Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

BACHELORTHESIS

Prototyp für eine virtuelle Tastatur basierend auf IMUs und maschinellem Lernen - Systementwurf

vorgelegt von

Paul Bienkowski

MIN-Fakultät

Fachbereich Informatik

Arbeitsbereich Technische Aspekte Multimodaler Systeme

Studiengang: Informatik

Matrikelnummer: 6415133

Erstgutachter: Dr. Norman Hendrich

Zweitgutachter: Florens Wasserfall

Kurzbeschreibung

Mit dem Projekt „Prototyp für eine virtuelle Tastatur basierend auf IMUs und maschinellem Lernen" entwickeln wir ein System, welches eine Alternative zur klassischen Tastatur als Texteingabemethode für Computersysteme darstellt, indem es die Bewegungen der Hand beim Tippen erfasst und mithilfe maschinellen Lernens wiedererkennt. In dieser Arbeit befasse ich mich mit dem Design dieses Systems, ausgeschlossen hiervon ist das maschinelle Lernen. Weiterhin zeige ich, wie die gesamte Umsetzung der Hardware sowie die Architektur der Software diese Anwendung ermöglicht. Die Befestigung von IMUs (*inertial measurement unit*) an der Hand mittels eines „Datenhandschuhs" erwies sich für diese Anwendung als sehr geeignet. Ich bewerte das Gesamtsystem und komme zu dem Schluss, dass unsere Vision umsetzbar erscheint und das vorgestellte Systemdesign einen geeigneten Weg zu diesem Ziel darstellt.

Abstract

In our project „Prototype for a virtual keyboard based on IMUs and machine learning" we develop an alternative text input system for computers, capable of replacing a traditional computer keyboard, which records movements made by the hand while typing and recognizes these using a machine learning algorithm. In this work I cover the design of this system, excluding the machine learning part. I show the actual realization of the complete hardware as well as the software architecture supporting the system and evaluate the complete system. I consider the attachment of IMUs (inertial measurement units) to the hand using a „data glove" to be suitable for this task. I conclude that our vision appears feasible, and that the presented system design provides the necessary means to reach our goal.

Inhaltsverzeichnis

Abkürzungsverzeichnis

3D	3-dimensional
AR	*augmented reality*, erweiterte Realität
CNN	*convolutional neural network*, Faltungsnetz
DoF	*degrees of freedom*, Freiheitsgrade
EEPROM	*electrically erasable programmable read-only memory* (Speicherbaustein)
EMG	Elektromyografie (Messung elektrischer Muskelaktivität)
GND	*ground*, Masse, Bezugspotenzial
GPIO	*general-purpose input/output* (universell einsetzbarer Kontakt auf einer Leiterplatte)
HAR	*human activity recognition* (Erkennung menschlicher Aktivitäten aus Sensordaten)
I^2C	*Inter-Integrated Circuit* (serieller Datenbus)
IMU	*inertial measurement unit*, inertiale Messeinheit
IPC	*inter-process communication*, Interprozesskommunikation
MEMS	*microelectromechanical system*, mikroelektromechanisches System
ML	*machine learning*, maschinelles Lernen
ROS	*Robot Operating System*
SCL	*signal clock*, Taktleitung bei I^2C
SDA	*signal data*, Datenleitung bei I^2C
SERCOM	*serial communication interface* (in Atmel SAM D Mikrokontrollern)
PWM	*pulse-width modulation*, Pulsweitenmodulation
SMD	*surface-mount device*, oberflächenmontiertes Bauelement
SPI	*Serial Peripheral Interface* (serieller Datenbus)
UART	*Universal Asynchronous Receiver Transmitter* (Baustein zur Realisierung serieller Schnittstellen)
UDP	*User Datagram Protocol* (Transportprotokoll)
VCC	*voltage at the common collector* (Versorgungsspannung)
VR	*virtual reality*, virtuelle Realität
YAML	*YAML Ain't Markup Language* (Datenformat)

1 Einleitung

In diesem Kapitel gebe ich eine Einführung in die Thematik der vorliegenden Bachelorarbeit. Ich erläutere und motiviere die Zielsetzung des Projektes und beschreibe den Umfang und Aufbau meiner Arbeit.

1.1 Motivation

Der Computer ist allgegenwärtig, und zu ihm gehört seit jeher die Tastatur. Ursprünglich von der Schreibmaschine inspiriert, wurde sie zum primären Eingabegerät für PCs und ist nun von keinem Schreibtisch mehr wegzudenken. Auch den Übergang zu anderen Geräten, etwa Smartphones und Tablets, hat die Tastatur mitgemacht – teils in Form echter Tasten auf dem Gerät, teils virtualisiert auf einem Touchscreen.

Es gibt viele Gründe für die weitreichende Verbreitung von Tastaturen. Zunächst ist ihr Prinzip sehr einfach, somit ist ihre Bedienung leicht zu erlernen, und mit etwas Übung wird ein Benutzer auch schnell effizient in ihrer Verwendung. Außerdem ist die Herstellung einer Tastatur aufgrund der technischen Einfachheit günstig. Es kommt hinzu, dass sie sehr präzise arbeitet – drückt man die richtige Taste, kann der Computer dies mühelos interpretieren, es gibt keinen Raum für Fehler.

Allerdings haben Tastaturen viele Nachteile. Allen voran ist die Problematik der Ergonomie zu nennen. Viele Bürokrankheiten haben entweder mit der Überlastung des Handapparates zu tun oder werden durch die zwanghafte Sitzhaltung verursacht, die unter anderem durch die Verwendung einer Tastatur vorgeschrieben wird (Gerr, Marcus und Monteilh, 2004). Auch sind Menschen mit eingeschränkten motorischen Fähigkeiten, etwa aufgrund von Krankheiten oder fehlender Finger, nicht in der Lage, mit vergleichbarer Effizienz die Tastatur zu benutzen.

Zudem sind Tastaturen relativ groß und somit nur eingeschränkt portabel. Auf den immer kleiner werdenen mobilen Geräten wird die Erforderlichkeit einer Tastatur immer mehr zum limitierenden Faktor. Auf einer „Smartwatch" zum Beispiel gibt es in der Regel aufgrund der Größe keine eigene Tastaturfunktion, man muss diese mit einem Mobiltelefon verbinden.

Außerdem sind Tastaturen nicht flexibel für unterschiedliche Anwendungsfelder einsetzbar. Ein Beispiel ist die Grafikbearbeitung, bei welcher der Benutzer stufenlose Werte wie Pinselgröße oder Farbe anpasst. Die Tastatur mit ihren binären Tasten liefert hierfür keine zufriedenstellende Möglichkeit. Auch die Verwendung von Tastenkürzeln, bei der mehrere Tasten in Kombination gedrückt werden, um eine andere Funktion als Texteingabe zu bewirken, ist nicht benutzerfreundlich und vielen Benutzern unbekannt (Lane u. a., 2005).

1.2 Vision

In diesem Projekt versuchen wir[1], eine Alternative zur klassischen Tastatur zu entwickeln. Wir möchten ein System entwerfen, welches Text- und Zeicheneingabe, aber auch sonstige Interaktion mit einem digitalen Gerät ermöglicht, und dabei möglichst wenige der oben genannten Nachteile klassischer Tastaturen mit sich bringt. Das Tippen soll dabei komfortabler und flexibler werden.

Dazu möchten wir uns im ersten Schritt nicht zu weit von der gelernten Texteingabemethode entfernen. Die Bewegungen der Hand, die ein geübter Tastaturbenutzer im Muskelgedächtnis gespeichert hat und somit mühelos durchführen kann, möchten wir aufzeichnen und zu Tastendrücken konvertieren, ohne dass dafür die Verwendung einer herkömmlichen Tastatur nötig wäre. Dafür soll unser System in der Lage sein, die Bewegungen der Finger und der ganzen Hand zu messen und mithilfe maschinellen Lernens aus diesen die gedrückten Tasten abzuleiten.

1.3 Zielsetzung und Umfang

Da sich das Projekt im Umfang an 2 Bachelorarbeiten orientiert, ist es relativ begrenzt. Wir zielen darauf ab, anhand eines Prototyps eine prinzipielle Funktionsweise zu entwickeln, und somit den Grundstein für eine eventuelle Entwicklung eines benutzbaren Produktes zu legen.

Unser Projektziel lässt sich in drei Teile gliedern:

1. Entwurf eines Systems zur Aufzeichnung der charakteristischen Handbewegungen beim Tippen sowie der dazugehörigen Tastatureingaben.
2. Definition eines Ansatzes zum Rückschließen auf die Tastatureingaben aus den aufgezeichneten Bewegungen unter der Verwendung von Verfahren des maschinellen Lernens.

[1] Ich verwende in dieser Arbeit die erste Person Singular („ich"), um den Inhalt meiner Arbeit wiederzugeben, und die erste Person Plural („wir"), wenn ich mich auf das gesamte Projekt beziehe.

Abbildung 1: Der im Rahmen dieser Arbeit entwickelte Prototyp einer virtuellen Tastatur, hier gezeigt beim Aufzeichnen der Lerndaten. Am Handschuh sind die Sensoren (6 IMUs) sowie der Mikroprozessor befestigt. Auf den Mikroprozessor ist die Entwicklungsplatine aufgesteckt, welche Verkabelung der Sensoren ermöglicht.

3. Bewertung der Qualität dieser Rückschlüsse und der Nutzbarkeit eines solchen Verfahrens als Alternative zur klassischen Tastatur.

Ich werde in der vorliegenden Bachelorarbeit auf den Systementwurf eingehen. Dazu gehört vor allem die Entwicklung der Hardware, also die Wahl der Komponenten, die Mechanismen zur Datenübertragung und Befestigung, und der Entwurf der Software-Architektur.

Die Entwicklung und Konfiguration des Lernalgorithmus sowie die Experimente zum Erlernen eines Modells behandelt Carolin Konietzny in ihrer Arbeit (Konietzny, 2017).

Die Bewertung der einzelnen Leistungen findet sich selbstverständlich in beiden Arbeiten wieder, ebenso die Einschätzung des Gesamterfolgs unseres Projekts.

1.4 Abgrenzung

Im Rahmen der Bachelorarbeit schränken wir unsere Arbeiten auf einen Teilumfang der im Abschnitt 1.2 genannten Vision ein, und sehen unser Ziel primär in der Machbarkeitsanalyse. Wir erwarten nicht, ein fertiges Produkt zu entwickeln, mit welchem man mühelos und einwandfrei tippen kann – der Umfang wäre zu groß. Daher haben wir an einigen Stellen starke Vereinfachung gewählt, um möglichst vielen Problemen vorerst aus dem Weg zu gehen. Hierzu zählen unter anderem:

- Wir konzentrieren uns auf die Durchführung mit nur einer einzelnen Testperson. Unsere hergestellte Hardware sowie das gelernte Modell müssen nicht für andere Personen anwendbar sein.
- Die gewählte Testperson kann blind, aus dem Muskelgedächtnis, unter Verwendung (fast) aller Finger tippen. Eine Tippweise, bei der einzelne Tasten erst gesucht oder nur mit den Zeigefingern gedrückt werden, muss unser System nicht unterstützen.
- Aus Kostengründen und zur Verringerung der Komplexität statten wir vorerst nur eine Hand mit Sensoren aus.
- Die hergestellte Hardware muss, da es sich um einen Prototyp handelt, keine realistischen Qualitätsanforderungen an ein gebrauchsfähiges und alltagstaugliches Produkt erfüllen. Beim Design sollte jedoch darauf geachtet werden, dass die Entwicklung hierzu grundsätzlich möglich ist.
- Das entwickelte Verfahren für das maschinelle Lernen ist eines von vielen möglichen. Wir gehen nicht davon aus, dass wir in unserer Arbeit die beste Möglichkeit finden. Dies wird weitere Forschung erfordern, deren Umfang unsere Arbeiten sprengen würde. Unser Ansatz soll als Anfang dienen, um die allgemeine Machbarkeit einschätzen zu können und eine Vergleichsbasis schaffen.

1.5 Aufbau

Der Hauptteil dieser Arbeit erstreckt sich über 5 Kapitel. In Kapitel 2 beschreibe ich die Anforderungen an das zu entwerfende System und begründe deren Relevanz. Kapitel 3 zeigt verwandte Arbeiten und Projekte. Das tatsächliche Systemdesign mit den wichtigen Entscheidungen und deren Begründungen wird in Kapitel 4 erläutert. Im Kapitel 5 gehe ich kurz auf die Anwendung und die verwendeten Prinzipien des maschinellen Lernens ein, um dann in Kapitel 6 das entwickelte System zu bewerten und mit den anfangs gestellten Zielen vergleichen zu können. Im Fazit (Kapitel 7) fasse ich die Arbeit zusammen und gebe einen Ausblick auf weitere mögliche Forschungsthemen.

2 Ziele

In diesem Kapitel zeige ich die notwendigen Eigenschaften des zu entwerfenden Systems auf und begründe ihre Relevanz.

2.1 Kompatibilität zur klassischen Tastatur

Damit das Produkt eine taugliche Alternative zu einer klassischen Tastatur darstellt, muss es mindestens die gleiche Anzahl Eingaben wie diese unterscheiden können. Ebenso soll es Hilfstasten für Tastenkombinationen erkennen, sodass bestehende Software nicht angepasst werden muss. Wir möchten ausdrücklich mehr erreichen als nur eine beschränkte Eingabe zu erkennen, etwa limitiert auf die 26 Zeichen des lateinischen Alphabets.

Wir versuchen also, Tastenanschläge der Tastatur zu emulieren. Hierbei geht es uns nicht um die Wirkung, die diese Taste softwareseitig auslöst. Zum Beispiel wird das Tastaturlayout von unserem System nicht beachtet – stattdessen arbeiten wir wie eine normale USB-Tastatur mit Tastencodes, welche vom System dann einer beliebigen Wirkung zugewiesen werden können. Dies bedeutet im Besonderen, dass wir zwischen Groß- und Kleinbuchstaben nicht unterscheiden. Diese Unterscheidung findet erst im Betriebssystem statt, durch Interpretation der Shift-Taste.

2.2 Verwendung von maschinellem Lernen

Wie bereits in der Zielsetzung des Projektes festgehalten, soll das System Algorithmen des maschinellen Lernens (ML) verwenden, um Rückschlüsse auf die gemachten Tastatureingaben zu ziehen. Dies ist keine willkürliche Entscheidung. Wir halten dieses Projekt für einen geeigneten Anwendungsfall für maschinelles Lernen.

Ein Grund dafür ist, dass ein ML-Algorithmus viele Lerndaten benötigt, um darin Muster zu erkennen und zu erlernen und danach selbstständig Vorhersagen treffen zu können. Im Fall von überwachtem ML benötigt man zu den Eingabedaten auch die tatsächlichen Zielwerte (*ground truth*). Bei unserer Anwendung sind wir in der Lage, eine große Menge Lerndaten mit Zielwerten relativ einfach aufzuzeichnen, da wir die Handbewegung beim

Tippen auf einer normalen Tastatur zusammen mit den tatsächlichen Tastenanschlägen ermitteln können.

Außerdem sind ML-Algorithmen in der Lage, komplexe Zusammenhänge implizit zu lernen und wiederzuerkennen. Dazu gehören auch solche Zusammenhänge, die ein Programmierer bei der Entwicklung eines traditionellen Ansatzes nicht beachten würde. Ein Beispiel dafür ist die folgende Eigenart, welche ich an mir selbst beim Tippen beobachten konnte: Wenn ich den Buchstaben [B] [1] tippe, verwende ich dafür den Zeigefinger der rechten Hand. Zur gleichen Zeit bewegt sich mein linker Zeigefinger ein Stück nach links oben, um Platz zu machen. Dies ist ein Zusammenhang, der in meinen Bewegungsdaten erkennbar ist, und der für mich persönlich zutrifft. Ein ML-Algorithmus dürfte in der Lage sein, diesen Zusammenhang zu erkennen – ein Programmierer hätte dies jedoch vermutlich nicht als Regel im klassischen Algorithmus hinterlegt.

2.3 Messung charakteristischer Werte

Die Hardwarekomponenten des Systems müssen, damit das ML-Verfahren die unterschiedlichen Bewegungen zuordnen kann, die charakterischen Werte für die Bewegungen des Handapparates beim Tippen messen können. Im folgenden erläutere ich die dafür relevanten Aspekte.

2.3.1 Auswahl der Messwerte

Zunächst soll die innere Konfiguration der Hand gemessen werden, also die Position der Finger relativ zueinander und zu der gesamten Hand. Das liegt daran, dass die Finger eines geübten Tastaturbenutzers die meiste Arbeit erledigen. Je weniger langsame Handbewegungen nötig sind, desto schneller und effizienter ist das Tippen.

Bei den Fingerbewegungen kann der Rollwinkel (Drehung um die Längsachse) vernachlässigt werden, da die menschliche Hand zu dieser Bewegung nicht in der Lage ist. Vertikale und horizontale Bewegungen sind jedoch sehr wohl möglich und sollen erkannt werden, um benachbarte Tasten unterscheiden zu können.

Zudem müssen für einige Tastenanschläge die Hände bewegt werden, etwa um die oberen Tastenreihen (Ziffern und Funktionstasten) oder weiter außen oder innen liegende Tasten zu erreichen. Daher sollen auch Informationen über die Handbasis verfügbar sein, etwa die Position, Beschleunigung oder Orientierung.

[1] Wir verwenden für die gesamte Arbeit Tastaturen mit US-Layout. Die Testperson bin ich selbst – es sei angemerkt, dass ich zwar aus dem Muskelgedächtnis tippe, dies jedoch kein sauberes 10-Finger-System ist. Stattdessen ruhen die Finger meiner rechten Hand auf den Tasten HJKL, also eine Taste nach links verrutscht. Diese Erklärung ist notwendig zum Verständnis der später gezeigten Graphen und Experimente.

2.3.2 Genauigkeit

Die Tasten einer Tastatur sind relativ klein. Um unter zwei benachbarten Tasten unterscheiden zu können, muss die Genaugkeit der gelieferten Daten hoch genug sein. Aufgrund der vielen möglichen messbaren Größen können wir bezüglich der Genaugkeit keine quantitativen Anforderungen stellen. Ich werden jedoch im Analyseteil auf diese Fragestellung eingehen, und die ermittelten Daten darauf untersuchen, ob benachbarte Tasten daran unterscheidbar sind.

2.3.3 Datenrate

Geübte Benutzer können etwa 200 bis 400 Anschläge pro Minute[2] erreichen. Dies entspricht im Durchschnitt einem Intervall 150 ms bis 300 ms pro Anschlag. Aus diesem Grund muss die Datenrate ausreichend sein, um die komplette Bewegung zu erkennen. Sowohl die Hardware- als auch die Softwarekomponenten müssen diese Datenrate unterstützen. Wir haben uns eine Datenrate von 100 Hz zum Ziel gesetzt, erhalten also mindestens 15 Datenpunkte für jeden Tastenanschlag eines geübten Benutzers.

2.4 Geringe motorische Einschränkung

Eine geringe motorische Einschränkung des Benutzers durch die Sensoren ist von zweierlei Nutzen. Zunächst soll der Proband bei der Aufzeichnung der Lerndaten sowie bei der tatsächlichen Nutzung die gleichen Bewegungen aus dem Muskelgedächtnis abspielen. Dazu ist es notwendig, dass er in der Lage ist, eben diese Bewegungen auch durchzuführen.

Auch bei der späteren Anwendung ist eine Einschränkung des Benutzers durch die Sensoren nicht erwünscht. Ein alltagstaugliches Gerät sollte flexibel und leicht sein, insbesondere wenn es die Befestigung von Hardware an der Hand erfordert.

2.5 Haltungsunabhängigkeit

Um bessere ergonomische Eigenschaften gewährleisten zu können, sollten die Handbewegungen in jeder Körperhaltung durchgeführt werden können, zum Beispiel im Stehen, Sitzen oder Liegen. Das System soll unabhängig von dieser Körperhaltung die Eingaben erkennen können. Zum Tippen sollte auch keine bestimmte Armhaltung vorgeschrieben sein. Müsste man beispielsweise die Arme ausstrecken, um im Stehen zu tippen, würden diese schnell ermüden. In diesem Falle ist es eher wünschenswert, die Arme locker an den Seiten herunterhängen zu lassen, und nur mit der Bewegung der Finger tippen zu können.

[2] vgl. Statistik unter [1]

Zudem soll das System nicht ortsgebunden sondern mobil einsetzbar sein. Vorstellbar wäre die Verwendung mit einem mobilen Endgerät (z.B. Smartphone) oder künftig sogar „deviceless", also nur in Kombination mit peripheren Geräten wie einer AR-Brille. Wünschenswert bei Verwendung von Hardware an der Hand wäre die Möglichkeit, diese kabellos betreiben zu können.

2.6 Prototyping-geeignete Software

Der Entwurf des Systems beinhaltet nicht nur Hardwarekomponenten, sondern auch die Architektur der verwendeten Software. Diese muss nicht nur die funktionalen Anforderungen erfüllen, sondern soll in einer Weise strukturiert sein, die unterschiedliche, transparente und wiederholbare Experimente erlaubt, und somit gut für Prototyping geeignet ist.

2.7 Ausbaufähigkeit zu marktfähigem Produkt

Damit das entworfene System nicht nur ein Prototyp bleibt, sondern irgendwann eine echte Alternative zur Tastatur für viele Benutzer werden kann, muss es von vornherein darauf ausgelegt sein, zu einem marktfähigen Produkt entwickelbar zu sein. Dafür sollten die Hardwarebausteine günstig sein, insbesondere in Massenproduktion. Das fertige Produkt muss robust und alltagstauglich sein.

Es wäre wünschenswert, sowohl die Hardware als auch die Software soweit generalisieren zu können, dass ein neuer Benutzer diese nicht aufwendig anpassen muss. Wie in Abschnitt 1.4 erwähnt lassen wir diesen Gesichtspunkt in unserer Arbeit jedoch vorerst außer Acht.

3 State of the Art

In diesem Kapitel gebe ich einen Überblick über Projekte, welche ähnliche Zielsetzungen haben, sowie über publizierte Forschungsarbeiten zu verwandten Themen.

3.1 Sensorhandschuhe

Verschiedene Arten von Sensorhandschuhen und sogenannten „smart gloves" finden in der Forschung Anwendung. Die einfachsten davon enthalten Taster oder leitfähige Flächen [2], um direkte Eingaben zu ermöglichen (vgl. Abbildung 2). Andere basieren auf Flex- oder Stretch-Sensoren (Li u. a., 2011), um den Biegungsgrad der Finger, teilweise auch der einzelnen Fingerglieder zu messen. In Unterabschnitt 4.1.1 gehe ich darauf ein, warum wir diesen Ansatz nicht verfolgt haben. Eine Übersicht über diese frühen Ansätze bieten Dipietro, Sabatini und Dario (2008).

Die Verwendung von IMUs[1] in Sensorhandschuhen ist relativ neu (in der Übersicht von 2008 finden sich noch keine Projekte mit IMUs), sie sind besonders in der medizinischen Forschung präsent. Sensorhandschuhe können hier zum Beispiel Ärzte unterstützen, indem sie die Handbewegungen von Patienten während der Rehabilitation nach Handverletzungen aufzeichnen (Lin u. a., 2014). An den Bewegungswerten sind Muster zu erkennen, welche auf den Fortschritt der Heilung schließen lassen. Sensorhandschuhe werden ebenso zu Untersuchungen in Studien der Parkinson-Krankheit eingesetzt (Cavallo u. a., 2013).

Andere Projekte erweitern den Sensorhandschuh um Komponenten für Feedback an den Benutzer, hauptsächlich visuelles Feedback durch LEDs und haptisches Feedback durch *vibro-tactile stimulators*. Die „InerTouchHand" (Lobo und Trindade, 2013) zum Beispiel wurde zur Forschung der *human-machine interaction* entwickelt, Zielanwendung ist die Steuerung einer menschenähnlichen Roboterhand. Auch dieses Projekt verwendet IMUs zur Messung der Handhaltung.

Es gibt ferner kommerzielle Produkte, welche ähnliche Technologien verwenden. Angetrieben wird die Entwicklung dieser Produkte meist von der Computerspielebranche. Ein populäres Produkt ist die Nintendo Wii™ Spielekonsole, in deren Fernbedienung ein Ac-

[1] Inertial measurement units; in Unterabschnitt 4.1.3 erläutere ich deren Aufbau und Funktionsweise.

Bildquelle: https://vimeo.com/23269969

Abbildung 2: Der „Keyglove" ist ein tragbares Gerät zur Tastatureingabe, basierend auf leitfähigen Kontaktpunkten [2].

celerometer verbaut ist [3], um innovative Steuerung von Spielen zu ermöglichen. Auch in der Virtual-Reality Branche wird viel mit diesen Sensoren gearbeitet. So enthalten zum Beispiel die Brille und Hand-Controller des HTC Vive® VR-Systems die InvenSense MPU-6500 (Accelerometer und Gyroskop) [4].

Ein tatsächlicher Sensorhandschuh ist der Hi5 VR Glove von Noitom Ltd, ein kürzlich vorgestelltes Produkt. Es wird als „wireless consumer glove designed for virtual reality headsets" [5] beworben, ist jedoch noch nicht auf dem Markt erhältlich. Der Handschuh enthält, genau wie unser Prototyp, sechs IMUs. Eine Software, die Tastaturemulation durchführt, wird allerdings nicht erwähnt.

3.2 Alternative Texteingabegeräte

Sehr ähnlich zu unserem Projekt war das Produkt *Gest* [6]. Hierbei handelte es sich um ein tragbares Gerät, welches an der Hand und 4 Fingern befestigt wurde (vgl. Abbildung 3) und ebenfalls IMUs enthielt. Das Produkt wurde im November 2015 mit einer Kickstarter-Kampagne[2] finanziert, und sollte ein General-Purpose-Device zur Interaktion mit Computern werden. Die Entwickler stellten das Gerät auch, anders als der Hersteller des Hi5 VR Glove, für die Verwendung als Tastatur vor, und zeigten beispielhaft in einem Werbevideo[3] diese Funktionalität. Sie spezifizierten leider nicht öffentlich wie die Erkennung durchgeführt wurde, und das Projekt wurde aufgrund mangelnder Investitionen

[2] https://www.kickstarter.com/projects/asdffilms/gest-work-with-your-hands
[3] https://vimeo.com/143556093

Mitte 2016 eingestellt.

Eine weitere zu unserem Projekt sehr ähnliche Arbeit ist „Gestures as input: neuroelectric joysticks and keyboards“ (Wheeler und Jorgensen, 2003). Damit wurde bereits 2003 ein Ansatz entwickelt, um mithilfe von EMG-Messungen Joysticks und Tastaturen zu emulieren. Für letztere befestigten die Autoren 8 EMG-Elektroden am Unterarm des Probanden, zeichneten Bewegungen auf und lernten, die Muster mit Hidden Markov Models zu erkennen. In ihren Versuchen stellten sie Probleme bei der Verwendung von EMG-Eletroden fest. Grundsätzlich war die Leistung des Systems von der Position der Elektroden, Schweiß, trockener Haut, Tragezeit sowie der allgemeinen Tagesform des Probanden abhängig. Für die Steuerung eines Flugzeuges im Simulator waren die 4 erkannten Joystick-Gesten (links, rechts, auf, ab) ausreichend akkurat. Das Tippen auf dem Ziffernblock einer Tastatur klappte ebenfalls mit etwa 80% Genauigkeit, wenn das Modell am Tag der Verwendung trainiert wurde, und die Elektroden somit identisch platziert waren.

Bildquelle: https://gest.totemapp.com/company

Abbildung 3: Gest sollte ein tragbares General-Purpose-Device zur Interaktion mit Computern werden. Dazu war ebenfalls die Umsetzung einer virtuellen Tastatur angekündigt. Das Projekt wurde jedoch eingestellt. (Werbebild)

3.3 Visuelles Hand-Tracking

Forscher von Microsoft Research haben einen Algorithmus entwickelt, um Handposen aus Kamerabildern mit Tiefeninformation zu extrahieren (Taylor u. a., 2016). Verwendet wird hierbei eine Microsoft Kinect. Die rekonstruierte Handpose wird für die Manipulation virtueller 3D-Objekte benutzt.

„*The Learning Keyboard*" (Ellithorpe und Tan, 2012) verwendet ebenfalls eine Kinect, allerdings um getippte Wörter auf einer Tastatur zu erkennen. Hierfür kommen verschiedene Bildverarbeitungsschritte zur Vorverarbeitung und *Support Vector Machines* zur Klassifikation zum Einsatz. Diese erlangten eine erstaunliche Genauigkeit von über 80% auf einem Datensatz mit 114 verschiedenen Wörtern.

3.4 Maschinelles Lernen und IMUs

Ein weiterer relevanter Forschungsbereich ist die *Human Activity Recognition* (HAR). Das Ziel der HAR ist es, menschliche Aktivitäten aus Sensordaten zu erkennen. Hierfür kommen zahlreiche unterschiedliche Sensoren zum Einsatz, darunter Accelerometer, Gyroskope und Magnetometer aber zum Beispiel auch GPS, Mikrofone, Drucksensoren, Kameras und Thermometer. Mithilfe von Methoden des maschinellen Lernens wird versucht, zwischen alltäglichen Aktionen zu unterscheiden, etwa dem Gehen, Stehen, Sitzen, Liegen, aber auch Öffnen von Türen oder Schubladen sowie die Manipulation kleinerer Objekte (Lara und Labrador, 2013). Verwendet werden dafür seit längerem Ansätze des maschinellen Lernens. Inzwischen kommen unter anderem auch *Convolutional Neural Networks* (CNNs) zum Einsatz (Yao u. a., 2017), welche wir auch in diesem Projekt verwenden.

4 Systemdesign

In diesem Kapitel stelle ich die tatsächliche Umsetzung des Projektes vor. Ich werde auf alle relevanten Designentscheidungen eingehen und erläutern, welche Alternativen es gab und warum wir welche Methode gewählt haben. Insbesondere geht es dabei um die Auswahl und Befestigung der Hardwarekomponenten, die System-Architektur, die unterstützenden Softwarekomponenten sowie die Verbindungen zwischen den jeweiligen Komponenten.

4.1 Sensortypen

Den Kern der Hardware bilden die Sensoren, die die Bewegungen der Hand aufzeichnen. Hierfür kommen einige Sensortypen infrage, welche ich nun vorstellen werde. Dabei werde ich erläutern, warum wir uns in diesem Projekt für die Verwendung von IMUs entschieden haben.

4.1.1 Flex- und Stretchsensoren

Viele Forschungsprojekte, die mit Sensorhandschuhen arbeiten, verwenden Flexsensoren um den Beugungsgrad der Finger oder Fingerglieder zu messen. Diese Flexsensoren können zum Beispiel aus einem biegsamen elektrischen Widerstand bestehen, dessen Widerstandswert vom Grad der Biegung abhängig ist. Mithilfe eines Spannungsteilers kann ein analoges Signal erzeugt werden, dessen Stärke die Biegung angibt. Alternativ kommen Stretchsensoren (z.B. optische Linearcodierer) zum Einsatz, welche die Verlängerung eines auf der Oberseite der Finger gespannten Bandes beim Beugen der Finger messen.

Beide Sensortypen sind zwar sehr einfach zu verwenden, haben jedoch einige Einschränkungen. So lassen sich Flexsensoren meist nur in eine Richtung biegen und werden beschädigt, wenn man sie in die falsche Richtung zwingt, und Stretchsensoren können keine Stauchung messen.

Außerdem liefern sie nur einen eindimensionalen Wert. Die vielen Freiheitsgrade der Hand zu erkennen würde entsprechend viele dieser Sensoren erfordern. Dies ist zum einen nicht wünschenswert, da sowohl die Kosten als auch der Wartungsaufwand hoch wären,

zum anderen würden viele Sensoren den Tragekomfort einschränken und flüssiges Tippen erschweren.

Hinzu kommt, dass Flex- und Stretchsensoren nur einen relativen Wert zu einem physisch nahen Bezugspunkt angeben können. Man könnte die gesamte Hand mit Sensoren ausstatten und wäre dann trotzdem nur in der Lage die Handbewegungen, nicht aber Bewegungen des ganzen Armes über der Tastatur zu erkennen.

Aus diesen Gründen haben wir uns gegen die Verwendung dieser Art von Sensor entschieden.

4.1.2 Visuelles System

In Abschnitt 3.3 habe ich einige Projekte vorgestellt, in denen Kamera-Tracking verwendet wird. Dazu wird ein zwei- oder dreidimensionales Bild der Hände aufgenommen und analysiert. Hierfür kommen komplexe Bilderkennungsalgorithmen zum Einsatz, um die Position der Hand und der Finger zu extrahieren.

Wir haben uns aus zweierlei Gründen gegen diesen Ansatz entschieden. Zunächst ist bereits das Extrahieren der relevanten Daten ein erheblicher Aufwand und ein schwieriges Problem, sodass wir vermutlich nicht die angestrebte Genauigkeit erreichen würden. Außerdem haben handelsübliche Kameras eine Bildrate von $30\,\mathrm{Hz}$, sodass wir unsere angestrebte Datenrate damit nicht erreichen könnten.

Hinzu kommt, dass man hierfür eine Kamera korrekt platzieren müsste und sich dann nicht aus ihrem Sichtfeld entfernen dürfte. Dies widerspricht unserem Anspruch, ein von der Haltung und Position unabhängiges und mobiles System zu entwickeln.

4.1.3 Inertial Measurement Units (IMUs)

Zuletzt möchte ich die Vorteile von IMUs aufzeigen, welche uns dazu bewegt haben, diese zu verwenden.

IMUs sind heutzutage als integrierte Schaltungen leicht erhältliche Bausteine zur Messung kinetischer Eigenschaften. Sie enthalten in der Regel zwei oder drei MEMS-Bausteine (mikroelektromechanisches System): Das Accelerometer misst die lineare Beschleunigung, das Gyroskop die Winkelgeschwindigkeit, und das Magnetometer die Ausrichtung relativ zum Erdmagnetfeld. Üblich sind hierbei jeweils 3 DoF (*degrees of freedom*, Freiheitsgrade), also Messungen in jeweils 3 verschiedenene Achsen.

Häufig werden diese IMUs als „System-in-Package" ausgeliefert, das heißt sie enthalten einen Mikrocontroller, der eine Vorverarbeitung der gemessenen Daten durchführt und diese, genau wie auch die Rohdaten, über ein geeignetes Protokoll, zum Beispiel I^2C, zur Verfügung stellt. Dabei läuft ein Fusionsalgorithmus auf dem Chip, der versucht, aus den gemessen Werten die absolute Orientierung im Raum abzuleiten. Diese wird dann als

Euler-Winkel oder Quaternion[1] kodiert bereitgestellt.

Besonders interessant ist für uns die Unabhängigkeit dieser Sensoren von ihrer Befestigung. Durch diese sind wir in der Lage, für jeden Finger eine IMU zu verwenden. Um Fingerbewegungen relativ zur Hand und auch Bewegungen der ganzen Hand ermitteln zu können, wird zusätzlich eine IMU am Handrücken befestigt.

Wir halten die relative Beschleunigung und die relative Orientierung der Finger zur Hand für die beiden aussagekräftigsten Messwerte beim Tippen. Diese Werte erhalten wir mit wenig Aufwand und sowohl in ausreichender Genauigkeit als auch angemessener Geschwindigkeit von handelsüblichen IMUs.

IMUs haben jedoch auch einige Nachteile. Insbesondere die kostengünstigeren MEMS-Bausteine sind teilweise ungenau und ihre Daten rauschen mehr oder weniger stark. In Flugzeugen beispielsweise kommen für die Navigation auch IMUs zum Einsatz, hier werden dann allerdings hochwertige und sehr teure IMUs verwendet, die auf anderen Technologien basieren.

Problematisch sind bereits sehr kleine Messfehler. Möchte man anstatt der Beschleunigung vom Accelerometer die Position des Sensors errechnen, muss man diese zweimal über die Zeit integrieren (Beschleunigung → Geschwindigkeit → Position). Eine kleine Abweichung der Beschleunigung bewirkt von da an eine falsche Geschwindigkeit, sodass die Position immer weiter abweicht. Um dies zu verhindern, sind möglichst genaue Daten, gute Kalibrierung, und Fehlerkorrekturschritte nötig, etwa das Annullieren der Geschwindigkeit zu bekannten Zeitpunkten der Ruhe. Ähnliches gilt für die Orientierung, da das Gyroskop nur die Rotationsgeschwindigkeit ermitteln kann – diese kann jedoch durch Messung des Gravitationsvektors und des Erdmagnetfeldes besser korrigiert werden.

4.2 System-Architektur

In Abbildung 4 zeigen wir die grundlegende Architektur unseres Systems. Sie beinhaltet alle Komponenten und deren Beziehungen, die hauptsächlich Datenströme wiederspiegeln. Das gesamte System kann in zwei Modi betrieben werden. Beim Lernen kommen dabei andere Komponenten zum Einsatz als bei der Anwendung.

Gemein ist beiden Modi der Prozess der Datenerfassung. Die rohen Sensordaten und fusionierten Quaternionen werden vom Mikroprozessor aus den Sensoren extrahiert und an den Host-PC übertragen. Hierfür stehen zwei alternative Kanäle zur Verfügung (USB oder WLAN). Das Nachrichtenformat wird im Abschnitt 4.7 „Datenübertragung" genauer betrachtet. Die übermittelten Nachrichten werden vom *Serial Parser* interpretiert.

Im Lernmodus wird außerdem jede Tastatureingabe vom Betriebssystem durch den *Key-*

[1] Die Quaternion ist ein mathematisches Konstrukt, welches unter anderem für die Repräsentation von Drehungen oder Orientierungen im dreidimensionalen Raum verwendet werden kann. Sie besteht aus 4 Werten und hat rechnerisch einige Vorteile gegenüber z.B. Euler-Winkeln.

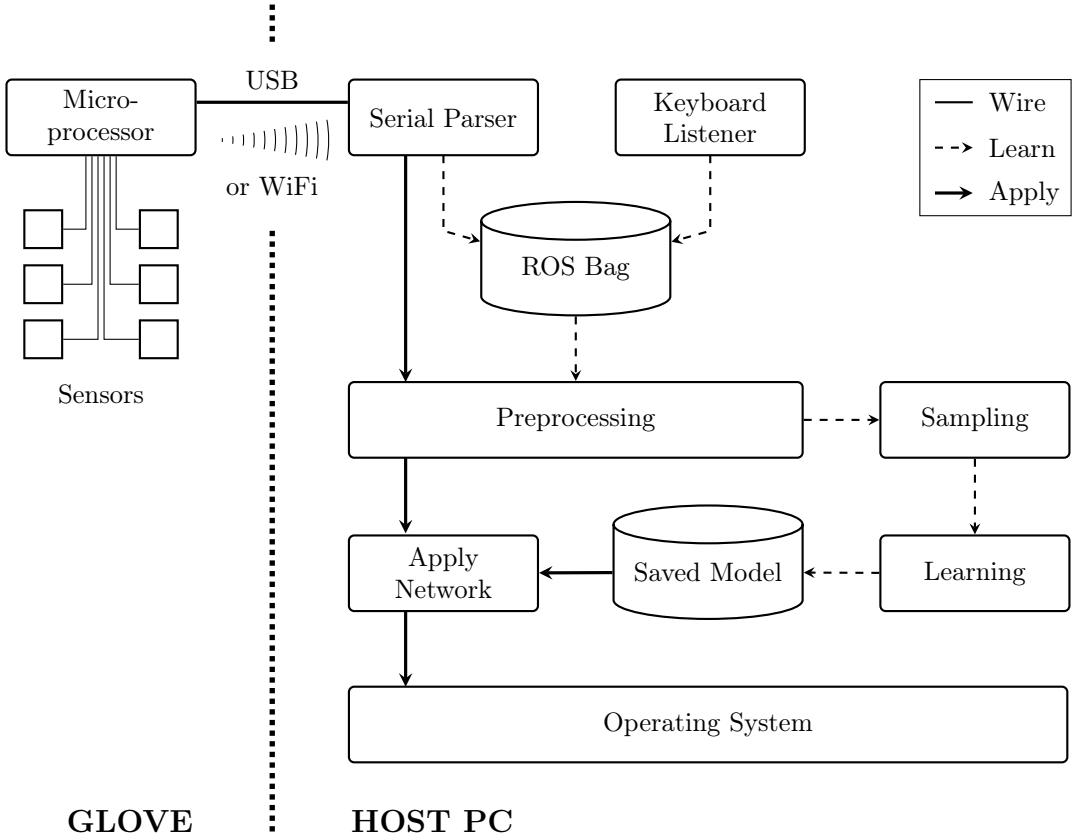
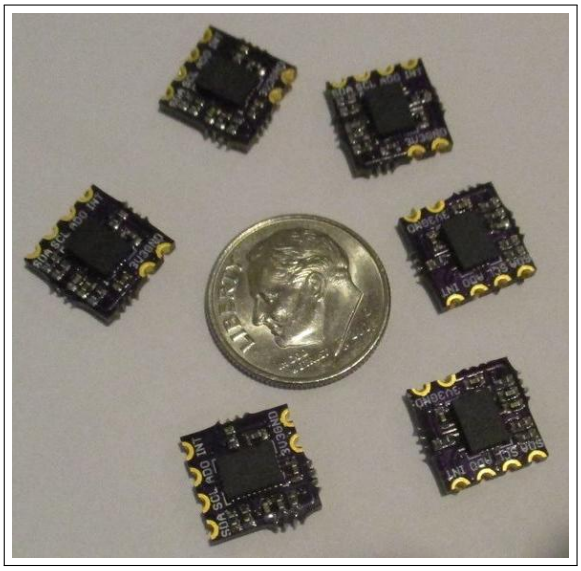


Abbildung 4: Aufbau des Systems. Auf der linken Seite sind die Hardwarekomponenten im Handschuh, auf der rechten die Softwarekomponenten auf dem Host-PC dargestellt. Die Verbindungen geben an, wie die Daten durch die Komponenten gereicht werden.
Es sind 2 Modi dargestellt. Im Lernmodus wird das Modell erstellt und zwischengespeichert, im Anwendungsmodus wird dieses verwendet.



Bildquelle: https://www.tindie.com/products/onehorse/wearable-bno055-nano-board/

Abbildung 5: Das Wearable BNO055 Nano Board ist mit $10\,\mathrm{mm} \times 10\,\mathrm{mm}$ Größe ein besonders kleines Breakout Board. Es enthält den von uns gewählten Sensor BNO055, wir verwenden pro Handschuh 6 dieser Boards.

board Listener abgefangen. Sowohl die Tastatureingaben als auch die Sensordaten werden dann zum späteren Lernen zwischengespeichert. Das Lernen besteht aus den Schritten *Preprocessing*, *Sampling* und dem tatsächlichen *Learning*. Das gelernte Modell wird ebenfalls gespeichert.

Im Anwendungsmodus benötigen wir keine Tastatureingabe, stattdessen wird diese generiert. Hierfür werden die erfassten Daten direkt vorverarbeitet, und an das neuronale Netz weitergegeben (*Apply Network*), welches das zuvor gelernte Modell geladen hat. Die Ausgaben des Netzes werden dann in das Betriebssystem als Tastatureingaben eingespeist.

4.3 Auswahl des IMU-Boards

Bei der Auswahl der tatsächlichen IMU haben wir uns in erster Linie an der Größe des Entwicklungsboards orientiert. Einen Eigenbau in SMD-Technologie zu löten kam für den Prototyp nicht infrage. Stattdessen suchten wir nach einem „Breakout Board"[2], welches klein genug ist, um es an der Hand zu befestigen und das Tippen nicht einzuschränken.

[2]Breakout Boards sind Platinen, die einen oder mehrere IC-Bausteine enthalten, und auf denen ausgewählte Pins dieser Bausteine zum Anlöten zur Verfügung stehen. Häufig enthalten Breakout Boards auch weitere Sekundärbauteile zur Ansteuerung, etwa Widerstände oder Kondensatoren. Dies erleichtert die Ansteuerung der ICs und die Entwicklung von Prototypen.

Ein möglicher IMU-Typ war die InvenSense MPU-9150, welche die klassische MPU-6050 sowie einen AK8975 3D-Kompass enthält. Diese IMU ist in vielen verschiedenen Breakout Boards verfügbar, dadurch sind auch relativ kleine Bauweisen erhältlich[3].

Als Alternative haben wir die Bosch Sensortec BNO055 evaluiert. Besonders der Formfaktor des Wearable BNO055 Nano Boards[4] (siehe Abbildung 5) ist für unseren Anwendungszweck interessant. Dieses Board ist lediglich $10\,\mathrm{mm} \times 10\,\mathrm{mm}$ groß und der eingebaute Fusionsalgorithmus verspricht Datenraten von bis zu $100\,\mathrm{Hz}$.

The BNO055 enthält ein 3 DoF 14-bit Accelerometer, ein 3 DoF 16-bit Gyroskop und ein 3 DoF Magnetometer (Bosch Sensortec, 2016). Das Accelerometer lässt sich konfigurieren auf einen Wertebereich von $\pm 2\,\mathrm{g}/\pm 4\,\mathrm{g}/\pm 8\,\mathrm{g}/\pm 16\,\mathrm{g}$. Für das Gyroskop stehen Wertebereiche zwischen $\pm 125\,^\circ/\mathrm{s}$ und $\pm 2000\,^\circ/\mathrm{s}$ zur Verfügung. Der enthaltene Mikroprozessor ist ein 32-bit Cortex M0+, auf dem der proprietäre Bosch Sensortec Fusionsalgorithmus vorinstalliert ist.

Dies macht die Ansteuerung des Sensors besonders einfach, da lediglich über I^2C die entsprechenden Datenregister ausgelesen werden müssen. Im Gegensatz dazu mussten wir bei der MPU-9150 zuerst eine passende Software aufspielen.

Außerdem ist das Breakout Board der BNO055 kleiner und leichter. Für diese Vorteile haben wir den höheren Preis von ca. 24,00 € statt ca. 15,00 € für die MPU-9150 in Kauf genommen und in unserem System die BNO055 verwendet.

Hervorzuheben ist bei der BNO055 weiterhin, dass der Fusionsmodus relativ mächtig ist. Ist die Sensorfusion aktiviert, stellt der enthaltene Prozessor nicht nur die berechnete Orientierung (wahlweise als Quaternion oder Euler-Winkel) bereit, sondern ebenfalls den Gravitationsvektor und die gravitationsbereinigte Beschleunigung (jedoch relativ zum Sensor). Auch die Gyroskop- und Magnetometerdaten werden bei aktiviertem Fusionsmodus bereinigt. Hierfür führt die IMU interne Selbsttests und Echtzeitkalibrierungen durch. Diese sind im Rohdatenmodus leider nicht verfügbar.

4.4 Auswahl des Prozessor-Boards

Auch für die Auswahl des Prozessors ist der Formfaktor relevant. Da wir diesen jedoch nicht an den Fingern anbringen müssen, sondern am Handgelenk befestigen, können wir uns auf andere Features konzentrieren.

Wir wählten das Adafruit Feather M0 WiFi[5] („Featherboard") aus mehreren Gründen [7]:

[3] zum Beispiel `https://new.sparkfun.com/products/13762`, ein Board mit dem Nachfolger MPU-9250 – inzwischen wird die MPU-9150 nicht mehr verkauft.
[4] erhältlich unter `https://www.tindie.com/products/onehorse/wearable-bno055-nano-board/`
[5] `https://www.adafruit.com/product/3010`

- Das Board ist mit 6.1 g besonders leicht, und mit 5.4 mm × 2.3 mm auch relativ klein.
- Es enthält das ATWINC1500 WLAN-Modul, und kann somit ohne weitere Hardware kabellos mit dem PC verbunden werden.
- Der Prozessor (ATSAMD21G18 / Cortex M0+) hat 6 eingebaute SERCOMs (Bausteine zur seriellen Kommunikation). Jeder dieser SERCOMs kann UART, I^2C und SPI unterstützen. Zur Verwendung mit der BNO055 benötigen wir 3 dieser Ports (siehe Abschnitt 4.5).
- Das Featherboard kann per LiPo-Akku betrieben werden, und taugt gleichzeitig als Ladegerät für diesen, wenn es per USB an eine Stromversorung angeschlossen ist. Wir haben den Akkubetrieb nicht getestet, prinzipiell erlaubt uns aber das Featherboard den einfachen Umstieg.

Ein paar weitere Merkmale dieses Produktes, welche für unseren Prototyp vorerst weniger relevant waren, möchte ich ebenfalls aufzeigen:

- Der Prozessor ist mit 48 MHz getaktet und bietet mit 256 kB Flash und 32 kB SRAM großzügig Speicher für einfache Anwendungen (im Vergleich hat der Arduino Uno mit dem ATmega328P nur 32 kB Flash und 2 kB SRAM bei 16 MHz Taktrate).
- Der Prozessor ist Arduino-kompatibel. Arduino ist eine open-source Plattform, welche es vereinfacht, Mikroprozessoren zu programmieren und damit deren Verwendung unkompliziert und leicht zugänglich macht. Somit ist auch das Featherboard leicht zu programmieren – ideal für einen Prototyp. Der Hersteller Adafruit liefert alle benötigten Bibliotheken und eine gute Dokumentation.
- Die Entwicklung der Software wird weiterhin durch die gute USB-Unterstützung vereinfacht.
- Es stehen 20 GPIO Pins zur Verfügung, 8 davon unterstützen PWM.
- Der Prozessor und das Featherboard enthalten kein EEPROM, also schreibbaren persistenten Speicher. Dieser ist für unsere Anwendung jedoch erforderlich, um die WLAN-Zugangsdaten zu speichern. Wir müssen daher ein externes EEPROM in die Schaltung einbringen.

4.5 Ansteuerung der Sensoren via I^2C

Die Ansteuerung der gewählten Sensor-Boards erfolgt ausschließlich über den I^2C-Bus. Zwar unterstützt die BNO055 auch die Ansteuerung über UART, allerdings ist das Breakout Board nur für I^2C konfiguriert. Über den Pin `AD0` lässt sich auswählen, auf welcher der 2 möglichen Slave-Adressen (`0x28` oder `0x29`) die BNO055 antwortet.

	Primäre Pads				Alternative Pads				
SERCOM	0	1	2	3	0	1	2	3	Standardbelegung
0	4†	3	1	0	(A3)	(A4)	8	9	Serial1
1	11	13	10	12	–	–	–	–	
2	22	–	2†	5	4†	3	1	0	
3	20	21	6*	7*†	(11)	(13)	10	12	Standard-I²C
4	22	–	23*	24*	A1	A2	2†	5	SPI
5	A5*	–	6	7†	(20)	(21)	–	–	Debugging Port

∗ müssen als alternative Pads konfiguriert werden
† Pins 2, 4, 7 und 8 werden intern für das WLAN-Modul verwendet.

Tabelle 1: Aus dem Datenblatt des Featherboards gewonnene Übersicht der SERCOMs mit ihren verfügbaren Pins. Jeder SERCOM kann I²C, UART und SPI-Schnittstellen an verschiedenen Pins bereitstellen. Die verfügbaren Kombinationen lassen sich hier ablesen. Unsere gewählten Pins sind eingekreist.

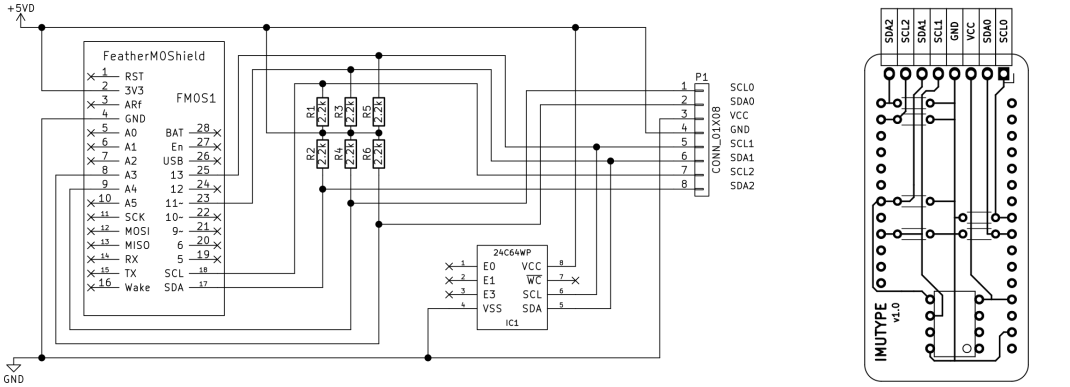
Um 6 Sensoren des gleichen Typs ansteuern zu können, von denen jeder nur 2 mögliche Adressen annehmen kann, benötigen wir 3 verschiedene Adressräume. Dies lässt sich entweder durch getrennte Busse oder durch Adress-Translatoren erzielen.

Wie bereits erwähnt, stehen beim Featherboard 6 SERCOMs zur Verfügung, das heißt, wir können 3 getrennte Busse aufbauen und mit je einem Prozessor alle Sensoren einer Hand ansteuern.

Jeder SERCOM kann zur Verwendung mit I²C, UART oder SPI konfiguriert werden. Dabei stehen für jeden dieser Bausteine (bis auf Ausnahmen) zwei Sätze Pins zur Verfügung, an denen die serielle Schnittstelle realisiert wird. Die Softwarebibliothek ermöglicht es, die SERCOMs zu konfigurieren, und für den jeweiligen Baustein den verwendeten Modus und die Pins festzulegen.

In Tabelle 1 sieht man die möglichen Pins für das Betreiben der SERCOMs. Für die Verwendung von I²C sind nur Pads 0 und 1 nötig. Bei der Auswahl der Pins ist zu beachten, dass der gewünschte SERCOM nicht für eine verwendete Funktion belegt ist, und dass sich die gewählten Pins nicht überschneiden. Wir haben uns willkürlich aus den möglichen für die hervorgehobe Kombination entschieden. Wir definieren 3 I²C-Busse, Bus 0 an Pins A3 und A4, Bus 1 an Pins 11 und 13, und Bus 2 an Pins 20 und 21.

In Abbildung 6a sehen wir den Schaltplan für die Verkabelung der Komponenten am Handgelenk. Nicht enthalten sind die tatsächlichen Sensoren, diese werden am Ende an die jeweiligen Busse angebracht (VCC, GND, SDA, SCL an die dazugehörigen Pads auf dem Sensor-Board). Am rechten Rand befindet sich stattdessen die Buchsenleiste, die



(a) Schaltplan (b) Leiterbahnlayout

Abbildung 6: Die Verkabelung der zur Ansteuerung der Sensoren benötigten Komponenten erfolgt auf dem „Shield", einer Platine, welche sich auf das Featherboard aufstecken lässt (vgl. Abbildung 7b).

SDA und SCL aller Busse sowie VCC und GND zur Verfügung stellt. Das EEPROM vom Typ 24C64WP ist an Bus 1 angeschlossen. Zusätzlich enthalten sind die Pull-up-Widerstände, die erforderlich sind, weil das Featherboard keine integrierten Pull-up-Widerstände enthält.

Um die Verkabelung einfach zu halten, entwarfen wir einen „Shield" zum Aufstecken auf das Featherboard. Dieses entspricht dem gezeigten Schaltplan, und beinhaltet eine Buchsenleiste zum Anstecken des Handschuhs. Abbildung 6b zeigt ein mögliches Leiterbahnlayout für diese Schaltung. Es hat aufgrund der Anordnung der Busse keine Überkreuzungen und benötigt somit nur eine Kupferschicht. Die Busse sind von links nach rechts in umgekehrter Reihenfolge angeordnet, dazwischen befinden sich GND und VCC. Die Buchsenleiste ist „oben", in Handrichtung.

Aus der Anordnung der Busse ergibt sich Tabelle 2, welche die Zuordnung der IMUs zu ihrem Bus zeigt, sowie die gewählte Adresskonfiguration. Außerdem erhält jede IMU eine ID zur späteren Identifizierung der Daten. Auch hier haben wir darauf geachtet, möglichst wenige Überkreuzungen zu verursachen. Da die linke und rechte Hand Spiegelbilder sind, aber der gleiche Prozessor und Shield verwendet werden, haben wir die Reihenfolge der Bus-Zuordnungen umgekehrt. Wie zu erkennen ist, werden die Busse von rechts nach links an der jeweiligen Hand angeordnet, statt von innen nach außen.

4.6 Befestigung

Besonders für das Ziel der geringen motorischen Einschränkung ist die Wahl der Befestigungsmethode relevant. Die Testperson muss in der Lage sein, die im Muskelgedächtnis gespeicherten Bewegungen ohne oder mit nur minimalen Beeinflussungen auszuführen.

Hand	IMU	Platzierung	Bus	Adresse	AD0-Pin
rechts	0	Handrücken	2	0x28	low
	1	Daumen	2	0x29	floating
	2	Zeigefinger	1	0x28	low
	3	Mittelfinger	1	0x29	floating
	4	Ringfinger	0	0x28	low
	5	Kleiner Finger	0	0x29	floating
links	6	Handrücken	0	0x28	low
	7	Daumen	0	0x29	floating
	8	Zeigefinger	1	0x28	low
	9	Mittelfinger	1	0x29	floating
	10	Ringfinger	2	0x28	low
	11	Kleiner Finger	2	0x29	floating

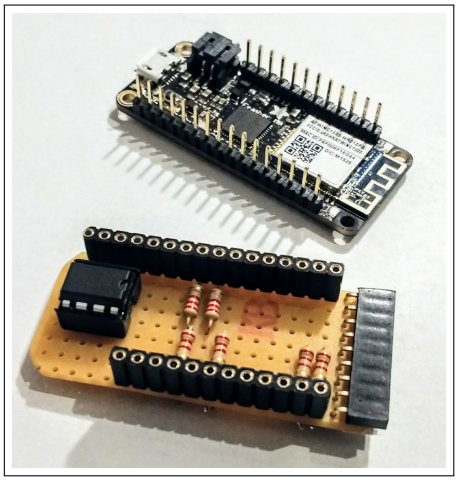
Tabelle 2: Zuordnung der IMUs zu Positionen an der Hand und den I^2C-Bussen und -Adressen.

Die Basis-IMU auf dem Handrücken lässt sich am einfachsten auf einem Handschuh platzieren. Auch vorstellbar ist zum Beispiel ein Clip, welcher sich von der Handinnenfläche über den Handrücken klemmt. Der erhöhte Produktionsaufwand und die erschwerte Verkabelung sprachen in unserem Projekt dagegen. Ein schmales Gummiband um die Hand kam für uns ebenfalls nicht infrage, da dieses stark gespannt sein müsste, um bei Bewegungen der Hand nicht zu verrutschen, der von dem Gummiband ausgeübte Druck würde dann jedoch den Tragekomfort beeinträchtigen.

Besonders wichtig für das Tippen sind die Fingerkuppen, daher haben wir uns dafür entschieden, diese und das dritte Fingerglied frei zu lassen. Also befestigen wir die Sensoren am zweiten Fingerglied. Um hier möglichst wenig Material zwischen die Finger zu bringen, das an den benachbarten Fingern reiben könnte, nehmen wir ein einfaches Elastikband, und bilden daraus Ringe. Die Sensoren kleben wir an der Oberseite auf. Für einen Prototyp ist diese Konstruktion stabil genug.

Wir verlöteten den `AD0`-Pin direkt an jedem zweiten Sensor mit `GND`. Die vier Kabel leiteten wir am Finger entlang und befestigten sie mit etwas Faden am fingerlosen Handschuh. So nah an den Fingerspitzen wie möglich bündelten wir die zusammengehörigen Kabel und verlöteten sie. Dadurch haben wir die Kabelmenge von 24 auf 8 Kabel reduziert und Gewicht eingespart. Die 8 übrigen Kabel werden wir zur Steckerleiste geführt, welche auf die Buchsenleiste des Shields passt. Abbildung 7a zeigt den fertigen Handschuh.

(a) Der fertige Handschuh



(b) Shield und Featherboard

Abbildung 7: Fotos der Hardwarekomponenten. Links ist der gesamte Handschuh zu sehen, inklusive der durch Gummiringe an den Fingern befestigten IMUs und der Basis-IMU auf dem Handrücken. Der Shield ist auf das Featherboard aufgesteckt und durch eine Steckverbindung mit dem vorderen Handschuhteil verbunden. Shield und Featherboard (einzeln im rechten Bild zu sehen) sind auf einer Manschette am Handgelenk befestigt.

4.7 Datenübertragung

Die erfassten Daten werden vom Mikroprozessor an den Host-PC übertragen. Dies soll über WLAN, also per Netzwerkprotokoll möglich sein. Zur einfachen Entwicklung des Systems ist es außerdem wünschenswert, den Handschuh per USB anzuschließen. Dadurch ist die Verbindung stabil, die Stromversorung sichergestellt und man muss keine WLAN-Zugangsdaten einpflegen.

Um sowohl die serielle USB-Schnittstelle als auch den netzwerkbasierten Modus unterstützen zu können, haben wir unser eigenes paketbasiertes Protokoll definiert. Mithilfe dieses Protokolls können wir die Daten bestimmen, welche übermittelt werden sollen. Dies hat den Vorteil, dass wir ausschließlich Daten von Interesse übermitteln und dadurch die Übertragungsgeschindigkeit verbessern.

Paketbasiert bedeutet in unserem Fall, dass wir jede Information in ein Datenpaket verpacken. Diese werden dann entweder seriell per USB übertragen, oder unabhängig voneinander per WLAN mit UDP gesendet.

Es gibt in unserem Anwendungsfall 4 verschiedene Arten von Paketen: Header, IMU Data, Debug Message und Key State Change.

Header Dieses Paket wird immer nach Abschluss des Bootvorganges gesendet und be-

ginnt mit einem „Magic header", der willkürlich gewählten, festen Bytefolge `49 D2 C8 CE`. Diese ist hilfreich, um zu erkennen, dass es sich tatsächlich um einen Datenstrom vom Mikroprozessor handelt. Alle nachfolgenden Bytes auf der seriellen Schnittstelle müssen dem in unserem Protokoll formulierten Format entsprechen.

Das Header-Paket beinhaltet die in Tabelle 3 aufgeführten Felder.

Offset	Länge	Wert	Beschreibung
0	4	`49 D2 C8 CE`	Magic header
4	1	`03`	Protokollversion
5	1		Anzahl IMUs
6	2		Wertebereich Accelerometer
8	2		Wertebereich Gyroskop
10	4		Aktuelle Prozessorzeit
14	18	`00 00 00 ...`	Padding auf 32 Byte Länge

Tabelle 3: Datenformat des Header-Pakets

Die Protokollversion gibt uns die Freiheit, das Protokoll zu erweitern und anzupassen, aber mit alten Datensätzen kompatibel zu bleiben.

Da die Anzahl der IMUs sowie deren Konfiguration hardwareseitig festgelegt wird, übertragen wir beides im Header-Paket an den Host, sodass dieser die Daten entsprechend korrekt interpretieren kann. Ebenso verhält es sich mit den konfigurierten Wertebereichen der Sensoren, welche in der späteren Berechnung der tatsächlichen Messwerte verwendet werden.

Alle Pakete, die nicht vom Typ Header sind, haben das in Tabelle 4 gezeigte Datenformat. Die aktuelle Prozessorzeit wird in allen Paketen übertragen, da es sich um zeitsensible Daten handelt, und die Prozessorzeit die genaueste Angabe ist, die wir zu einem Datensatz machen können.

Offset	Länge	Wert	Beschreibung
0	1	`C1`	Paketstart-Markierung
1	2		Pakettyp (s.u.)
2	4		Aktuelle Prozessorzeit
6	l		(Paketinhalt)
$6 + l$	1	`1C`	Paketende-Markierung

Tabelle 4: Datenformat aller Pakete, die nicht vom Typ Header sind.

Alle Ganzzahl-Werte werden in Little Endian gesendet. Dies ist in Netzwerkprotokollen zwar unüblich, da aber die Sensoren die Daten in dieser Byte-Reihenfolge bereitstellen, ersparen wir uns ein Umsortieren auf dem Mikroprozessor.

Debug Message (`0x01`) Dieser Pakettyp beinhaltet eine Nachricht zur Anzeige auf dem Host, zum Zwecke der Entwicklung, um zum Beispiel über die erfolgte Initialisierung einer IMU zu informieren. Der Paketinhalt besteht aus 2 Bytes Nachrichten-Code (arbiträr), 2 Bytes Textlänge, und dem Text der Nachricht, in ASCII codiert.

IMU Data (`0x02`) Das wohl wichtigste Paket beinhaltet die aktuellen Messwerte einer IMU. Der Paketinhalt ist 20 Bytes lang:

- 6 Bytes lineare Beschleunigung,
- 6 Bytes Winkelgeschwindigkeit und
- 8 Bytes Orientierung (Quaternion).

Wir senden keine Fließkommawerte, da die Sensoren Ganzzahlwerte bereitstellen, welche unter Berücksichtigung des eingestellten Wertebereiches zu Dezimalzahlen in Standardeinheiten umgerechnet werden können. Diese Umrechnung findet der Einfachheit halber erst auf dem Host-PC statt. Für die Vektoren errechnet sich der tatsächliche Wert durch folgende Formel:

$$V_{real} = \frac{V_{integer}}{2^{15} \cdot \text{range}}$$

Den Wertebereich (range) erhält man aus dem Header-Paket.

Quaternionen sind normalisiert und einheitslos, daher werden sie im Gegensatz zu den Vektoren ohne Verwendung eines Wertebereichs errechnet:

$$Q = \frac{Q_{integer}}{2^{14}}$$

Key State Change (`0x03`) Mit diesem Paket ist der Mikroprozessor in der Lage, eine Taste zu simulieren, etwa wenn Buttons am Handschuh befestigt sind. Wir haben diese Funktion in anfänglichen Experimenten verwendet, um die Funktionalität des Handschuhs zu testen. In Zukunft könnte man Funktionstasten in den Handschuh integrieren, etwa um zwischen verschiedenen Modi zu wechseln.

Das Paket beinhaltet neben den üblichen Feldern lediglich einen Key Code von 2 Bytes, sowie 1 Byte Tastenzustand (`0x01` für gedrückt, ansonsten `0x00`).

4.8 Datenverarbeitung

Die Software auf dem Host-PC ist komplett in Python implementiert. Hierfür kommen hauptsächlich folgende Bibliotheken zum Einsatz:

- `NumPy` (Walt, Colbert und Varoquaux, 2011) für die Datenverarbeitung mit Vektoren und mehrdimensionalen Matrizen

- `Theano` (Al-Rfou u. a., 2016) und `Lasagne` [8] zur Modellierung und Berechnung der neuronalen Netze
- `Matplotlib` (Hunter, 2007) zur Datenanalyse, Visualisierung der rohen und vorverarbeiteten Daten, Zeichnen der Lernfortschrittsgraphen
- `ROS` [9] zur Verknüpfung eigenständiger Komponenten über wohldefinierte Kommunikationskanäle, sogenannte „topics", dies beinhaltet `rosbag` als Datenspeicher für Lerndaten

Die Verwendung von NumPy ist unverzichtbar, da insbesondere Theano darauf aufbaut. Numpy ist eine weitverbreitete Bibliothek für die wissenschaftliche Datenverarbeitung in Python. Sie ist einfach zu benutzen, stark optimiert und auf die Verarbeitung hochdimensionaler Daten ausgelegt.

Theano baut hierauf auf und stellt Funktionen bereit, mit denen komplexe Rechenoperationen deklariert und dann auf unterschiedliche Arten ausgeführt werden können. Prinzipiell definiert man mit Theano zuerst den mathematischen Zusammenhang zwischen verschiedenen Symbolen und lässt sich dann eine Funktion generieren, welche diese Zusammenhänge auflöst und mitunter sehr komplexe Berechnungen durchführt. Theano kann konfiguriert werden, diese Funktion zum Beispiel just-in-time für eine oder mehrere CPUs, oder auch GPUs, zu kompilieren. Dadurch werden die Berechnungen sehr performant.

Lasagne ist eine Bibliothek, die es vereinfacht, neuronale Netze in Theano zu modellieren. Dadurch erspart man sich die komplexe Deklaration der mathematischen Modellierung eines Netzes mit Matrizen, und kombiniert nur einzelne Netzwerkschichten mit entsprechender Konfiguration zu einem Gesamtnetz. Dies beschleunigt die Entwicklung extrem und erlaubt es, sehr flexibel mit der Netzwerkkonfiguration zu experimentieren.

Wir verwenden ROS, das „Robot Operating System" aus verschiedenen Gründen. ROS ist eine Sammlung von Bibliotheken und Programmen, welche zur Entwicklung von Robotersystemen in der Forschung dienen. Den Kern bildet hierbei ein IPC-System (*interprocess communication*), das mithilfe sogenannter „topics" Kommunikation zwischen Prozessen über fest definierte Schnittstellen erlaubt.

Zunächst benutzten wir diesen Mechanismus der Kommunikation für jeden Verarbeitungsschritt, etwa auch zwischen Preprocessing und Sampling. Da dies jedoch einen gewissen Mehraufwand mit sich bringt und Verzögerungen einführt, entschieden wir uns dazu, nur noch bei einigen Verbindungen ROS-Topics einzusetzen. Am Ende nutzen wir ROS für die Übermittlung der Rohdaten sowie der tatsächlichen Tastenanschläge zum Preprocessing, und für die Ausgabe des Klassifikators im Anwendungsmodus. Alle anderen Verbindungen ersetzten wir durch Python Koroutinen (*coroutines*), welche ein ähnlich gekapseltes Programmieren ermöglichen wie die *topic callbacks* von ROS.

Ein großer Vorteil der Verwendung von ROS für die Rohdaten ist, dass wir *rosbag* verwenden können. Hierbei handelt es sich um ein Tool, das die Nachrichten eines oder mehrerer

Topics aufzeichnet und später abspielt. Dadurch konnten wir Lerndaten aufzeichnen, und später mit verschiedenen Parametern und Algorithmen anhand dieser Daten lernen. Wir spielen dabei die Aufzeichnung nicht ab, sondern verwenden die Python-Bibliothek von rosbag direkt, um die Daten auszulesen.

Es erwies sich als hilfreich, die gesamte Pipeline bis zum Schritt „Sampling" (vgl. Abbildung 4) vom Aufruf des Lernprozesses zu trennen. Das Zwischenergebnis, ein großes NumPy-Array mit allen Samples, speichern wir in eine eigene Datei. Wir schrieben uns einige Tools, um diese Samples analysieren zu können, einige dieser Visualisierungen zeigen wir im Kapitel 6.

Für andere Analysen waren die in ROS enthaltenen Tools sehr hilfreich. Insbesondere RViz, ein 3D-Visualisierungs-Programm für ROS, ist gut geeignet, um Fehler bei der Entwicklung zu erkennen. Hauptsächlich nutzte uns RViz beim Entwickeln der Vorverarbeitung der Quaternionen.

Für die Konfiguration des gesamten Prozesses verwenden wir eine zentrale Konfigurationsdatei. Diese wird im YAML-Format geschrieben und enthält alle Parameter, die für den Aufbau des Experiments relevant sind. Codebeispiel 1 zeigt eine beispielhafte Ausprägung dieser Konfigurationsdatei. Die Bedeutung der einzelnen Einstellungen wird in Konietzny, 2017 erläutert. Hervorheben möchte ich an dieser Stelle, dass wir für jedes Experiment eine eigene Konfigurationsdatei anlegen können, und somit jedes Experiment nachvollziehbar und wiederholbar wird. Alle Softwareteile laden beim Start die angegebene Konfiguration und benutzen die für den jeweiligen Verarbeitungsschritt relevanten Teile daraus. Ins jeweilige Verzeichnis des Experiments, neben die Konfigurationsdatei, werden auch alle Eingabedaten (ROS-Bag), Zwischenergebnisse (NumPy-Array) und Ausgaben (Lernfortschritt-Statistiken, generierte Graphen, ...) abgespeichert.

4.9 Einbindung als Tastatur ins Betriebssystem

Vorerst unterstützt unser System nur die Einbindung unter GNU/Linux, sodass die generierten Tastendrücke auch tatsächlich als solche benutzt werden können. Verwendet wird dafür das Input-Event-System *evdev*, welches über die dazugehörige Python-Bibliothek sehr einfach zu integrieren ist.

Diese Verwendung von `python-evdev` ist in Codebeispiel 2 skizziert. Eine ähnliche Integration wäre für andere Betriebssysteme selbstverständlich ebenfalls denkbar.

```yaml
# General / sampling
imu_ids: [0, 1, 2, 3, 4, 5]
key_codes: [20, 34, 48, 21, 35, 49, 22, 36, 50, 57]

# Sampling
sequence_length: 16
sequence_ratio: 0.5
sampling_mode: quat
sampling_rate: 25
base_imu_relative_average_rate: 0.2

# Learning / network
epochs: 0
batch_size: 0
training_ratio: 0.8
cost_function: mse
network_type: cnn2d
learning_rate: 0.002

# RNN
n_hidden: 20

# CNN
convolution_filter_count: 50
convolution_filter_size: [3, 3]
convolution_iterations: 2
convolution_dense_layer_units: [10]
convolution_deep_filters: true
```

Codebeispiel 1: Beispiel für eine Konfigurationsdatei. Wir verwenden diese, um die Initialisierung aller Softwarekomponenten zu beeinflussen. Dies beinhaltet die Vorverarbeitung, die mit Lasagne modellierte Netzwerkarchitektur, das Lernverhalten und den Anwendungsmodus.

```python
from evdev import uinput, ecodes

with uinput.UInput() as ui:
    # ...
    ui.write(ecodes.EV_KEY, event.code, event.pressed)
    ui.syn()
```

Codebeispiel 2: Beispiel für die Verwendung von `python-evdev` zur Emulation von Tastendrücken in einem Linux-System. Das `event` stammt hierbei vom ROS-Topic `/predicted_key_events`, welches die vom Klassifikator erkannten Tastendrücke übermittelt.

5 Maschinelles Lernen

In diesem Kapitel möchte ich kurz das Vorgehen und die Ergebnisse des maschinellen Lernens beschreiben, um eine vollständige Verständlichkeit des Gesamtprojektes zu ermöglichen. Ich werde jedoch nicht alle Details beleuchten, da dies außerhalb des Themas dieser Arbeit liegt. Die genaue Ausarbeitung dieser Aspekte sind Inhalt der Bachelorarbeit von Carolin Konietzny (2017), und dort nachzulesen.

5.1 Überblick

In Abbildung 4 habe ich bereits alle Komponenten des maschinellen Lernens in unserem Projekt vorgestellt. Angefangen im *Preprocessing* werden die Rohdaten von den Sensoren und der Tastatur vorverarbeitet. Aus diesem Datenstrom werden im *Sampling* sogenannte Samples extrahiert, also unabhängige Datensätze, welche jeweils ein Lernbeispiel beinhalten. Im *Learning* wird der gewählte Algorithmus angepasst, indem ihm Lernbeispiele vorgestellt werden. Das gelernte Modell wird gespeichert und kann dann in *Apply Network* direkt mit den vorverarbeiteten Daten angewendet werden, um eine Vorhersage (*prediction*) zu generieren.

Die wichtigste Entscheidung ist die Wahl des ML-Algorithmus. Da wir Datensätze mit den dazugehörigen tatsächlichen Ergebnissen (Tastaturanschläge) aufzeichnen, können wir einen Algorithmus des überwachten Lernens wählen. Wir entscheiden uns für ein neuronales Netzwerk, da diese in der Lage sind sehr komplexe Zusammenhänge zu erlernen, und wir flexibel das Netzlayout und die Parameter konfigurieren und optimieren können. Auch sind neuronale Netze, sobald sie trainiert wurden, sehr effizient anzuwenden, was in unserer Echtzeitanwendung wichtig ist.

Bei den neuronalen Netzen gibt es zahlreiche Varianten, die sich alle in ihrem Aufbau und damit in ihrer Funktion unterscheiden. Da wir Zeitfolgen lernen wollen, experimentierten wir zuerst mit einfachen rekurrenten Netzen (Elman, 1991). Hier traten jedoch Probleme auf, da unseren Daten unausgeglichenen sind: Der Anteil der Zeit, in der gerade eine Taste gedrückt wird, ist verhältnismäßig zum Rest sehr gering. Außerdem beobachten wir das *Vanishing Gradient Problem* (Hochreiter, 2004), also die Problematik, dass die vorderen Schichten im Netz nur sehr langsam angepasst werden, weil rekurrente Netze

sehr tief sind.

Stattdessen wählten wir ein *Convolutional Neural Network* (CNN; Lecun u. a., 1990). Dieser Netztyp wird häufig in der Bilderkennung eingesetzt, da er in der Lage ist, mehrdimensionale Muster zu unterscheiden. Dies geschieht durch eine Aneinanderreihung mehrerer Convolution-Pooling Schichtpaare. In der Convolution werden mithilfe von Filtern (ähnlich den Kerneln in der Bildverarbeitung) verschiedene Features extrahiert. Das markanteste lokale Feature wird im Pooling ermittelt, wobei die Größe des Datensatzes verringert wird. Somit reduziert sich nach einigen Wiederholungen die Datenmenge auf die „interessanten" Informationen aus den Originaldaten. Diese können dann mit einer oder mehreren vollständig verknüpften Netzwerkschichten klassifiziert werden.

5.2 Experimente

Zur Bewertung des Systems führten wir verschiedene Experimente durch, in denen wir die Brauchbarkeit der Hardware und des ML-Ansatzes testeten. Diese Experimente lassen sich in 4 Phasen einteilen:

Vorbereitung Zunächst entwickelten wir unter Verwendung von Dummy-Aufzeichnungen das gesamte Softwaresystem, um alle Einzelschritte der kompletten Pipeline und deren Zusammenspiel zu testen.

Phase 1: Erkennen einer Taste Hier zeichneten wir einen Datensatz auf, in welchem der Proband ausschließlich 2 Tasten, N und H drückte. Unser Ziel war es, den Datensatz zunächst manuell zu analysieren, und anschließend automatisch die Tastendrücke erkennen. Dabei wollten wir untersuchen, ob der gewählte ML-Algorithmus in der Lage war, anhand der Daten Tastendrücke zu erkennen und unter den benachbarten Tasten zu unterscheiden.

Phase 2: Differenzieren verschiedener Tasten In dieser Phase weiteten wir die Anzahl der Tasten von 2 auf 10 aus und benutzten somit auch mehrere Finger (Daumen, Zeigefinger und Mittelfinger). Die dabei aufgetretenen Probleme beschreibe ich in Abschnitt 6.2.

Phase 3: Flüssiges Schreiben Diese Phase haben wir noch nicht begonnen, sie wird darauf abzielen alle Tasten einer Hand ohne Pausen zwischen den Tastenanschlägen benutzen zu können.

5.3 Ergebnisse

In der ersten Phase bemerkten wir bereits, dass unser ursprünglicher Ansatz mit dem rekurrenten Netz nicht geeignet war, und wechselten zum CNN. Es kristallisierte sich heraus, dass ein CNN in der Lage ist, die Fingerbewegungen für das Drücken einer Taste

zu erkennen und in einem gewissen Maße auch der korrekten Taste zuzuordnen. In der ersten Phase erreichten wir eine Prädiktionsgenauigkeit von knapp 97%.

In der zweiten Phase wurde uns jedoch bewusst, dass die Aufgabe recht komplex ist und noch einiger Verbesserungen bedarf. Ein erstes Experiment in dieser Phase war weitaus weniger erfolgreich als erhofft. Das Netz war in der Lage, zwischen einigen der 10 Tasten zu differenzieren, allerdings wurden H , J , ␣ und ∅ („keine Taste gedrückt") nicht voneinander unterschieden. Wir begründeten dies damit, dass meine Finger in der Ruheposition auf diesen Tasten liegen, und die Bewegungen für diese Tasten nur relativ klein sind.

In einer zweiten Wiederholung des Experimentes mit gleichen Daten hatten wir mehr Glück, nach einiger Zeit wurden auch diese problematischen Tasten erkannt, wenn auch nicht mit der gleichen Genauigkeit wie die anderen Tasten. Insgesamt erreichte die Klassifikation eine Genauigkeit von 85%. Dieses Ergebnis ist recht zufriedenstellend, wenn auch noch ausbaufähig.

Für die nächste Phase, in welcher das flüssige Schreiben gelernt werden soll, müssen noch einige Verbesserungen durchgeführt werden. Vorschläge hierfür werden in der Arbeit von Carolin Konietzny (2017) geschildert.

6 Bewertung

In diesem Kapitel bewerte ich den nach unserem Design umgesetzten Datenhandschuh und das Gesamtsystem. Ich gehe dabei auf die Datenqualität ein, und stelle Probleme vor, die diese negativ beeinflusst haben. Danach bewerte ich, ob wir unsere Designziele aus Kapitel 2 erreicht haben, und erwähne mögliche Verbesserungen, die wir am System noch erkannt haben.

6.1 Datenqualität

Die vom Handschuh ermittelten Daten sind von ausreichender Qualität, wenn man an ihnen unterschiedliche Tastenanschläge erkennen kann. Nur dann ist auch ein ML-Algorithmus in der Lage, diese Erkennung ebenfalls durchzuführen.

Abbildung 8 zeigt die extrahierten, vorverarbeiteten Daten aus Phase 1. Auf der linken Seite wurde die Taste [N] und auf der rechen Seite die Taste [H] gedrückt. In den oberen Graphen ist der Pitch-Winkel, unten der Yaw-Winkel des rechten Zeigefingers relativ zur Handbasis aufgezeichnet. Entlang der x-Achse ist die Zeit vor und nach dem Tastenanschlag aufgeführt, zum Zeitpunkt des Tastenanschlages ist eine Hilfslinie im Graphen eingezeichnet. Jedes Sample (also jeder Tastendruck) tritt als einzelne Linie im Graphen auf. Anhand diesen Samples wird das Netz trainiert[1].

Die Unterschiede in den Bewegungsmustern für die verschiedenen Tastenanschläge sind hier gut zu erkennen. Wir stellen fest, dass sich alle Samples innerhalb des gleichen Graphen bis auf einige Ausnahmen sehr ähneln – dies ist wichtig für die Erkennung eines Musters. Die Amplitude des Ausschlages ist jeweils vergleichbar, einzig der y-Offset ist teilweise von Sample zu Sample unterschiedlich (gut erkennbar im N-Yaw-Graphen).

Abbildung 9 enthält viele verschiedene Graphen, welche aus den Aufzeichnungen für Phase 2 (siehe Kapitel 5) extrahiert wurden. Für einen Teil der Tasten[2] werden hier sowohl die Quaternionen, als auch die Accelerometerdaten über die Zeit aufgezeichnet. Der Tastenanschlag liegt nun bei drei Vierteln des gezeichneten Zeitfensters. Jeder Subgraph enthält

[1] Tatsächlich benutzen wir für das Training die Quaternionen, wir zeigen hier die Euler-Winkel, da diese einfacher zu verstehen sind.

[2] Phase 2 enthält 10 verschiedene Tasten, wir zeigen zur Übersicht nur 6 davon.

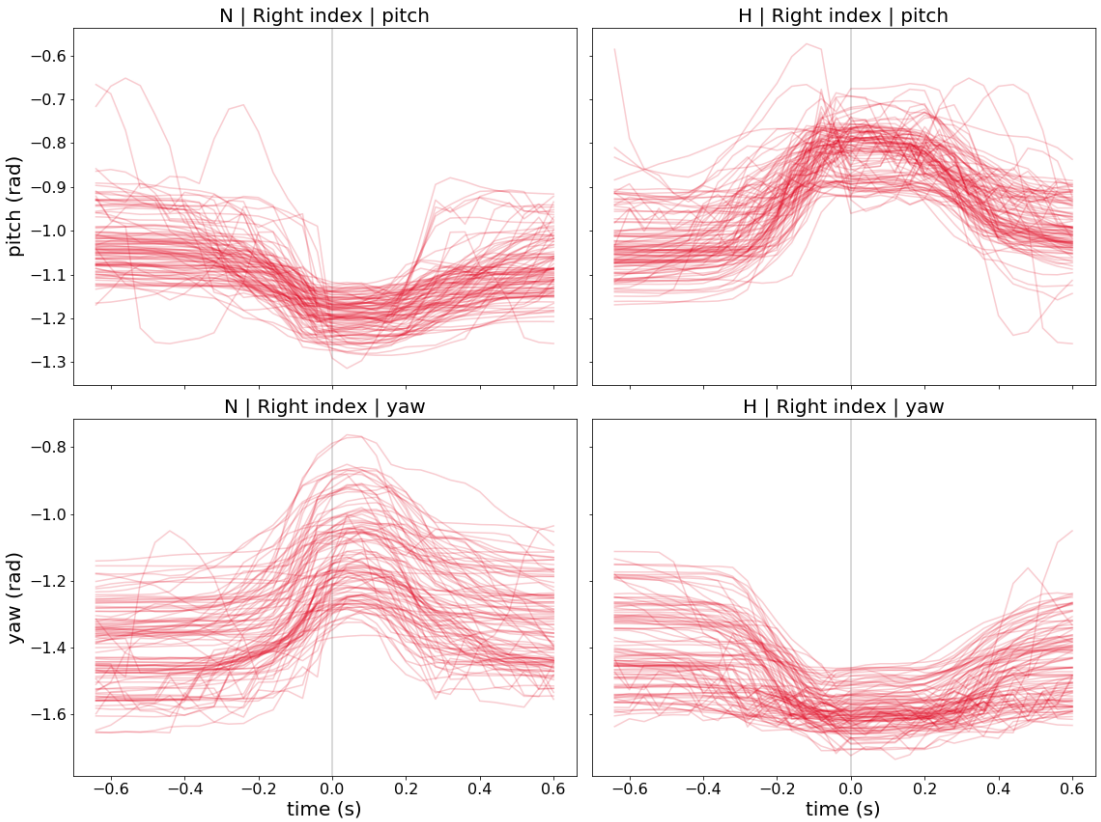


Abbildung 8: Mehrere Wiederholungen der Tasten N und H, überlagert zum Zeitpunkt des Drückens der Taste (Mittellinie). Gezeichnet sind zur einfacheren Visualiserung der relative Pitch- und Yaw-Winkel des rechten Zeigefingers. Der Zeigefinger liegt zwischen den Tastendrücken in Ruheposition auf der Taste H.

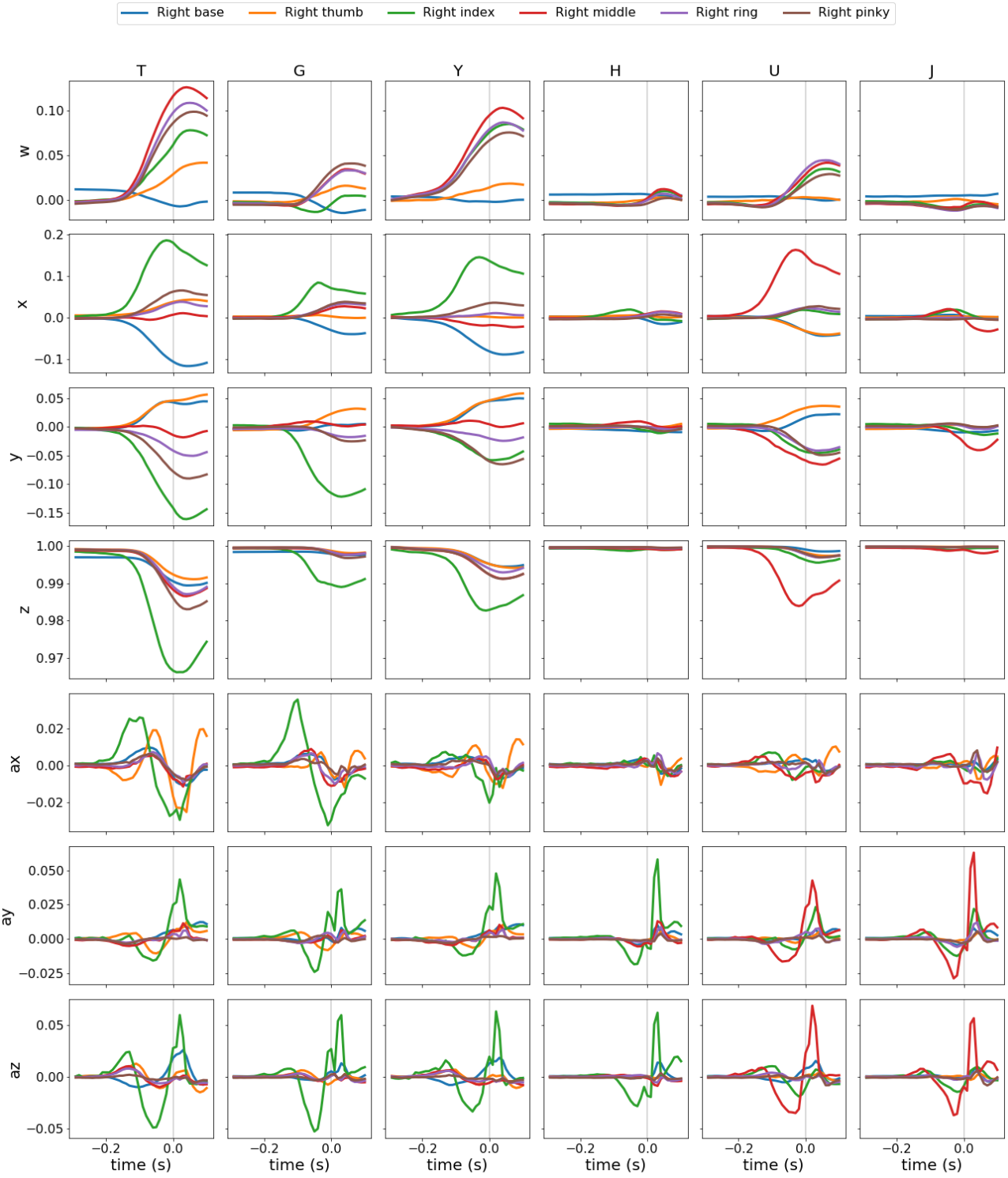


Abbildung 9: Mittel aller Samples verschiedener Tastenanschläge aus Phase 2, aufgeteilt nach Eingabewert und Sensor. Auf der x-Achse ist die Zeit vor und nach dem Tastenanschlag aufgetragen, die y-Achse zeigt den jeweiligen Eingabewert (v.o.n.u Quaternion w, x, y, z, Accelerometer x, y, z in m/s^2). Für jede Taste wurden etwa 150 Samples aufgezeichnet.

6 Kurven, jeweils eine für jeden der Sensoren der rechten Hand. Die einzelne Kurve ist dabei das Mittel aller für diesen Sensor, Tastenanschlag und Eingabewert ermittelten Samples.

Für jede der gezeigten Tasten ist in diesen Graphen ein eindeutiges Muster erkennbar. Beispielsweise sind sich die Muster der Tasten T und Y sehr ähnlich, unterscheiden sich jedoch im Ausschlag der z-Komponente der Quaternion des rechten Zeigefingers. Dass sich die Muster für diese Tasten grundsätzlich ähnlich sind, ist verständlich, da sie direkt benachbart sind, und der Zeigefinger für beide Tasten eine Bewegung nach oben links von der Ruheposition aus machen muss.

Ein anders Beispiel sind die Tasten H und J . Diese unterscheiden sich auf den ersten Blick nur geringfügig im Muster, es wird fast keine Änderung an der Orientierung der Sensoren festgestellt. Das liegt daran, dass in Ruheposition des Probanden der Zeige- und Mittelfinger auf diesen Tasten liegen, und somit kaum eine Bewegung durchgeführt werden muss. Bei genauerer Betrachtung stellt man jedoch fest, dass der y- und z-Ausschlag des Accelerometers von verschiedenen Sensoren stammt, das H wird also eindeutig mit dem Zeigefinger betätigt und das J mit dem Mittelfinger.

Wir sind grundsätzlich mit der Qualität dieser Daten sehr zufrieden und halten sie für geeignet, um damit eine Klassifizierung durchzuführen. Zu beachten ist aber, dass diese Daten schon vorverarbeitet worden sind. Einige der nachfolgend genannten Probleme wurden in der Vorverarbeitung dabei bereits behoben.

Es handelt sich bei den gezeigten um im Laborversuch gewonnene Daten. Reale Daten werden sicher noch ungenauer und uneindeutiger, wir hoffen hier mit weiteren Verbesserungen der Vorverarbeitungsmethoden gegensteuern zu können und mehr Eindeutigkeit zu erreichen.

6.2 Probleme

Im folgenden zeige ich die Probleme auf, welche wir im Laufe unserer Arbeit festgestellt haben.

6.2.1 Gyro clipping

Im Fusionsmodus, den wir für die Berechnung der Quaternionen aktiviert haben, steuert die BNO intern die Sensorkonfiguration, sodass sich diese nicht mehr von außen festlegen lässt, wie es etwa im Rohdaten-Modus der Fall wäre. Außerdem übernimmt der Fusionsalgorithmus die Kontrolle über die initiale Kalibrierung der Sensoren und führt Selbsttests und Echtzeitkalibrierungen durch.

Leider können wir einige Probleme mit dem Fusionsalgorithmus der BNO055 feststel-

len[3]. So ist das Gyroskop fest auf einen Wertebereich von $\pm 500\,\mathrm{deg/s}$ eingestellt, man kann hierauf keinen Einfluss nehmen. Dreht man den Sensor schneller als dieses Limit, wird der Wert abgeschnitten, und die Integration der Geschwindigkeit liefert eine falsche Orientierung. Diese schnellen Bewegungen treten bei flüssigem Tippen auf.

Wir konnten dies nachstellen, indem wir den Sensor senkrecht platzierten und von dort schnell in die Waagerechte drehten. Die berechnete Quaternion kommt dann nicht ganz der Drehung hinterher, es beinhaltet dann Fehler in der Pitch- und Roll-Komponente.

Eine mögliche Lösung hierfür wird zusammen mit dem folgenden Problem im nächsten Abschnitt diskutiert.

6.2.2 Heading drift

Ein weiteres Problem ist eine Ungenauigkeit des Headings, also der horizontalen Ausrichtung. Nach ruckartigen Bewegungen in nicht-waagerechter Orientierung verliert der Sensor seine Ausrichtung nach Norden. Wir vermuten, dass dies an der Konfiguration des Kalman-Filters liegt[4]. Dieser bewertet die Genauigkeit des Gyroskopes höher als die des Magnetometers, vermutlich da das Magnetometer von außen durch Veränderungen im Magnetfeld beeinflussbar ist. Die Abweichung betrifft nur das Heading, also die Komponente, die nicht durch den vom Accelerometer gemessenen Gravitationsvektor korrigiert werden kann.

Diese Abweichung in der Ausrichtung bleibt auch über längere Zeit bestehen, selbst bei Bewegung des Sensors. Dies erschwert lange Aufzeichnungen mit dem Handschuh, da die gemessenen Quaternionen zwischendurch „verrutschen" können und die Samples uneindeutig werden.

Um die Probleme mit dem Gyroskop und dem Drift zu lösen, probieren wir den Fusionsalgorithmus von Madgwick, Harrison und Vaidyanathan (2011) aus. Hierfür betreiben wir die BNO055 im Fusionsmodus, lesen aber auch die Rohdaten von Accelerometer, Gyroskop und Magnetometer aus. Die Rohdaten lassen wir vom Madgwick-Algorithmus in eine Orientierung umwandeln und vergleichen diese mit der Ausgabe des internen Fusionsalgorithmus, indem wir beide zu Euler-Winkeln konvertieren und über die Zeit zeichnen. Die Konvertierung in Euler-Winkel ermöglicht es uns, zwischen der horizontalen und vertikalen Ausrichtung zu unterscheiden.

Das Ergebnis ist in Abbildung 10 zu sehen. Wir erkennen, dass die Algorithmen grundsätzlich sehr ähnliche Ergebnisse liefern.

[3] Im Adafruit Forum gibt es eine lange Diskussion zu verschiedenen Problemen, die andere Benutzer mit der BNO055 erfahren haben, darunter auch die hier beschriebenen. `https://forums.adafruit.com/viewtopic.php?f=19&t=78459`

[4] Es ist nicht im Datenblatt der BNO055 spezifiziert, dass es sich bei dem Fusionsalgorithmus um einen Kalman-Filter handelt, Benutzer „Richart100" im „Chief Delphi"-Forum legt diese Vermutung jedoch nahe: „The on-board fusion algorithm, although unspecified, is likely a Kalman filter variant." (`https://www.chiefdelphi.com/forums/showthread.php?t=141100`, entnommen am 31.05.2017)

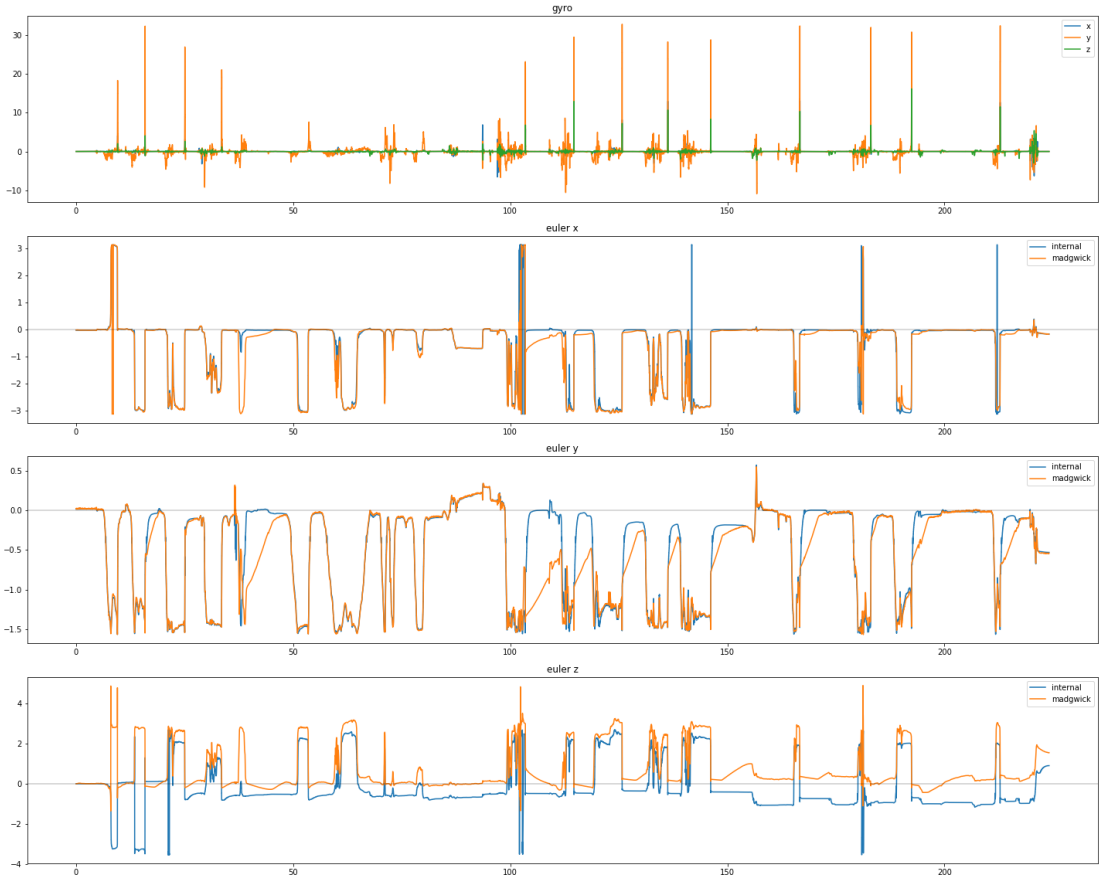


Abbildung 10: Interner Fusionsalgorithmus der BNO055 und Madgwick-Algorithmus im Vergleich. Gezeigt sind Rohdaten des Gyroskops (in rad/s) sowie Euler-Winkel der Orientierung (in rad).

Im unteren Graphen sehen wir, dass die horizontale Ausrichtung nach den Bewegungsphasen beim Madgwick-Algorithmus (orange) sich immer wieder dem Initialwert annähert. Der interne Algorithmus weist hier stattdessen den beschriebenen Drift auf – nach 200 Sekunden ist das Heading um knapp 1 rad zu niedrig, und korrigiert sich nicht wieder.

Der Madgwick-Algorithmus konvergiert jedoch viel langsamer zur korrekten vertikalen Orientierung, wenn er aufgrund schneller Rotation eine Differenz zwischen Gravitationsrichtung und gemessener Ausrichtung erkennt. Dies lässt sich besonders gut im dritten Graphen erkennen. Teilweise dauert es bis zu 10 Sekunden, bis sich der Wert an den des internen Algorithmus annähert.

Diese Konvergenz lässt sich durch Anpassung des „Gain"-Parameters beschleunigen, jedoch rät Madgwick davon ab, da es das Rauschen verstärkt. Das konnten wir leicht überprüfen, die Ausrichtung fängt mit höherem Gain-Wert an zu zittern. Dies ließe sich eventuell korrigieren, indem man den Wertebereich der Sensoren erhöht, und somit die Sensitivität verringert. Allerdings müssten wir dafür den Fusionsmodus deaktivieren und würden die automatische Echtzeit-Selbstkalibrierung des Sensors verlieren.

6.2.3 Robustheit

Unser Prototyp erwies sich nicht als besonders langlebig. Nach einigen Experimenten brachen teilweise die Lötstellen auf, da sie durch die Handbewegungen vielen Biegungszyklen ausgesetzt waren. Dies ist jedoch lediglich ein Problem des Prototyps, und keine allgemeine Einschränkung. Einige Zugentlastungen, eine einteilige Bauweise ohne Steckverbindungen, eingeschweißte Sensoren sowie mehradrige Kabelverbindungen würden die Robustheit des Handschuhs deutlich erhöhen.

6.2.4 Verwendung einer Laptop-Tastatur

In unseren Versuchen haben wir die in einen Laptop eingebaute Tastatur verwendet, welche im Vergleich zu Standard-Tastaturen relativ flach ist. Dadurch ist die Änderung der Orientierung für einige Tasten sehr gering, es handelt sich dabei um jene Tasten, auf denen die Finger in Ruheposition liegen (H, J, ␣). Der Algorithmus hat es dementsprechend schwer, diese Tasten richtig zu klassifizieren, und verwechselt sie ebenfalls mit der Klasse $\emptyset$. Erst in einer Wiederholung des Experiments lernte der Algorithmus, diese Tasten zu unterschieden werden, jedoch mit wesentlich geringerer Genauigkeit als die anderen Tasten. Vermutlich liegt dieses andere Ergebnis an einer zufällig besseren Initialisierung der Parameter im Netz.

Die Verwendung einer Tastatur mit genügend Tastenhub würde sicherlich bessere Ergebnisse erzielen. Außerdem erkennt man, wie in Abbildung 9 gesehen, an den Beschleunigungsdaten Unterschiede. Es wäre daher ebenfalls gut, den Klassifikator anzupassen, sodass er ebenfalls die Rohdaten des Accelerometers verwendet.

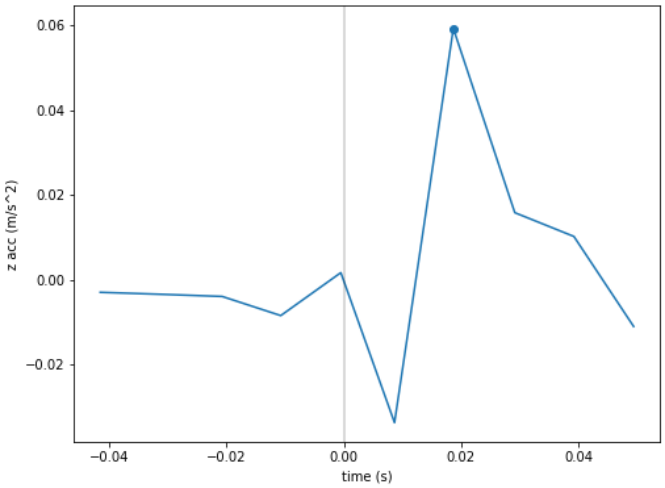


Abbildung 11: Messung der Verzögerung der IMU-Daten anhand der z-Komponente der Beschleunigung. Die IMU wird zusammen mit der Taste heruntergedrückt. Der dazugehörige Ausschlag des Accelerometers wird erst einige Zeit (hier 18.6 ms) nach dem Tastendruck (Zeitpunkt 0) registriert. Im Schnitt ergibt sich eine Verzögerung von rund 15 ms.

6.3 Sekundäre Designziele

In diesem Abschnitt gehe ich auf die im Kapitel 2 vorgestellten Ziele ein und bewerte, inwieweit wir diese Ziele erfüllt haben. Das primäre Designziel, charakteristische Werte zu extrahieren, haben wir bereits in Abschnitt 6.1 bewertet, und festgestellt, dass die Auswahl und Qualität der Daten unseren Anforderungen entsprechen.

6.3.1 Performance

Bezüglich der Performance haben wir unser Ziel, die angestrebte Datenrate von 100 Hz, fast erreicht. In unserem Messungen ermittelten wir eine durchschnittliche Datenrate von rund 90 Hz.

Die Verzögerung in der Datenübertragung von den Sensoren bis zum Host-PC ist ist relativ gering. In einem Experiment hierzu legten wir eine IMU auf die Tastatur und drückten sie zusammen mit der Taste herunter. Das Ergebnis dieses Experiments ist in Abbildung 11 zu sehen. Innerhalb von durchschnittlich 15 ms nach dem Tastenanschlag war der Ausschlag am Accelerometer sichtbar. Hiervon sind bis zu 10 ms durch die Datenrate bedingt, die restliche Verzögerung kommt durch die Übertragung zustande.

6.3.2 Geringe motorische Einschränkung

Der Handschuh schränkt beim Tippen kaum ein. Die Finger sind größtenteils frei beweglich. Man kann die Hand zur Faust formen, oder die Finger weit spreizen.

Bringt man die Finger dicht zusammen, reiben die Gummibänder an den Nachbarfingern. Dies ist jedoch beim Tippen nicht nötig, weil die Tasten einer Standard-Tastatur weiter auseinander liegen als die Finger breit sind. Auch im Alltag ist ein Zusammenbringen der Finger eine eher unnatürliche Haltung, daher werten wir diesen Nachteil als nicht besonders wichtig.

Der Handschuh ist leicht (ca. 70 g mit Akku), sodass die Hand auch bei längerer Benutzung nicht ermüdet. Weil der Handschuh aus einfachem, dünnem Baumwollstoff gefertigt ist, schwitzt man auch nicht darin. Die Gummiringe schränken die Durchblutung der Finger nicht ein, sitzen aber fest genug auf dem Finger. Hierfür ist es natürlich wichtig, dass die Fingerringe auf den Fingerumfang des Benutzers angepasst sind.

Der Handschuh ist alltagstauglich, übliche Büroaktivitäten zum Beispiel lassen sich mit Handschuh uneingeschränkt durchführen. Getestet wurden zum Beispiel das Öffnen von Türen, Greifen und Verwenden einer Kaffeetasse und die Benutzung von Stiften. Etwas weniger geeignet ist der Handschuh bei sehr feinmotorischer Interaktion mit Gegenständen, etwa wenn man einen Reißverschluss schließen möchte – dies liegt jedoch daran, dass der Prototyp nicht besonders robust ist. Dank der freiliegenden Fingerkuppen sind feine Manipulationen dieser Art trotzdem möglich.

6.3.3 Haltungsunabhängigkeit

Dieses Ziel haben wir bisher nur in beschränktem Maße erreicht. Prinzipiell ist die Benutzung in verschiedenen Ausrichtungen möglich, da die relative Orientierung der Sensoren ermittelt wird. Allerdings ist die BNO055 in der Senkrechten nicht so genau wie in der Waagerechten, vermutlich liegt das am internen Fusionsalgorithmus. Die ermittelte Orientierung springt hierbei teilweise um einige Grad.

Dennoch war es uns möglich, mit den Händen in der Luft oder auf dem Tisch zu tippen. Löst man also das Problem mit den Sensoren, erfüllt das Systemdesign diese Anforderung.

6.3.4 Prototyping-geeignete Software

Unser Softwaredesign ist sehr gut geeignet, um damit einen Prototyp zu entwickeln. Dank der flexiblen Konfiguration können wir ohne bedeutenden Aufwand alle wichtigen Parameter ändern. Auch grundlegendere Aspekte, wie den verwendeten Typ von neuronalem Netzwerk, stellen wir in der Konfigurationsdatei ein. Dadurch bleibt jedes Experiment nachvollziehbar und wiederholbar. Dies ist sehr wichtig, insbesondere wenn das gleiche Experiment mit anderen Daten wiederholen möchte oder Bugs behoben hat.

Die Verwendung von ROS war ebenfalls eine gute Entscheidung. ROS-Bags sind eine sehr einfache Methode, Zeitreihen aufzuzeichnen und wieder abzuspielen oder direkt auszulesen. Mit dem Topic-System bietet ROS darüber hinaus eine flexible Möglichkeit, Schnittstellen zwischen Programmen zu definieren und einzelne Verarbeitungsschritte unabhängig zu implementieren. Unabdingbar bei der Entwicklung eines solchen Systems ist die Möglichkeit, die erfassten und verarbeiteten Daten zur Fehlerbehebung zu visualisieren – ROS und die ROS-Community liefern hierfür zahlreiche Programme.

6.3.5 Ausbaufähigkeit zu marktfähigem Produkt

Der Prototyp ist, wie bereits bemerkt, nicht sonderlich robust. Ein industriell gefertigtes Produkt könnte diese Schwächen doch in unseren Augen gut überwinden, mit entsprechendem Aufwand wäre eine hochwertige Produktion sicher möglich. Durch die Verwendung von IMUs als Sensoren haben wir hierzu die Grundlage geschaffen, da IMUs in sich abgeschlossen und robust sind und ohne bewegliche Teile auskommen.

Die Bauteile für den Prototyp waren in der Anschaffung sehr teuer. Die Sensoren mit jeweils 24,00 € machen hierbei einen Großteil der Kosten aus, insgesamt kommen wir auf etwa 200 €. In industrieller Massenproduktion wäre man nicht auf die Verwendung von verfügbaren Breakout-Boards angewiesen und könnte die Bauteile selbst vom Hersteller beziehen und auf einer speziell gefertige Leiterplatte anbringen. Wir haben die Kosten für den Prototyp und das Endprodukt geschätzt und in Tabelle 5 aufgeführt. Demnach könnte ein „consumer ready" Handschuh pro Stück unter 100 €, bzw. 200 € für eine komplette „Tastatur", kosten. Eine hochwertige ergonomische Tastatur liegt heutzutage bei einem vergleichbaren Preis.

Dass man mit diesem Hardwareaufbau ein marktfähiges Produkt herstellen kann, zeigt auch der Hi5 VR Glove [5].

6.4 Verbesserungsmöglichkeiten

Für die Zukunft sehe ich noch einigen Raum für Verbesserungen am Design und der Umsetzung des Systems.

Selbstverständlich wäre es gut, die Probleme bei der Datenerfassung zu beheben. Evaluieren könnte man dafür andere IMUs, welche jeweils mit ihrem proprietären Fusionsalgorithmus ausgeliefert werden. Zusätzlich wäre es interessant, die BNO055 selbst zu kalibrieren, an der Konfiguration der Sensoren zu experimentieren, und dann einen verfügbaren Fusionsalgorithmus, zum Beispiel von Madgwick, Harrison und Vaidyanathan (2011), oder einen Kalman-Filter auszuprobieren. Anstatt dieses Problem in der Datenerfassung zu lösen, wäre es ebenfalls denkbar, eine geeignete Fehlererkennung in der Vorverarbeitung zu implementieren und die Messwerte anhand des ermittelten Fehlers anzupassen. Hier besteht großes Optimierungspotenzial, sicherlich wäre es mit einigem

	Kosten in €	
Komponente	Prototyp	Produktion
IMUs	6×24.00	6×7.00
Mikroprozessor (Cortex M0)	42.00	2.60
WLAN-Modul (ATWINC1500)	$*$	5.00
Sekundärbauteile (Widerstände etc.)	$*$	10.00
Akku	5.00	5.00
Leiterplatte	–	10.00
Verkabelung, Steckverbindungen	10.00	10.00
Handschuh	0.20	5.00
Summe	201.20	84.60

$*$ bereits im Mikroprozessor enthalten

Tabelle 5: Kostenübersicht für die Herstellung eines Handschuhs. Preise für IC-Bausteine wurden ermittelt von `http://www.mouser.de/` am 26. Mai 2017, restliche Kosten sind Schätzwerte.

Aufwand möglich, wesentlich genauere Daten ohne Clipping oder Drift zu erhalten. Die Evaluation dieser Lösungsansätze ist jedoch aufgrund des hohen Aufwandes nicht Teil unsere Bachelorarbeiten.

Der Handschuh an sich ließe sich robuster gestalten, auch in Form eines Prototyps. Für einen hochwertigerer Handschuh könnte man die Finger-Ringe mit dem Handschuh zu verbinden, und eventuell auch das am Handgelenk befestigte Teil in den Handschuh integrieren. Damit wäre der Handschuh einteilig und hätte weniger anfällige Verbindungsstellen.

Das System unterstützt bisher nur den Betrieb entweder über USB, oder über WLAN. Diese Konfiguration muss fest in der Firmware hinterlegt werden, ein Umschalten im Live-Betrieb ist dadurch nicht möglich. Diese Funktionalität, sowie eine Verbesserung der Robustheit der Übertragung (eine Übertragungsunterbrechung erfordert zurzeit einen Neustart des Prozessors) könnten in der Zukunft implementiert werden. Besonders interessant wird dies, wenn man hauptsächlich den WLAN-Modus verwendet, und die Verbindung bei Bewegung im Gebäude unterbrochen wird.

Natürlich sind ebenfalls Verbesserungen am Lernalgorithmus vonnöten, um ein konkurrenzfähiges Produkt zu erschaffen. Weitere Forschung wird die angestrebte Zuverlässigkeit und Genauigkeit erhöhen. Einige Ideen und Ansätze hierfür werden in der Arbeit von Carolin Konietzny (2017) aufgezeigt.

Außerdem wäre es schön, ein einfaches Benutzerinterface zur Konfiguration des Hand-

schuhs und zum Training des ML-Algorithmus zu entwickeln. Hiermit könnte man die WLAN-Zugangsdaten festlegen und Profile für verschiedene Benutzer oder Anwendungen verwalten.

7 Fazit

In diesem Kapitel fasse ich meine Arbeit zusammen und gebe einen Ausblick auf weitere mögliche Forschungsthemen.

7.1 Zusammenfassung

In der vorliegenden Bachelorarbeit habe ich den Entwurf eines Systems zur Aufzeichnung der charakteristischen Handbewegungen beim Tippen sowie der dazugehörigen Tastatureingaben erläutert, und die Qualität dieses Entwurfes bewertet. Ich bin außerdem auf die Nutzbarkeit eines solchen Systems in Verbindung mit einem Verfahren des maschinellen Lernens als Alternative zur klassischen Tastatur eingegangen.

Zunächst stellte ich die Kriterien auf, welche das System erfüllen müsse, und erläuterte deren Relevanz. Besonderer Fokus lag hierbei auf der Auswahl der für die Handbewegung beim Tippen charakterischen Werte, und der Art und Weise, wie diese ermittelt werden könnten.

Im State of the Art stellte ich relevante Projekte vor, welche ähnliche und verwandte Themengebiete bereits erforscht haben.

Den Kern der Arbeit bildete das Systemdesign. Ich stelle vor, warum wir uns für die Verwendung von IMUs als Sensoren im Allgemeinen und für welche genauen Hardwarekomponenten im Speziellen wir uns entschieden haben. Außerdem behandelte ich die Verbindung der Komponenten, die Befestigung der Sensoren an der Hand mithilfe eines Handschuhs, und den Aufbau des dazugehörigen Softwaresystems.

Nach einer kleinen Zusammenfassung der Arbeit von Konietzny (2017), welche das maschinelle Lernen in unserem Projekt behandelt, bewertete ich unseren Ansatz in Bezug auf das Systemdesign und die Umsetzung des Hardwareteils. Ich stellte aufgetretene Probleme dar, sowie wie mögliche Lösungsansätze für diese.

Zuletzt möchte ich einen kleinen Ausblick geben, welche Möglichkeiten sich aus dem vorgestellten System ergeben.

7.2 Ausblick

Wir sehen viele Möglichkeiten für eine Weiterentwicklung dieses Projektes.

Wir können uns vorstellen, dass ein solches System die Entwicklung weg vom Layout einer traditionellen Tastatur ermöglicht. Das System könnte *online-learning* implementierten, und somit lernen, mit Veränderungen in den Bewegungen beim Tippen umzugehen. Verwendet man dann dieses System über längere Zeit, wird der Benutzer automatisch seine Bewegungen verändern, und irgendwann wird sich ein für ihn optimiertes Muster eingespielt haben, welches er verwenden kann, und welches das System versteht. Hier sehen wir großes Potential in allen Arten der Kommunikation mit Computern, sei es Texteingabe oder andere Steuerung.

Im weiteren sehen wir Möglichkeiten für den Einsatz unseres Systems für die Steuerung von Robotern. Hierfür müsste ein entsprechendes kinematisches Handmodell benutzt werden, um aus den Sensordaten möglichst akkurat die Handkonfiguration abzuleiten. Auch mit ML könnte hier gearbeitet werden – komplexe Bewegungen, die der Mensch im Laufe seines Lebens erlernt hat, wie etwa das Greifen von Gegenständen, könnten einem Robotor „vorgemacht" werden, sodass dieser hieraus ebenfalls lernen kann.

Auch in der Videospielebranche ist eine Anwendung denkbar. Wie wir feststellen konnten, ist dort die virtuelle Realität ein aktuelles Thema. Hier hat sich noch keine Steuerungsmethode durchsetzen können, und ein Datenhandschuh könnte die Brücke schlagen zwischen echter und digitaler Welt, um mühelos und intuitiv mit virtuellen Gegenständen interagieren zu können.

Wir haben gezeigt, dass unsere Vision nicht unrealistisch ist, und mit entsprechendem Aufwand umsetzbar scheint. Wir sind uns sicher, dass viele Menschen von einem auf unseren Prinzipien basierenden Texteingabegerät profitieren könnten, und damit besser, effizienter und gesünder mit Computern zu interagieren.

Anhang

1 Quellenverzeichnis

Al-Rfou, Rami u. a. (2016). „Theano: A Python framework for fast computation of mathematical expressions“. In: *CoRR* abs/1605.02688.

Bosch Sensortec (2016). *BNO055. Intelligent 9-axis absolute orientation sensor*. Rev. 1.4. Bosch Sensortec.

Cavallo, F. u. a. (2013). „Preliminary evaluation of SensHand V1 in assessing motor skills performance in Parkinson disease“. In: *Rehabilitation Robotics (ICORR), 2013 IEEE International Conference on*, S. 1–6.

Dipietro, Laura, Angelo M Sabatini und Paolo Dario (2008). „A survey of glove-based systems and their applications“. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.4, S. 461–482.

Ellithorpe, Jonathan und Pearl Tan (2012). *The Learning Keyboard. Using the Xbox Kinect to Learn User Typing Behavior*.

Elman, Jeffrey L (1991). „Distributed representations, simple recurrent networks, and grammatical structure“. In: *Machine learning* 7.2-3, S. 195–225.

Gerr, Fred, Michele Marcus und Carolyn Monteilh (2004). „Epidemiology of musculoskeletal disorders among computer users: lesson learned from the role of posture and keyboard use“. In: *Journal of Electromyography and Kinesiology* 14.1, S. 25–31.

Hochreiter, Sepp (2004). „The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions.“ In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.2, S. 107–116.

Hunter, J. D. (2007). „Matplotlib: A 2D graphics environment“. In: *Computing In Science & Engineering* 9.3, S. 90–95.

Konietzny, Carolin (2017). *Prototyp für eine virtuelle Tastatur basierend auf IMUs und maschinellem Lernen – Anwendung*.

Lane, David M u. a. (2005). „Hidden costs of graphical user interfaces: Failure to make the transition from menus and icon toolbars to keyboard shortcuts“. In: *International Journal of Human-Computer Interaction* 18.2, S. 133–144.

Lara, Oscar D und Miguel A Labrador (2013). „A survey on human activity recognition using wearable sensors.“ In: *IEEE Communications Surveys and Tutorials* 15.3, S. 1192–1209.

Lecun, Y. u. a. (1990). „Handwritten Digit Recognition with a Back-Propagation Network“. In: *Advances in Neural Information Processing Systems 2*. Hrsg. von D. S. Touretzky. Morgan Kaufmann, S. 396–404.

Li, Kang u. a. (2011). „Development of finger-motion capturing device based on optical linear encoder“. In: *Journal of rehabilitation research and development* 48.1, S. 69.

Lin, B. S. u. a. (2014). „Data Glove Embedded with 6-DOF Inertial Sensors for Hand Rehabilitation“. In: *2014 Tenth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, S. 25–28.

Lobo, Jorge und Pedro Trindade (2013). „Interactive demonstration of InerTouchHand – iTH a glove device with distributed inertial sensors and vibro-tactile feedback“. In: *2013 2nd Experiment@ International Conference (exp.at'13)*, S. 178–179.

Madgwick, S. O. H., A. J. L. Harrison und R. Vaidyanathan (2011). „Estimation of IMU and MARG orientation using a gradient descent algorithm“. In: *2011 IEEE International Conference on Rehabilitation Robotics*, S. 1–7.

Taylor, Jonathan u. a. (2016). „Efficient and precise interactive hand tracking through joint, continuous optimization of pose and correspondences“. In: *ACM Transactions on Graphics (TOG)* 35.4, S. 143.

Walt, Stéfan van der, S. Chris Colbert und Gaël Varoquaux (2011). „The NumPy Array: A Structure for Efficient Numerical Computation“. In: *Computing in Science & Engineering* 13.2, S. 22–30.

Wheeler, K. R. und C. C. Jorgensen (2003). „Gestures as input: neuroelectric joysticks and keyboards“. In: *IEEE Pervasive Computing* 2.2, S. 56–61.

Yao, Rui u. a. (2017). „Efficient Dense Labeling of Human Activity Sequences from Wearables using Fully Convolutional Networks“. In: *CoRR* abs/1702.06212.

2 Online-Quellen

[1] Waalboer, Juerd (2017). *Typing Test @ AOEU*. URL: `http://typing-speed-test.aoeu.eu/` (besucht am 12. 07. 2017).

[2] Rowberg, Jeff (2015). *Keyglove*. URL: `http://www.keyglove.net/` (besucht am 01. 05. 2017).

[3] Seidle, Nathan (2006). *Wii-mote guts*. URL: https://www.sparkfun.com/tutorials/43 (besucht am 03. 06. 2017).

[4] iFixit (2016). *HTC Vive Teardown*. URL: https://de.ifixit.com/Teardown/HTC+Vive+Teardown/62213 (besucht am 03. 06. 2017).

[5] Ltd., Noitom (2017). *Hi5 VR Glove*. URL: http://hi5vrglove.com/ (besucht am 01. 05. 2017).

[6] Apotact Labs (2017). *Gest*. URL: https://gest.co/ (besucht am 30. 04. 2017).

[7] Adafruit Industries (2017). *Adafruit Feather M0 WiFi - ATSAMD21 + ATWINC1500*. URL: https://www.adafruit.com/product/3010 (besucht am 03. 06. 2017).

[8] Battenberg, Eric u. a. (2015). *Lasagne*. URL: http://lasagne.readthedocs.io/en/latest/ (besucht am 12. 06. 2017).

[9] Open Source Robotics Foundation, Inc. (2017). *ROS.org*. URL: http://www.ros.org/ (besucht am 30. 04. 2017).

3 Abbildungsverzeichnis

4 Tabellenverzeichnis

5 Datenträger-Verzeichnis

Dieser Arbeit ist ein digitaler Datenträger beigelegt. Darauf finden Sie die Arbeit in digitalem Format, sowie begleitende Materialen:

- diese Arbeit, sowie die Arbeit von Carolin Konietzny
- den Quelltext der Handschuh-Firmware
- den Quelltext der Softwarekomponenten
- diverse Tools und Skripte zur Datenanalyse
- das Exposee, welches unserem Projekt zugrunde liegt
- die Folien unseres Kolloquiumvortrags (in englischer Sprache)
- Schaltplan und Leiterbahnlayout des „Shields"
- Fotos und Filme des fertigen Handschuhs

Nicht enthalten sind aus Speicherplatzgründen die Aufzeichnungen der Experimente. Diese können bei Interesse vom Projektverzeichnis auf Github heruntergeladen werden:

`https://github.com/imutype/bachelor`

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 und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Hamburg, den 12.06.2017 Paul Bienkowski

Veröffentlichung

Ich stimme der Einstellung der Arbeit in die Bibliothek des Fachbereichs Informatik zu.

Hamburg, den 12.06.2017 Paul Bienkowski