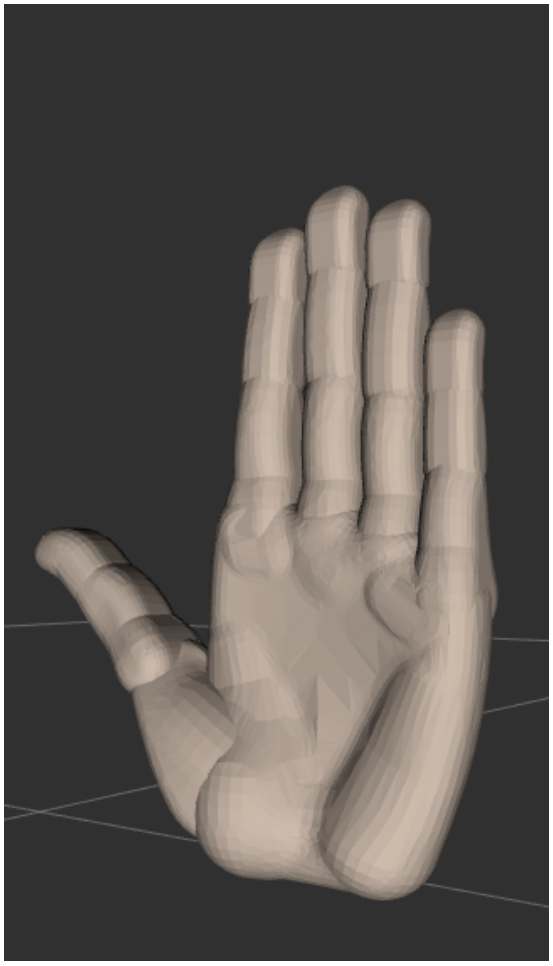

Abbildung 27: Hand in Startposition

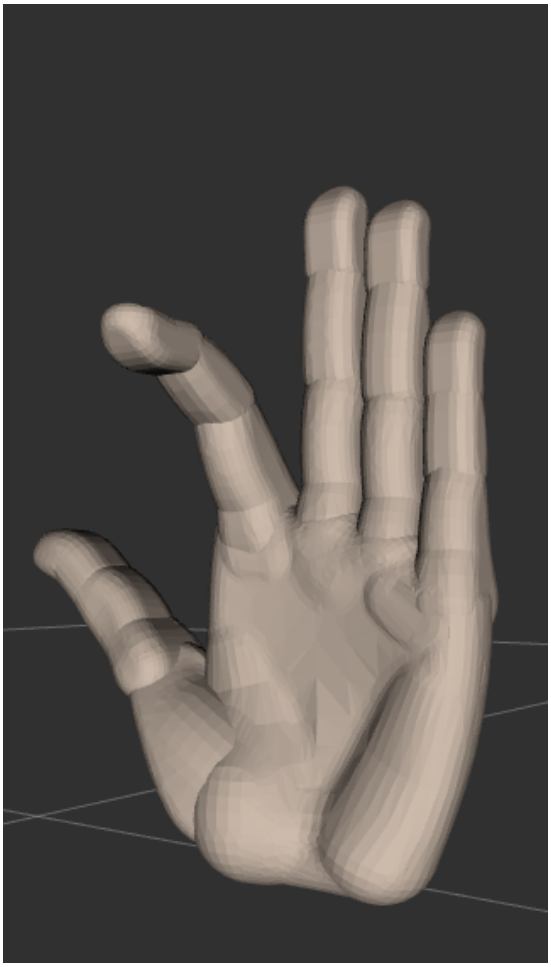

Abbildung 28: Hand mit gebeugtem Finger

Abbildung 25 zeigt, dass bei richtiger Kalibrierung und einer korrekten Generierung der Sensorfehlanpassungen die Genauigkeit für relativ grobe Tätigkeiten ausreichend ist. Werden feinmotorische Tätigkeiten ausgeübt, sind die in Abbildung 25 aufgezeigten Ungenauigkeiten jedoch ein großes Problem, was insbesondere das Abduktionsgelenk des Fingers betrifft. Die Beweglichkeit des Fingers in diese Richtung ist natürlicherweise stark eingeschränkt und die Abweichungen treten dementsprechend stärker in Erscheinung, was oftmals zu unnatürlichen Fingerstellungen in der Visualisierung der Hand führt. Standardmäßig wird das Abduktionsgelenk deshalb innerhalb des Programms nicht berechnet. Eine Änderung ist leicht möglich, indem man die auskommentierte Zeile 95 innerhalb der Klasse "JointStateHandler"wieder aktiviert.

```
# Uncomment if you use the abduction joint
# joint_state_msg.position[self.joint_names.index(f'{finger}_abduction_joint')] =
```

10 Zusammenfassung

In dieser Bachelorarbeit wurde ein Datenhandschuh entwickelt und evaluiert, dessen Hauptziel es war, präzise und zuverlässige Bewegungsdaten zu erfassen, um eine präzise Nachverfolgung von Hand- und Fingerbewegungen zu ermöglichen. Der Datenhandschuh nutzt eine Kombination aus Inertial Measurement Units (IMUs) und Vibrationsmotoren, um sowohl grobe als auch feine Handbewegungen zu detektieren und zu visualisieren.

Die Arbeiten von Can Bagdas bildeten die Grundlage für den in dieser Arbeit entwickelten Prototypen. Seine Bachelorarbeit lieferte nicht nur die architektonische Basis des Systems, sondern auch wertvolle Erkenntnisse bezüglich der Sensorauswahl und der Systemarchitektur. Diese solide Grundlage ermöglichte die Weiterentwicklung, die insbesondere die Integration von Visualisierungsfunktionen in ROS sowie die Einführung von haptischem Feedback umfasste.

Die Evaluation der Datenqualität zeigte, dass die Hauptfaktoren, die die Leistung des Systems beeinflussen die Datenerhebungsfrequenz, sowie die Drift der Sensoren sind. Auch die Auswirkungen von Vibrationsmotoren auf die Sensoren wurden untersucht, stellten sich jedoch als minimal heraus. Die Datenerhebungsfrequenz konnte erfolgreich optimiert werden, sodass eine Frequenz von ca. 60 Hz über Wi-Fi und 100 Hz über serielle Verbindung erreicht wurde, was die Reaktionszeit des Systems signifikant verbesserte.Die Drift der Sensoren wurde durch die Anwendung einer Kalibrierung, insbesondere der Ferraris-Kalibrierung, drastisch reduziert, was zu einer deutlichen Verbesserung der Präzision führte.

Zusätzlich wurde ein Visualisierungskonzept entwickelt, das sowohl die Darstellung der Bewegungsdaten in Form von Plots als auch die 3D-Visualisierung der Hand- und Fingerbewegungen in Rviz umfasst. Die Visualisierung in Rviz ermöglichte es, die Handbewegungen anschaulich darzustellen, wobei die Genauigkeit für grobe Bewegungen zufriedenstellend war. Feine Handbewegungen, insbesondere bei Fingerabduktionen, wiesen jedoch zu große Abweichungen auf, was auf die begrenzte Sensorgenauigkeit zurüchzuführen ist.

Abschließend lässt sich festhalten, dass der entwickelte Datenhandschuh einen funktionalen, vergleichsweise kostengünstigen und Open-Source basierten Prototypen zur Erfassung von Hand- und Fingerbewegungen darstellt. Die durchgeführten Tests und die anschließende Analyse der Datenqualität und Visualisierung haben gezeigt, dass das System in der Lage ist, zuverlässige Bewegungsdaten zu liefern, jedoch noch Potenzial für Verbesserungen besteht, insbesondere im Bereich der feinmotorischen Bewegungen und der Langzeitnutzung.

A Anhang

A.1 Schaltplan des linken Handschuhs

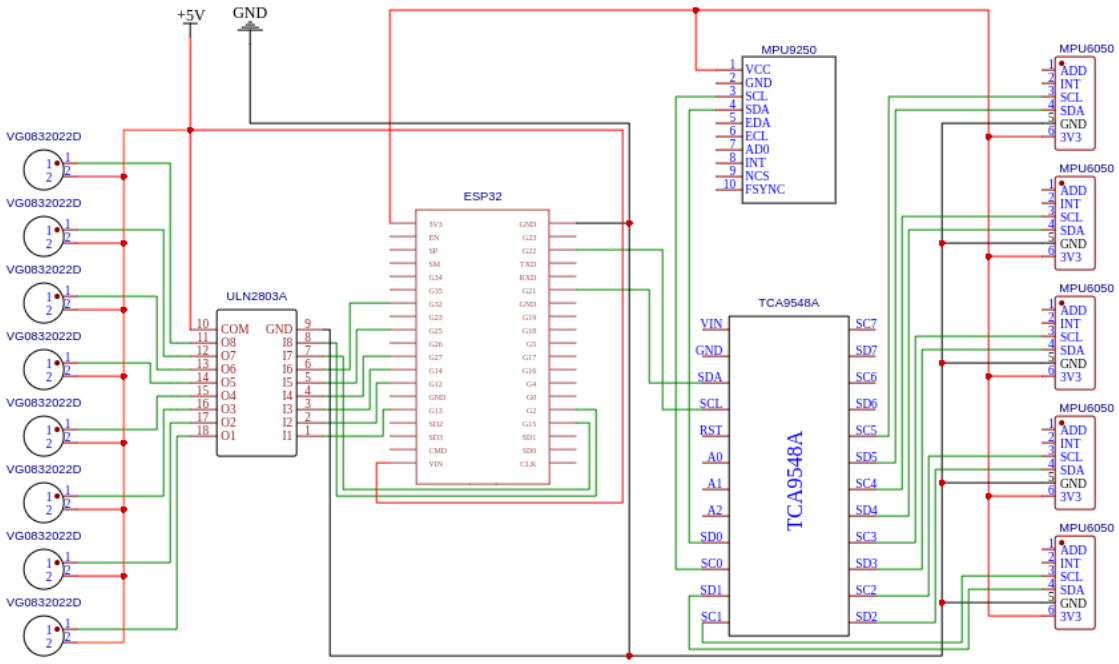

Abbildung 29: Schaltplan des liken Handschuhs

A.2 Video

Abbildungsverzeichnis

Literatur

[1] Advanced Navigation, “Inertial Measurement Unit (IMU) - An Introduction,” [Online]. Available: https://www.advancednavigation.com/tech-articles/inertial-measurement-unit-imu-an-introduction/. [Zuletzt aufgerufen: 13-Nov-2024].

[2] Martin Grunwald und Lothar Beyer, “Der bewegte Sinn: Grundlagen und Anwendungen zur haptischen Wahrnehmung,” Huber Verlag, 2001.

[3] “rviz - ROS visualization tool,” *GitHub repository*, [Online]. Available: https://github.com/ros-visualization/rviz. [Zuletzt aufgerufen: 13-Nov-2024].

[4] “RViz - ROS Wiki,” *ROS Wiki*, [Online]. Available: http://wiki.ros.org/rviz. [Zuletzt aufgerufen: 13-Nov-2024].

[5] Madgwick, S.O.H., “An Efficient Orientation Filter for Inertial and Inertial/Magnetic Sensor Arrays,” 2010.

[6] OlliW, “IMU Data Fusing: Complementary, Kalman, and Mahony Filter,” [Online]. Available: https://www.olliw.eu/2013/imu-data-fusing/. [Zuletzt aufgerufen: 13-Nov-2024].

[7] “Ferraris Calibration Guide,” *IMUcal Documentation*, [Online]. Available: https://imucal.readthedocs.io/en/latest/guides/ferraris_guide.html. [Zuletzt aufgerufen: 13-Nov-2024].

[8] E. Lange, “Optische Messtechnik,” in *Industrielle Messtechnik*, 2. Auflage, Springer-Verlag, 2011, pp.

[9] O. Woodman, *An introduction to inertial navigation*, University of Cambridge, Computer Laboratory, Technical Report, 2007.

[10] R. Kuc, “Ultrasound: Sensing, Imaging, and Signal Processing,” John Wiley & Sons, 1992.

[11] E. Vezzoli, “Review of Hybrid Motion Capture Systems: Optical and Inertial Sensor Fusion,” *Sensors*, vol. 20, no. 22, pp. 6441, 2020.

[12] K. J. Kuchenbecker, “Haptics,” in *Springer Handbook of Robotics*, Springer, 2009, pp. 719-739.

[13] SenseGlove, “SenseGlove,” [Online]. Available: https://www.senseglove.com/. [Zuletzt aufgerufen: 13-Nov-2024].

[14] Bagdas, C., “Entwicklung eines Prototypen für einen Datenhandschuh basierend auf IMUs - Version 2,” Universität Hamburg, 2022.

[15] TDK InvenSense, “MPU-9050 Product Specification,” Revision 1.1, 2016. [Online]. Available: https://invensense.tdk.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf. [Zuletzt aufgerufen: 13-Nov-2024].

[16] TDK InvenSense, “MPU-6050 Product Specification,” Revision 3.4, 2013. [Online]. Available: https://www.alldatasheet.com/datasheet-pdf/pdf/1132807/TDK/MPU-6050.html. [Zuletzt aufgerufen: 13-Nov-2024].

[17] Bosch Sensortec, “BMI160 Product Specification,” Revision 1.2, 2018. [Online]. Available: https://www.bosch-sensortec.com/products/motion-sensors/imus/bmi160/. [Zuletzt aufgerufen: 13-Nov-2024].

[18] Texas Instruments, “TCA9548A Low-Voltage 8-Channel I2C Switch with Reset,” [Online]. Available: https://www.ti.com/lit/ds/symlink/tca9548a.pdf, 2024. [Zuletzt aufgerufen: 13-Nov-2024].

[19] Espressif Systems, “ESP32 Series Datasheet,” [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. [Zuletzt aufgerufen: 13-Nov-2024].

[20] Vybronics, “VG0832022D Vibration Motor Datasheet,” [Online]. Available: https://www.vybronics.com/wp-content/uploads/datasheet-files/Vybronics-VG0832022D-datasheet.pdf. [Zuletzt aufgerufen: 13-Nov-2024].

[21] Vybronics, “LRA Coin Vibration Motor - VG0832013D,” [Online]. Available: https://www.vybronics.com/coin-vibration-motors/lra/v-g0832013d, 2024. [Zuletzt aufgerufen: 13-Nov-2024].

[22] Texas Instruments, “ULN2803A Darlington Transistor Arrays Datasheet,” [Online]. Available: https://www.ti.com/lit/ds/symlink/uln2803a.pdf. [Zuletzt aufgerufen: 13-Nov-2024].

[23] “Human Hand - GitHub Repository,” *GitHub*, [Online]. Available: https://github.com/ubi-agni/human_hand. [Zuletzt aufgerufen: 13-Nov-2024].

Eidesstattliche Erklärung

„Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Bachelorstudiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe.

Ort, Datum

Hamburg, 13.11.2024

Unterschrift