

Entwicklung eines Systems für die Mobile Sensordatenerfassung zur Erkennung von Ganzkörpergesten in Echtzeit

**Development of a system for mobile sensor data acquisition to recognize full body gestures
in real time**

Bachelor-Arbeit

Pascal Dornfeld

KOM-B-0645



**TECHNISCHE
UNIVERSITÄT
DARMSTADT**

Fachbereich Elektrotechnik
und Informationstechnik
Fachbereich Informatik (Zweitmitglied)
Fachgebiet Multimedia Kommunikation
Prof. Dr.-Ing. Ralf Steinmetz

Entwicklung eines Systems für die Mobile Sensordatenerfassung zur Erkennung von Ganzkörpergesten in Echtzeit

Development of a system for mobile sensor data acquisition to recognize full body gestures in real time

Bachelor-Arbeit

Studiengang: Informatik

KOM-B-0645

Eingereicht von Pascal Dornfeld

Tag der Einreichung: 23. Juli 2019

Gutachter: Prof. Dr.-Ing. Ralf Steinmetz

Betreuer: Philipp Müller und Augusto Garcia-Agundez

Technische Universität Darmstadt

Fachbereich Elektrotechnik und Informationstechnik

Fachbereich Informatik (Zweitmitglied)

Fachgebiet Multimedia Kommunikation (KOM)

Prof. Dr.-Ing. Ralf Steinmetz

Erklärung zur Abschlussarbeit gemäß § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Pascal Dornfeld, die vorliegende Bachelor-Arbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Bachelor-Arbeit stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

Darmstadt, den 23. Juli 2019

Pascal Dornfeld



Inhaltsverzeichnis

1 Einleitung	3
1.1 Motivation	3
1.2 Problemstellung und Beitrag	4
1.3 Überblick der Gliederung	5
2 Hintergrund	7
2.1 Sensor	7
2.1.1 Beschleunigungssensor	7
2.1.2 Gyrosensor	7
2.2 Chipkommunikation	8
2.2.1 I2C	8
2.2.2 SPI	9
2.3 Endgerätekommunikation	9
2.4 Lithiumknopfzelle	10
3 Aktueller Stand	11
3.1 Bestehende Lösungen	11
3.1.1 Texas Instruments CC2650STK	11
3.1.2 aconno ACNSENSA	11
3.1.3 Arduino Primo Core	12
3.2 Zusammenfassung und verwandte Arbeiten	12
4 Design und Planung	13
4.1 Anforderungen	13
4.2 Energieversorgung	13
4.3 Datenübertragung zwischen Wearable und Endgerät	13
4.4 Wahl der Hauptkomponenten	15
4.4.1 MCU	15
4.4.2 IMU	15
4.5 Zusammenwirken der Komponenten	16
4.6 BLE Schnittstelle	17
4.7 Applikation des Android Smartphones	17
5 Implementierung	19
5.1 Wearable Programm	19
5.1.1 Entwicklungsumgebung	19
5.1.2 Programmaufbau	19
5.1.3 BLE-Service	20
5.1.4 IMU Integration	21
5.2 Wearable Hardwareumsetzung	25
5.3 Android App	26
5.4 Zusammenfassung	29
6 Evaluation	31
6.1 Goal and Methodology	31

6.2 Evaluation Setup	31
6.3 Evaluation Results	31
6.4 Analysis of Results	31
7 Conclusions	33
7.1 Summary	33
7.2 Contributions	33
7.3 Future Work	33
7.4 Final Remarks	33
Literaturverzeichnis	33

Zusammenfassung

The abstract goes here...



1 Einleitung

1.1 Motivation

Die Nachfrage nach Wearables steigt kontinuierlich. Abbildung 1.1 zeigt, dass sich der Absatz in den letzten vier Jahren versechsfacht hat. Wearables sind Geräte, die am Körper getragen werden, um zum Beispiel mithilfe von Sensoren Daten zu erfassen [Ben19].

Ein weit verbreiteter Anwendungsfall ist die Herzfrequenzmessung beim Sport mit einem Fitnessarmband. Durch das Auswerten dieser Daten kann die Trainingsintensität in Echtzeit an die Person angepasst und somit die Effektivität gesteigert werden.

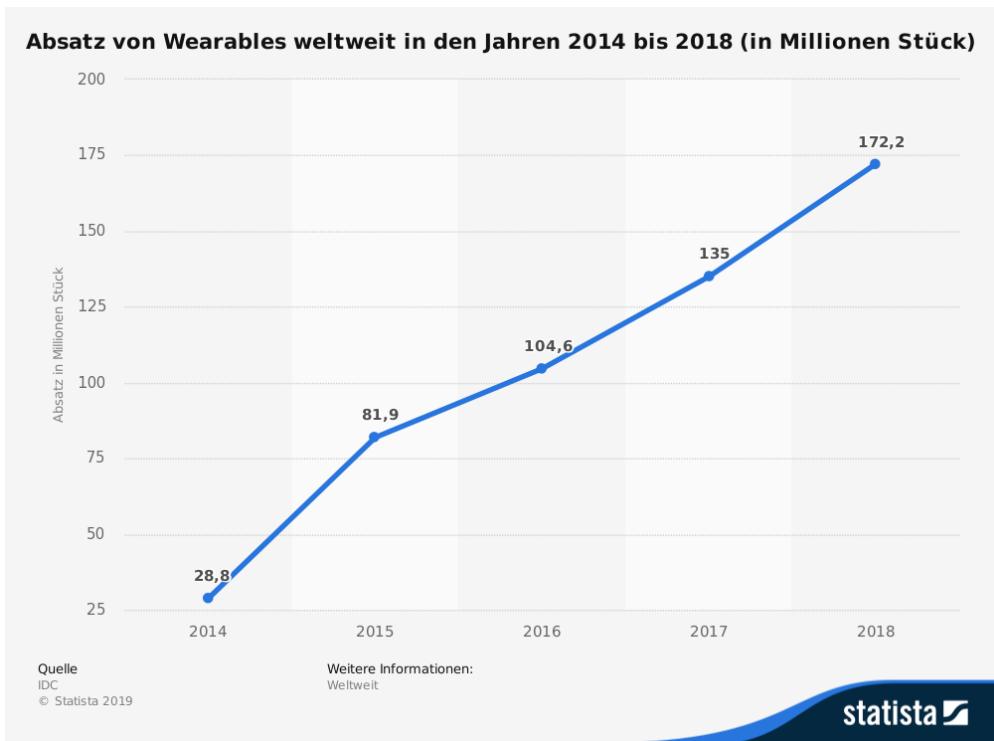


Abbildung 1.1: Absatz von Wearables [Int19]

In dieser Arbeit hingegen wird ein Wearable entworfen, das die Position und Rotation von Gelenken erfasst. Auf der Analyse dieser Daten aufbauend können dann weiterführende Anwendungsfälle entwickelt werden. Folgende sind zum Beispiel interessante Anwendungen:

- Das Erkennen von falsch ausgeführten Übungen beim Sport oder falscher Haltung beim Sitzen oder Stehen im Alltag. Dadurch können negative gesundheitliche Folgen vermieden werden.
- Eine Lösung für mobiles Motion Capturing zum Erstellen von Animationen in Filmen oder Videospielen. Mit der Anzahl der Wearables kann die Auflösung der Bewegung proportional zum Preis skaliert werden.
- Ein Echtzeitsystem zur Übersetzung von Gestensprache in gesprochene Sprache zur Kommunikation von stummen Menschen.
- Für die Bewegungserkennung in Videospielen und Virtual Reality. Abbildung 1.2 zeigt, dass die Nintendo Wii die bisher fünftmeist-verkaufte Konsole ist (Stand: Ende 2018) und damit die Geschichte der Videospielkonsolen prägt. Die Fernbedienung übernimmt dabei vor allem eine Funktion: Das

Tracken der Handposition. Zusätzlich kann man mit dem Nunchuk per Kabel einen zweiten Sensor für die andere Hand anschließen. Würde stattdessen eine Konsole mit dem hier entwickelten Wearable entworfen werden, würden die Hände beim Spielen frei bleiben, das Kabel zwischen den Händen wegfallen und es könnten weitere Gliedmaßen getrackt werden. VR-Headsets und die Xbox Kinect nutzen dagegen Kameras zur Ganzkörpergestenerkennung. Diese Lösung ist dagegen nicht mobil einsetzbar und anfällig auf Störeinflüsse wie fehlender Sichtkontakt.

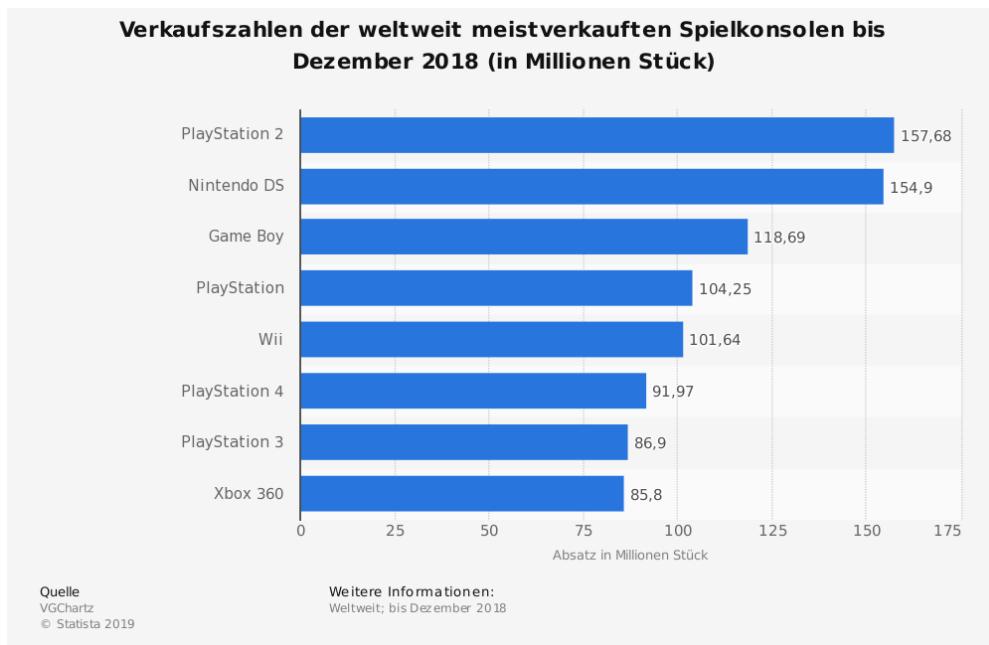


Abbildung 1.2: Verkaufszahlen der weltweit meistverkauften Spielkonsolen [VGC19]

Das Wearable kann also in vielen Bereichen Anwendung finden. Von Medizin über Kommunikation zur Produktivität und Unterhaltung.

1.2 Problemstellung und Beitrag

Folgende Ziele ergeben sich für diese Arbeit:

1. Um eine gute Mobilität zu gewährleisten, muss das Wearable klein und leicht sein, damit es auch beim Sport praktikabel bleibt.
2. Die Datenübertragung muss ausreichend schnell sein um die Datenrate zu übertragen, die für Gestenerkennung nötig ist.
3. Die Energiekapazität und der Energieverbrauch müssen verhältnismäßig zueinander abgestimmt sein. Die Batterie soll keinesfalls öfter als einmal pro Tag geladen oder gewechselt werden müssen.
4. Die eingesetzten Protokolle sollen weit verbreitet sein, damit möglichst viele Endgeräte von dem Wearable profitieren können und bestenfalls einzelne Komponenten des Wearables einfach durch Neuere ersetzt werden können.

Wegen der hohen Mobilität und Verbreitung wurde als Empfänger der Daten ein Smartphone ausgewählt, wobei durch Implementierung der Schnittstelle später auch andere Geräte genutzt werden können. Das Smartphone soll die Schnittstelle exemplarisch implementieren und die Daten sowie Statistiken zur Verbindung zur Auswertung anzeigen. Das Wearable soll bei Verbindung Sensordaten mit variabler Rate bereitstellen um Datenübertragung und Energieverbrauch bei verschiedenen Konfigurationen auswerten zu können.

1.3 Überblick der Gliederung

Zunächst werden verschiedene Protokolle, Techniken oder Komponententypen verglichen, die für das Wearable wichtig sind. Dann werden bestehende Lösungen vorgestellt und geprüft, welche Vor- und Nachteile sie bieten. Im Anschluss werden die eingesetzten Komponenten bezeichnet und beschrieben, wie sie miteinander interagieren sollen. In der Implementierung wird die Beschaltung beschrieben und es werden die Probleme benannt, die bei der Implementierung aufgetreten sind. Zum Schluss wird das System anhand der genannten Ziele evaluiert und ein Fazit gezogen.



2 Hintergrund

2.1 Sensor

Um Gelenkstellungen anzugeben wird eine Position und Rotation gebraucht. Diese Daten werden von einem Beschleunigungssensor und einem Gyrosensor erfasst und bestehen aus je 3 Achsen. Die Sensoren sind platz- und energiesparend zusammen in einer Inertialen Messeinheit (IMU) erhältlich. Im Folgenden wird ein Einblick in die Funktionsweise der beiden Sensorentypen gegeben.

2.1.1 Beschleunigungssensor

Beschleunigungssensoren können die Beschleunigung messen, die auf den Sensor einwirkt. Im Bereich der kleinen Bauteile werden zwei Techniken dafür eingesetzt.

Beim kapazitiven Sensor, der in Abbildung 2.1 illustriert ist, wird eine Masse senkrecht-federnd parallel zu einer Platte befestigt, mit der sie einen Kondensator bildet. Bei einer Beschleunigung bewegt sich die Masse zur Platte hin oder davon weg und die Kapazität des Kondensators ändert sich. Durch diese Änderung lässt sich dann der Wert für die Beschleunigung berechnen. [Pat19d]

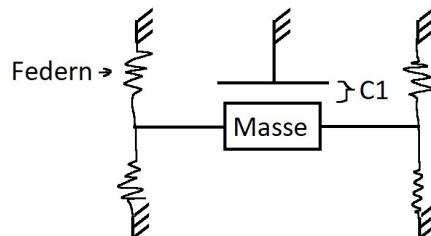


Abbildung 2.1: Funktionsweise eines kapazitiven Beschleunigungssensors. Basierend auf [sen18]

Beim piezoresistiven Sensor, der in Abbildung 2.2 dargestellt ist, wird ein Stoff mit piezoresistivem Effekt genutzt, der bei Dehnung seinen Widerstand ändert. Eine Masse wird an diesen piezoresistiven Stoff befestigt und durch die Beschleunigung wird der Stoff gedehnt, wodurch durch die Änderung des Widerstands die Beschleunigung berechnet werden kann. [Pat19b]

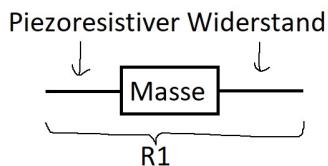


Abbildung 2.2: Funktionsweise eines piezoresistiven Beschleunigungssensors. Basierend auf [sen18]

2.1.2 Gyrosensor

Mit einem Gyrosensor, zu sehen in Abbildung 2.3, wird die Drehung gemessen. Eine Masse wird dabei senkrecht-federnd parallel zu einer Platte befestigt, sodass wieder ein Kondensator entsteht und parallel zu der Platte von außen zum Schwingen gebracht. Dieses Konstrukt gibt es ein zweites Mal in die andere Richtung schwingend. Bei einer Drehung bewegen sich die Massen durch den Coriolis Effekt.

Durch die Subtraktion der beiden Kapazitäten kann man dann die Drehgeschwindigkeit berechnen. Da die Schwingung gegensätzlich ist, bewegen sich die Massen durch den Coriolis Effekt bei einer Drehung in entgegengesetzte Richtungen und die Differenz der Kapazitäten ändert sich. Bei einer geraden Beschleunigung bewegen sich beide Massen in die selbe Richtung und die Differenz der Kapazitäten ändert sich nicht. [VPdN⁺¹⁰]

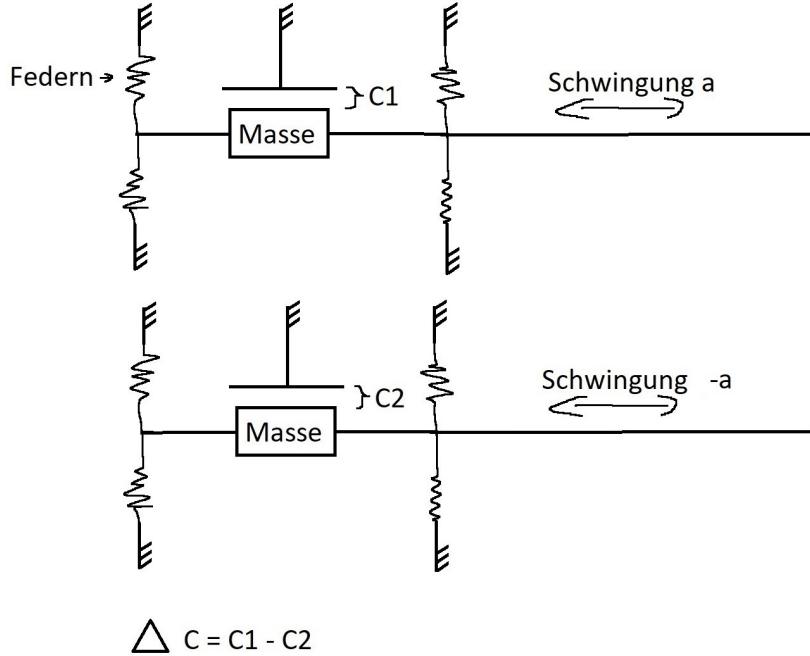


Abbildung 2.3: Funktionsweise eines Gyrosensors

Da konstant eine Schwingung auf der Masse gehalten werden muss, damit der Coriolis Effekt auftritt, benötigt der Gyrosensor meist mehr Strom als der Beschleunigungssensor, was später von den Komponenten bestätigt wird.

2.2 Chipkommunikation

Eine IMU enthält in der Regel keine programmierbare Chiparchitektur sondern wird mit einer Microprocessor Unit (MPU) betrieben. Die IMU beinhaltet unter anderem eine Recheneinheit und einen Programmspeicher. Zur Kommunikation zwischen IMU und MPU kann man wiederum zwischen einigen Systemen wählen. Die verwendeten IMUs unterstützen die weit verbreiteten Master-Slave-Systeme Inter-Integrated Circuit (I2C) und Serial Peripheral Interface (SPI), die auch Beide auf der gewählten MPU laufen. Die Chipkommunikation läuft in der Regel mit einer langsameren Frequenz als die Recheneinheit. Einerseits kann die Übertragungsfrequenz von der Software emuliert werden, indem Takte der Recheneinheit übersprungen werden, wodurch die Recheneinheit nicht in den Schlafmodus gehen kann. Andererseits kann die Recheneinheit einen weiteren Chip nutzen, der mit einem Buffer die Übertragung effizient abarbeiten kann.

2.2.1 I2C

Beim I2C-Protokoll sieht die Verdrahtung Serial Clock (SCL) und Serial Data (SDA) mit Pull-Up Widerständen vor. Der Master startet eine Datenübertragung mit START indem er SDA runter zieht während SCL oben ist. Dann gibt der Master den Takt auf SCL vor. Es folgt ein Paket mit 7 Bit für Slaveadresse und

1 Bit für Read oder Write. Der Slave bestätigt den Empfang mit ACK indem es SDA für ein Bit runterzieht. Nun können 8-Bit-Datenpakete übertragen werden, die jeweils mit ACKs bestätigt werden. Zum Schluss sendet der Master STOP indem er SDA hoch zieht während SCL oben ist. [NXP14]

2.2.2 SPI

Beim SPI-Bussystem sind Chip Select (CS), Serial Clock (SCK), Master-In-Slave-Out (MISO) und Master-Out-Slave-In (MOSI), meist mit Pull-Up Widerständen, verkabelt. Der Master startet die Übertragung, indem es CS runter zieht und dann auf SCK den Takt vorgibt. Auf MISO kann der Slave dann Daten zum Master schicken und auf MOSI der Master Daten zum Slave. Um die Übertragung zu beenden wird CS vom Master wieder hochgezogen. [mic]

2.3 Endgerätekommunikation

Damit die Daten von der MCU zum Endgerät kommen, wird ein weiteres Protokoll benötigt. Es wurde Bluetooth Low Energy (BLE) gewählt und ermöglicht, dass zukünftig auf Bluetooth Mesh (BT Mesh) gewechselt werden kann. Die Entscheidung wird später im Abschnitt 4.3 begründet.

Bluetooth Low Energy

BLE ist eine Abwandlung von Bluetooth, die für geringen Energieverbrauch optimiert ist. Geräte können dabei die Rolle vom Central oder Peripheral einnehmen, die einem Master-Slave Protokoll gleichkommen. Um eine Datenverbindung zu starten, kündigt sich die Peripheral auf den drei Advertising Frequenzen an während die Central diese Frequenzen abhört. Die Parameter dafür, zum Beispiel die bis zu 31 Byte große Payload, werden im Generic Access Profile (GAP) eingestellt. Die Central kann auf die Ankündigung der Peripheral antworten und sich mit ihm verbinden. [Ada19]

Nach dem Verbindungsauflauf gibt die Central eine Geschwindigkeit vor. Die Geschwindigkeit besteht aus einem Connection Interval, einer Slave Latency und einem Supervision Timeout. Das Connection Interval beschreibt in welchen Zeitabständen die Datenbursts passieren. Die Slave Latency bestimmt, wie oft das Peripheral die Datenbursts pausieren kann um Energie zu sparen, falls keine Daten gesendet werden müssen. Der Supervision Timeout besagt, nach welcher Zeit eine Verbindung als abgebrochen gilt, wenn eine Seite nicht mehr antwortet. Es gelten die Gleichungen 2.1 bis 2.4.

$$7.5ms \leqslant \text{ConnectionInterval} \leqslant 4s \quad (2.1)$$

$$0 \leqslant \text{SlaveLatency} \leqslant 499 \quad (2.2)$$

$$100ms \leqslant \text{SupervisionTimeout} \leqslant 32s \quad (2.3)$$

$$\text{SupervisionTimeout} > \text{ConnectionInterval} * (1 + \text{SlaveLatency}) \quad (2.4)$$

Später kann einerseits die Peripheral neue Geschwindigkeiten vorschlagen, die von der Central angenommen oder abgelehnt werden oder andererseits die Central neue Geschwindigkeiten von sich aus bestimmen. [Tex16b]

Bei einer Verbindung stellt die Peripheral Services zur Verfügung, die aus Characteristics bestehen. Jeder Service und jede Characteristic ist im Generic Attribute Profile (GATT) definiert und hat eine eindeutige UUID. Die UUID ist 16 Bit groß, wenn sie aus den vordefinierten Profilen besteht oder 128 Bit für eigene Definitionen. Eine Characteristic repräsentiert einen Wert und kann im Lese- und Schreibzugriff eingeschränkt werden. Die Central kann das notify- oder indicate-Bit setzen, wodurch die Peripheral Änderungen des Wertes mitteilt, wobei beim Indicate der Empfang der Updates von der Central bestätigt werden muss. [Nor16]

BLE arbeitet nach einem Schichtenmodell ähnlich dem OSI-Schichtenmodell. Während GATT und GAP auf der oberen Schicht liegen, liegt das Logical Link Control and Adaptation Layer Protocol (L2CAP) in der Mitte. Die Pakete der oberen Schicht werden im L2CAP in Päckchen aufgeteilt. Ein Päckchen besteht aus Header und Data und hat die Größe der Maximum Transmission Unit (MTU-Größe). Falls das Paket nicht komplett in Data passt, werden mehrere Päckchen erstellt. Der Header bei einem Notify-Paket ist 3 Byte groß. [Tex16c]

Bluetooth Mesh

BT Mesh erweitert BLE insoweit, als dass die Datenpakete über ein Mesh-Netzwerk geflutet werden. Der Vorteil davon ist, dass die Reichweite erhöht wird, da Pakete über Zwischenknoten gesendet werden können und, dass die Stabilität steigt, da einzelne Routen ausfallen können. Nachteilig ist, dass der Energieverbrauch der Geräte höher ist, da sie seltener in den Schlafmodus können, weil sie Daten von anderen Knoten übertragen müssen. Über Proxy-Knoten können BLE-Geräte eingebunden werden, die BT Mesh selber nicht unterstützen. [Pat19a]

2.4 Lithiumknopfzelle

Zur Energieversorgung wird eine Knopfzelle aus Lithium genutzt. Die Knopfzelle gibt es in verschiedenen Größen. Die gängigsten Größen CR2032, CR2430 und CR2450 eignen sich dafür gut. Die mittleren beiden Ziffern geben jeweils den Durchmesser und die hinteren beiden Ziffern die Dicke in Millimeter an. Da sich dementsprechend die Kapazität verändert, sind in der Tabelle 2.1 die Kapazitäten von stellvertretenden Modellen gelistet. Alle Modelle haben eine Nennspannung von 3 V und gelten unter 2 V als entladen. Durch ihre Selbstentladung von < 1 % pro Jahr, sind sie lange haltbar. [Var14c]

Tabelle 2.1: Typische Kapazität von Knopfzellen verschiedener Größen

Größe	Typische Kapazität
CR2032	230 mAh [Var14a]
CR2430	300 mAh [Var14b]
CR2450	620 mAh [Var14c]

3 Aktueller Stand

3.1 Bestehende Lösungen

Einige bestehende Hardwarelösungen eignen sich bereits für die Anwendungsfälle.

3.1.1 Texas Instruments CC2650STK

Darunter fällt beispielsweise das TI CC2650STK. Es besteht aus der CC2650 MCU sowie 10 Sensoren, darunter eine MPU-9250 IMU, die über I2C angeschlossen ist. Eine Auflistung der überflüssigen Komponenten, die nicht komplett abgeschaltet werden können, findet sich in Tabelle 3.1. Eine CR2032 Knopfzelle mit 230 mAh verliert dadurch $\frac{365d * 24h}{(230mAh/0.00351mA)} \approx 13.4\%$ der Kapazität pro Jahr. Die IMU verbraucht 3.4 mA, wenn Gyrosensor bei 1 kHz und Beschleunigungssensor bei 4 kHz laufen [Inv16]. Die MCU unterstützt zwar BLE aber kein BT Mesh. Das System ist nicht mehr in Europa verfügbar. [Tex]

Tabelle 3.1: Stromverbrauch der ausgeschalteten zusätzlichen Komponenten am TI CC2650STK

Sensortyp	Stromverbrauch ausgeschaltet
Lastschalter	0.9 μ A [Tex15]
Feuchtigkeitssensor	0.11 μ A [Tex16a]
Drucksensor	0.1 μ A [Bos18b]
Infrarottemperatursensor	2 μ A [dig]
Lichtsensor	0.4 μ A [Tex17]]
Gesamt	3.51 μ A

3.1.2 aconno ACNSENSA

Das aconno ACNSENSA besteht aus einer nRF52832 MCU und 5 Sensoren, darunter eine LSM9DS1 IMU über I2C. Eine Auflistung der überflüssigen Komponenten, die nicht komplett abgeschaltet werden können, findet sich in Tabelle 3.2. Eine CR2450 Knopfzelle mit 620 mAh verliert dadurch nur $\frac{365d * 24h}{(620mAh/0.0014mA)} \approx 2\%$ der Kapazität pro Jahr. Der Gyrosensor der IMU verbraucht 4.0 mA bei 952 Hz. Weitere 600 μ A sind zusammen angegeben für den Beschleunigungssensor bei 952 Hz und Magnetometer bei 20 Hz, der sich aber ausschalten lässt [STM15]. Die MCU unterstützt BT Mesh. [aco17b] Das System ist nicht gut verfügbar. So sind bei bekannten Händler nur 5 Exemplare zu bekommen [mou]. Die Produkt Webseite sowie die Datenblätter waren während über den Zeitraum dieser Arbeit teilweise nicht zugreifbar und es existieren mehrere Versionen unter dem selben Namen (vgl. Datenblatt und Produktbild bei mouser.de).

Tabelle 3.2: Stromverbrauch der ausgeschalteten zusätzlichen Komponenten am aconno ACNSENSA

Sensortyp	Stromverbrauch ausgeschaltet
Drucksensor	1 μ A [aco17b]
Feuchtigkeitssensor	0.06 μ A [aco17b]
Lastschalter	0.34 μ A [Vis15]
Gesamt	1.4 μ A

3.1.3 Arduino Primo Core

Der Arduino Primo Core beinhaltet ein nRF52832 MCU und 3 Sensoren, inklusive einer LSM6DS3 IMU über I2C. Eine Auflistung der überflüssigen Komponenten, die nicht komplett abgeschaltet werden können, findet sich in Tabelle 3.3. Eine CR2032 Knopfzelle mit 230 mAh verliert dadurch $365d * 24h \approx 5.7\%$ der Kapazität pro Jahr. Die IMU verbraucht bei 1.6 kHz nur 1.25 mA [STM17b]. Das System kann nicht mehr gekauft werden und ist auf der Produktseite als 'retired' gelistet.[ARD] [Ard16]

Tabelle 3.3: Stromverbrauch der ausgeschalteten zusätzlichen Komponenten am Arduino Primo Core

Sensortyp	Stromverbrauch ausgeschaltet
Magnetometer	1 μ A [STM17a]
Feuchtigkeitssensor	0.5 μ A [STM16]
Gesamt	1.5 μ A

3.2 Zusammenfassung und verwandte Arbeiten

Mit den existierenden Lösungen wäre es grundsätzlich möglich das Wearable umzusetzen. Allerdings gibt es neben der problematischen Verfügbarkeit eine Limitierung auf die gegebene Hardware. So verschwenden alle Lösungen Energie und Platz durch Sensoren, die für den benötigten Anwendungsfall nicht nötig sind, auch wenn das im Falle des aconno ACNSENSA vernachlässigbar ist. Hier ist auch zu beachten, dass zusätzlich zu dem Standbyverbrauch der Komponenten auch Leckströme durch Kondensatoren und Verluste durch Pull-Up Widerstände hinzuzufügen sind.

Alle genannten Lösungen nutzen für die Kommunikation zwischen MCU und IMU das I2C Protokoll, während in einer Arbeit [MT12] ermittelt wurde, dass I2C einen doppelt so hohen Energieverbrauch wie SPI hat. Auch zwischen den IMUs gibt es gravierende Unterschiede im Energieverbrauch. So verbraucht die IMU des Arduino Prime Core weniger als ein Drittel der Energie der anderen Beiden und enthält als Einziges keinen Magnetometer.

Ein Magnetometer entspricht einem elektrischen Kompass und bestimmt die Rotation in Ausrichtung zu den Erdpolen. In der Arbeit [ABCO12] wurde ein Sensorfusionsalgorithmus entwickelt, der die Genauigkeit des Gyrosensors mit den Daten eines Magnetometers verbessert. Der Algorithmus kann in zukünftigen Versionen integriert werden, falls die Genauigkeit der verwendeten IMU nicht ausreichen sollte.

In dem Projekt [Cho] wird ein System zur Bewegungsaufnahme mit einer IMU entwickelt. Hierbei werden die Sensoren per Kabel miteinander verbunden. Da die Software allerdings open-source ist, könnte darauf aufbauend eine eigene Bewegungserkennungssoftware erstellt werden.

4 Design und Planung

Um das Vorhaben umzusetzen, müssen zuerst die Anforderungen bestimmt, einige Entscheidungen getroffen und erste Pläne entworfen werden.

4.1 Anforderungen

Als Komponenten werden hauptsächlich eine IMU und eine kompatible MCU, die Daten über BLE übertragen kann sowie für zukünftige Tests auch BT Mesh unterstützt, benötigt. Die Komponenten sollen ähnliche Anforderungen an die Spannungsversorgung besitzen wie eine Lithiumknopfzelle sie bereitstellen kann, damit kein zusätzlicher Spannungsregler benötigt wird. Ferner sollen die in der Abschnitt 1.2 festgelegten Ziele angestrebt werden.

4.2 Energieversorgung

Zu den typischen mobil verfügbaren Spannungsquellen gehören wiederaufladbare Akkus und Einwegbatterien, die elektrische Energie chemisch speichern können. Die Gängigsten basieren auf Nickel oder Lithium, wobei die Lithiumbasierenden wegen der höheren Energiemenge bei gleichem Gewicht und Größe unter dem Gesichtspunkt der Mobilität präferiert werden. Lithiumakkus benötigen eine Ladeelektronik sowie einen Entladeschutz. Diese Komponenten nehmen Platz auf der Platine ein, verbrauchen Energie, wodurch die Effizienz sinkt, und verlieren ihre maximale Kapazität mit zunehmendem Alter. Durch ihre maximale Spannung von 4.2 Volt könnten sie nicht ohne Spannungswandler mit den meisten Komponenten betrieben werden, der wiederum Effizienz einbüßen würde. [Pat19c]

Deshalb wurden Lithiumknopfzellenbatterien als Energieversorgung gewählt. Durch ihre flache Bauform können sie platzsparend an Platinen befestigt werden. Während auf dem Prototyping-Board Platz für eine CR2032-Knopfzelle bereit steht, soll auf dem Evaluationsboard aus praktischen Gründen eine CR2450-Knopfzelle genutzt werden.

4.3 Datenübertragung zwischen Wearable und Endgerät

Um das am Besten geeignete Protokoll zu finden, um die Sensordaten der IMU auf die Endgeräte zu übertragen, wurden verschiedene Protokolle verglichen und nach dem Ausschlussverfahren gewählt. Eine drahtgebundene Datenübertragung ist schnell und sehr energiesparend und es sind viele verbreitete Anschlüsse spezifiziert. Die Mobilität ist dabei aber sehr stark eingeschränkt, da Kabel insbesondere beim Sport sehr stören.

Während einige drahtlose Kommunikationsprotokolle wie NFC wegen der geringen Reichweite nicht geeignet sind, sind Mobilfunkprotokolle wie LTE wegen zu hoher Reichweite ineffizient [ZJM17]. Die bekannten Protokolle Wi-Fi und Bluetooth arbeiten unter anderem auf der 2.4-GHz Antenne, weswegen Peer-to-Peer(P2P) Protokolle auf dieser Frequenz in Tabelle 4.1 verglichen wurden.

Tabelle 4.1: P2P-Protokolle auf der 2.4-GHz Frequenz

Protokoll	Typische Anwendung	Kommentar
Wi-Fi Hotspot	Teilen der Internetverbindung in einem lokalem Netzwerk	Am Smartphone kann ein Hotspot erstellt werden, die Wearables könnten ihn suchen und sich verbinden, woraufhin z.B. über eine REST-API kommuniziert werden kann. Es kann auch die 5-GHz Frequenz genutzt werden.
Wi-Fi Direct	P2P-Kommunikation	Wie Wi-Fi Hotspot aber die Wearables bekommen keinen Internetzugriff vom Smartphone.
Bluetooth (BT)	Datenübertragung, Audiostreaming	Nicht für niedrigen Energieverbrauch ausgelegt.
BT Low Energy (BLE)	P2P-Kommunikation mit geringem Energieverbrauch, z.B. Smartwatches	Während die maximale Geräteanzahl nicht vom Protokoll spezifiziert ist, ist sie in Android auf 7 Geräte beschränkt ^a . Weiterhin ist die Anzahl zum Beispiel durch die Auslastung der Geräte und des Funkkanals begrenzt.
BT Mesh	Mesh-Kommunikation mit geringem Energieverbrauch	Nachrichten werden hierbei über das gesamte Netzwerk geflutet, wodurch die Reichweite des Netzwerkes erhöht wird und die Auslastung auf das Netzwerk verteilt wird aber der Energieverbrauch aller Knoten steigt. BLE Geräte können mit Mesh-Netzwerken kompatibel gemacht werden, indem sie sich zu einem Knoten des Netzwerks verbinden.
Thread, ZigBee	Smarthome	Nicht ohne weitere Hardware unter Android
Ant, Ant+	Sensordatenübertragung mit geringem Energieverbrauch	Nicht von allen Smartphones ohne zusätzliche Hardware unterstützt

^a Beschränkung durch GATT_MAX_PHY_CHANNEL [Goo14]

Da Bluetooth nicht für einen geringen Energieverbrauch ausgelegt ist, werden die Alternativen BLE und BT Mesh bevorzugt. Die Smarthome Protokolle funktionieren ohne zusätzlicher Hardware nicht am Smartphone. Ant und Ant+ sind zwar für die Sensordatenübertragung bei Smartphones ausgelegt, aber brauchen zusätzlich ein Dongle, wenn sie das Protokoll nicht unterstützen. Sowohl handelsübliche Laptops als auch das vorliegende Smartphone, ein Pocophone F1, unterstützt die Protokolle nicht. "BLE ist etwa 30% energieeffizienter als Wi-Fi"¹ [PPLA17], weswegen die Wi-Fi Protokolle heraus fallen. BT Mesh löst zwar das Problem der Geräteanzahlbeschränkung unter Android, allerdings auf Kosten des Energieverbrauchs. Die erweiterte Reichweite durch das Mesh-Netzwerk wird bei einem Wearable in der Regel nicht benötigt.

Die Datenübertragung zwischen Wearable und Endgerät soll mit BLE stattfinden, da dieses Protokoll von fast jedem Smartphone und Laptop unterstützt wird und den Anforderungen erfüllen kann. Zusätzlich soll BT Mesh als Option offen gelassen werden, da es gegebenenfalls auftretende Probleme mit BLE lösen kann. Der zusätzliche Energieverbrauch durch das Mesh-Netzwerk müsste zunächst evaluiert werden.

¹ Übersetzung durch den Verfasser

4.4 Wahl der Hauptkomponenten

Die Arbeit hat sich zunächst danach gerichtet Komponenten zu wählen, die es auf fertige Platinen gelötet gibt, da sich QFN-Komponenten händisch nicht löten lassen.

Da sich der Energieverbrauch von MCUs anhand der Datenblätter schlecht vergleichen lässt, wurde betrachtet, welche Komponenten sich oft zusätzlich auf den Platinen befinden aber viel Strom verbrauchen. In einem Blog [mad19] wurden verschiedene Komponenten von einem Arduino Pro Mini entfernt und deren Auswirkungen auf den Stromverbrauch ermittelt. Das Entfernen der LED, die dauerhaft leuchtet um den Betrieb anzuzeigen, hat eine Einsparung von mehr als 15 % bei aktiver Nutzung bewirkt und mehr als 90 % im Schlafmodus der Recheneinheit. Das Entfernen des Spannungsreglers hat mindestens weitere 9 % bei Nutzung und 75 % im Standby eingespart. Ein anderer Blog [ALE16] beschreibt, dass ein integrierter Programmierer zwar einen einfach zu verwendenden USB Anschluss bietet, aber gleichzeitig die Stromaufnahme verdreifachen kann. Während eine LED problemlos zu entfernen ist, wurde auf die anderen beiden Bauteile nach Möglichkeit verzichtet, da ein Entfernen auch die Funktion der Platine einschränken oder die Platine zerstören kann.

4.4.1 MCU

Die Bluetooth SIG, die das Bluetooth Protokoll spezifiziert, hat eine Liste mit geprüften Implementierungen von BT Mesh veröffentlicht [Blu]. Mit dieser Liste wurden die Herstellerseiten nach passenden verfügbaren MCUs durchsucht bis die Wahl auf die nRF52832 MCU gefallen ist. Die MCU war schon in den zwei fertigen Produkten von Arduino und aconno enthalten weswegen von einer hohen Bekanntheit auszugehen ist.

Der nRF52832 hat eine Spannungsanforderung von 1.7 bis 3.6 V, wodurch er sich direkt an einer Knopfzelle betreiben lässt. Das EasyDMA-Modul enthält einen Buffer um SPI- und I2C-Kommunikation abzuhandeln, während die MCU im Schlafmodus ist. Das schwächere Modell nRF52810 unterstützt BT Mesh nicht. Das stärkere Modell nRF52840 hat mehr Speicherplatz und unterstützt weitere Protokolle aber verbraucht dementsprechend mehr Energie. [Nor]

Durch den Pinabstand von 2.54 mm, wie er sich auch auf Arduinos vorfindet, ist die Platine von Spark-Fun sehr einfach zu verwenden. Sie enthält eine Power-LED, die sich über JP6 trennen lässt. Es existiert ein Spannungsregler, der sich überbrücken lassen müsste. Die Platine stellt eine Alternative dar, falls sich mit den kleineren Platinen wegen der Größe nicht mehr arbeiten lassen würde. Die Größe dieser Platine ist nicht direkt angegeben, aber wegen des Pinabstands mindestens 15 x 43 mm. [Spa16]

Der ACN52832 von aconno enthält weder Power-LED noch Spannungsregler sondern nur die nötigsten Komponenten. Er hat eine Größe von 20 x 24.8 mm. [aco17a]

Der BL651 von Laird ebenfalls nur die nötigsten Komponenten und hat eine Größe von 10 x 14 mm [Lai19]. Es existieren noch weitere Platinen vom nRF52832 sodass eine sehr gute Verfügbarkeit gegeben ist. Sollte eine Platine nicht mehr erhältlich sein, könnte sie durch eine Andere ersetzt werden, da die Platinen nur die Pins vom nRF52832 besser zugänglich machen und die nötigen Taktgeber, Kondensatoren und Antennen bereitstellen sollen. Für den Prototypen wurde zunächst das nRF52 DK von Nordic verwendet, da es auch den externen Programmierer für die Platinen enthält und die Pins ohne zu Löten mit Jumperkabel verbunden werden können.

4.4.2 IMU

Auf der Suche nach einer passenden IMU haben sich der STMicroelectronics LSM6DSL und Bosch BMI260 als besonders energiesparend hervorgetan. Der LSM6DSL läuft mit 1.71 - 3.6 V, verbraucht 0.4 mA im normalen Modus und 0.65 mA im Performance-Modus [STM17c]. Die zugehörige Platine STEVAL-MKI178V2 enthält keine überflüssigen Komponenten.

Der BMI260 läuft mit 1.7 - 3.6 V und verbraucht 0.685 mA. Der Sensor war zu Beginn der Arbeit angekündigt. Da er pinkompatibel zum Vorgänger BMI160 ist, wurde mit dem BMI160 entwickelt. Zum Ende der Arbeit sollte er durch den BMI260 getauscht werden, aber leider war er da nicht verfügbar. [Bos] Der BMI160 läuft auch mit 1.7 - 3.6 V aber verbraucht 0.925 mA [Bos18a]. Das BMI160 Shuttle Board enthält zusätzlich einen BMM150 Magnetometer. Der Magnetometer hat im Standby einen Verbrauch von $1 \mu\text{A}$, kann aber mechanisch entfernt werden [Bos19].

Beide IMUs haben einen FIFO-Buffer eingebaut. Die gemessenen Daten können in dem Buffer hinterlegt werden, sodass die MCU die Daten in Bursts lesen und länger im Schlafmodus bleiben kann, da ein Wechsel der Energiemodi wiederum Energie kostet.

Mit zwei Interrupt Pins können die IMUs die MCU über bestimmte Events informieren. Das DataReady-Event benachrichtigt, wenn der Sensor neue Daten gemessen hat. Das FIFO-Watermark- oder FIFO-Threshold-Event wird geworfen, wenn die FIFO über einen eingestellten Prozentsatz gefüllt ist. Das FIFO-Full-Event zeigt an, dass die FIFO komplett gefüllt ist. Für spezielle Anwendungen unterstützen beide IMUs weitere Events, wie das Significant-Motion-Event um ein Aufheben des Sensors durch einen Nutzer zu melden.

Da die Platinen der IMUs wegen ihrer Größe nicht optimal sind und zukünftige Anwendungen bessere Sensoren oder einen eingebauten Magnetometer benötigen können, soll der Code modular aufgebaut sein, sodass das Sensormodell einfach getauscht werden kann.

4.5 Zusammenwirken der Komponenten

Während SPI 4 Datenverbindungen braucht, braucht I2C 2 Datenverbindungen. Da die Pins in der Software der MCU zugewiesen werden, kann für SPI verkabelt und die selbe Verdrahtung auch für I2C genutzt werden. Abbildung 4.1 stellt die Verbindungen graphisch dar.

Wenn das Wearable nicht mit einem Endgerät verbunden ist, soll es in den Schlafmodus wechseln. Über einen Knopf soll es aufgeweckt werden und mit dem Advertise beginnen. Mit dem Knopf soll das Advertise auch wieder abgebrochen werden und das Wearable in den Schlafmodus versetzt werden können. Wenn das Wearable mit einem Endgerät verbunden ist, soll es sich trennen lassen, indem der Knopf über einige Sekunden gedrückt gehalten wurde, damit es sich nicht versehentlich trennt.

Um dem Nutzer ein Feedback über den Status zu geben, soll eine LED verwendet werden. Die LED soll beim Advertise in regelmäßigen Abständen blinken, sodass sich der Stromverbrauch nicht wie bei einer immer angeschalteten Power-LED auswirkt. Wenn eine Verbindung hergestellt wurde, soll eine zweite LED zu Beginn der Verbindung aufblinken. Wenn während einer Verbindung kurz auf den Knopf gedrückt wird, soll die zweite LED kurz aufleuchten.

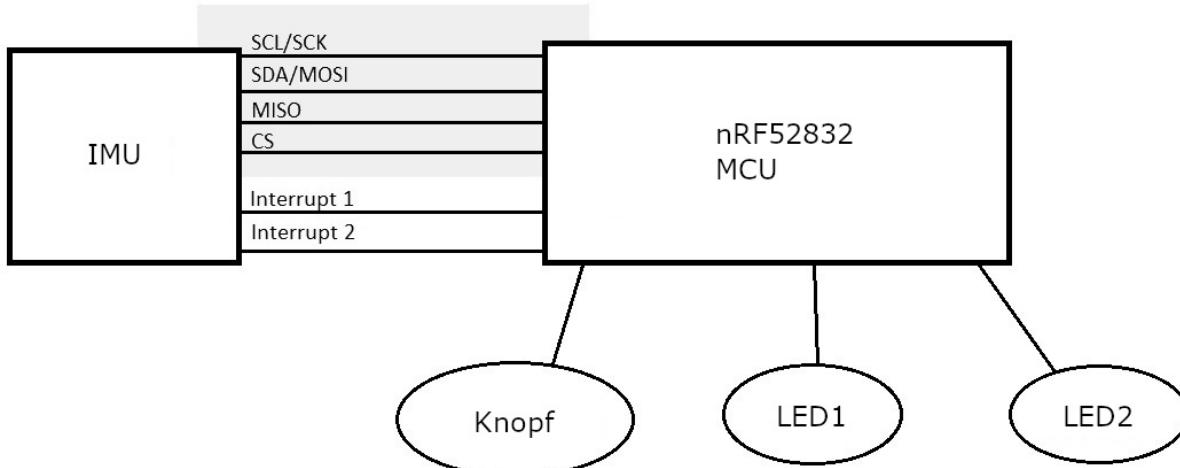


Abbildung 4.1: Veranschaulichung der Verbindungen der benötigten Komponenten

4.6 BLE Schnittstelle

In einer Characteristic sollen die 3 Achsen des Beschleunigungssensors, die 3 Achsen des Gyrosensors und der Zeitstempel hinterlegt werden. Damit soll sich der Bewegungsablauf des Sensors zeitlich korrekt rekonstruieren lassen, da BLE die Daten in Burst mit Zeitabständen vom Connection-Interval übergibt und die Daten in Burst von der FIFO abgerufen werden können. Um die MCU möglichst selten aus dem Schlafmodus zu holen, soll die FIFO der IMU in den Buffer der EasyDMA geleert werden, sobald sie komplett gefüllt ist. Zusätzlich soll umgefüllt werden, wenn die MCU durch andere Interrupts die z.B. durch den BLE-Stack aufgeweckt wird. Der Wert der Characteristic soll von der MCU nur gelesen werden können, aber es soll das Notify-Bit gesetzt werden können.

Mit einer zweiten Characteristic soll sich die Abfragerate der IMU einstellen lassen. Da die Sensorabfragerate damit von der Datenübertragungsrate unabhängig ist, lässt sich die Energieaufnahme getrennt evaluieren. Während eine niedrige Sensorabfragerate mit einer hohen Datenübertragungsrate sehr geringe Latenzen der Sensordaten ermöglicht, sind hohe Sensorabfrageraten mit niedrigen Datenübertragungsraten bei weniger zeitkritischen Anwendungen zur intensiven Auswertung interessant. Damit die Verbindung aufrecht erhalten werden aber die Datenerfassung pausiert werden kann, soll die IMU über die Characteristic auch kurzzeitig ausgeschaltet werden können. Die zweite Characteristic soll von der MCU gelesen und geschrieben werden können, sowie das Endgerät bei Änderungen benachrichtigt werden.

4.7 Applikation des Android Smartphones

Beim Start der App soll nach den Wearables gesucht und zu den Wearables verbunden werden. Bei einer Verbindung soll das Notify-Bit der Characteristics gesetzt werden, damit neue Daten beim Smartphone ankommen. Um einschätzen zu können, ob die empfangenen Daten korrekt erfasst werden und ankommen, sollen die Sensordaten und Statistiken zur Verbindung graphisch dargestellt werden. Da es im Android SDK keine einfache Möglichkeit dazu gibt, soll die Bibliothek Androidplot² genutzt werden. Der Wert der zweiten Characteristic soll gesetzt und die Verbindungsdatenrate geändert werden können.

² <http://androidplot.com/>, abgerufen am 10.07.2019



5 Implementierung

Die Implementierung setzt sich zusammen aus dem Programm auf dem Wearable, der Hardware des Wearables und der Android App. Im Folgendem werden die Teilaufgaben einzeln näher betrachtet.

5.1 Wearable Programm

5.1.1 Entwicklungsumgebung

Die nRF52832 MCU wird neben dem offiziellen SDK auch von der plattformübergreifenden Softwareumgebung mBed unterstützt. Eine plattformübergreifende Softwareumgebung zu nutzen kann es vereinfachen, den Code von der MCU auf ein anderes MCU-Modell zu portieren. Während die Einrichtung von Mbed sehr einfach ist, da der Code im Browser geschrieben wird, unterstützt Mbed bisher leider nicht BT Mesh, weswegen es sich nicht zur weiteren Evaluation eignet [Arm19].

Die Arbeit nutzt das nRF5 SDK v15.3.0. Damit das SDK vom Compiler bei Nutzung der IDE Segger Embedded Studio gefunden werden kann, muss es nach C:\nrf52\ sdk entpackt werden oder die Pfade in den .emProject-Dateien ersetzt werden. Hier würde es sich eignen, zukünftig eine Lösung mit systemweiten Pfadvariablen zu finden. Bevor eine nRF52832 MCU genutzt werden kann, muss zunächst der BLE-Stack installiert werden, indem die .hex-Datei aus dem Softdevice 132 v6.1.1 auf die MCU übertragen wird.

Statt die IDE Segger Embedded Studio zu nutzen, kann alternativ der GCC Compiler mit einer beliebigen IDE eingesetzt werden. Bei der komplizierten Einrichtung von GCC und den benötigten Tools ist dann zu beachten, dass bei der zum Zeitpunkt der Arbeit aktuellen Version 8-2018-q4-major der GNU Arm Embedded Toolchain¹ ein Out-of-Range-Fehler in einer Intel-Hex-File geworfen wird. Der Fehler kann behoben werden, indem die Datei bin\arm-none-eabi-objcopy.exe mit der aus Version 7-2018-q2-update ersetzt wird.

In der Arbeit wurde wegen der einsteigerfreundlicheren Einrichtung mit der IDE Segger Embedded Studio gearbeitet. Die IDE stürzt in der genutzten Version 4.12 leider regelmäßig ab. Wird die IDE dagegen aktualisiert, gibt das eingebaute Debug Terminal nur noch unsichtbare Zeichen aus (vgl. [Mau19]). Im Debug Terminal wird der Log, der vom RTT-Protokoll übertragen wird, angezeigt. Wenn der Log über RTT zu schnell ist, werden Zeilen verworfen. Stellt man den Log über den Configuration Wizard in sdk_config.h auf das ineffizientere UART-Protokoll um, lässt er sich über das externe Programm Putty² auslesen.

5.1.2 Programmaufbau

Das Programm main.c ist der Einstiegspunkt der MCU. Hier werden die vordefinierten und eigenen Module initialisiert und die Hauptschleife verwaltet. Unter den vordefinierten Modulen sind zum Beispiel Logging, Timer und Energiemanagement. Da das Template 700 Zeilen umfasst, wurden einige Initialisierungen in weitere Dateien ausgelagert. Der Code peer_advertise.c verwaltet die beiden Module, die zum Advertisen und zur optionalen Verbindungsverschlüsselung nötig sind. Hier kann unter anderem eingestellt werden, wie lange das Advertisen passiert, bevor das Wearable automatisch in den Schlafmodus wechselt und in welchen Abständen die Pakete beim Advertisen verschickt werden. Die Datei

¹ <https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm>, abgerufen am 13.07.2019

² <https://www.putty.org/>, abgerufen am 11.07.2019

`board_manager.c` steuert die LEDs und den Knopf. Die Definition `custom_board.h` beschreibt, wie viele LEDs und Knöpfe angeschlossen sind und an welchen Pins sie liegen.

Im Package `ServiceCharacteristicSensor` befindet sich der Code für die BLE-Service und den Sensor.

5.1.3 BLE-Service

Um den BLE Service und die Characteristics zu erstellen wurde nach dem Tutorial³ eines Mitarbeiters von Nordic Semiconductor gearbeitet. Beim Initialisieren des Services `my_service.c` werden die beiden Characteristics `chara_conf.c` und `chara_data.c` erstellt und der Sensor vorbereitet. Wenn sich ein Endgerät verbindet, startet der Sensor das Sammeln der Daten und beim Trennen wird er angehalten und zurückgesetzt.

Erfasst der Sensor einen neuen Datensatz, wird der Datensatz im Wert der Characteristic `chara_data.c` veröffentlicht. Da die Werte erst zum nächsten Connection Interval verschickt werden, müssen sie vom BLE-Stack zwischengespeichert werden. Ist der Buffer allerdings voll, wird beim Einfügen ein `NRF_ERROR_RESOURCES`-Fehler von der Funktion `sd_ble_gatts_hvx` geworfen. Die Minimalgröße des TX-Buffers kann vor dem Initialisieren des BLE-Stacks mit der Konstanten `TX_QUEUE_SIZE` manuell gesetzt werden und ist standardmäßig 0. Es ist möglich die tatsächliche Größe abzuschätzen, da bei einer Teilentleerung des Buffers ein `TX_COMPLETE`-Event geworfen wird. Wird beim Einfügen in den TX-Buffer ein Counter hoch- und bei einem `TX_COMPLETE`-Event runtergezählt, kann die Kapazität bei einem `NRF_ERROR_RESOURCES`-Fehler erhalten werden. Nach diesem Verfahren wurde bei dem Programm dieser Arbeit ermittelt, dass die tatsächliche Kapazität etwa 3 mehr als die Minimalgröße beträgt.

Wenn betrachtet wird, dass die verwendeten IMUs die Daten mit bis zu 1600 Hz erfassen können, ist die benötigte Buffergröße $1600\text{Hz} * 4\text{s} = 6400$ wegen dem maximalen Connection Interval von 4 Sekunden. Den Buffer auf diese Größe zu setzen muss aber nicht sinnvoll sein, da nicht sicher gestellt ist, dass in einem Connection Interval der komplette Buffer geleert wird. Das ist von der Übertragungsgeschwindigkeit abhängig, die sich auch aus der Empfangsqualität und der Paketverlustrate ergibt. Ist der Buffer größer als die Übertragungsgeschwindigkeit erlaubt, wird der Buffer volllaufen, Sensordaten werden letztendlich verworfen und die eingereichten Sensordaten kommen erst in einem späteren Connection Interval an, sodass eine Verzögerung entsteht. Wird der Buffer kleiner gewählt, passen nicht alle Sensordaten hinein und müssen verworfen werden.

Im Optimalfall sollte die Übertragungsgeschwindigkeit maximiert werden und der Buffer dazu passend mit etwas Spielraum nach unten gewählt werden. Die maximale MTU-Größe kann mit dem Configuration Wizard in `sdk_config.h` mit der Konstanten `NRF_SDH_BLE_GATT_MAX_MTU_SIZE` geändert werden. Sowohl eine größere MTU-Größe als auch ein größerer TX-Buffer benötigt zusätzlichen Platz in der BLE-Stack-Section vom RAM. Der reservierte Platz im RAM kann eingestellt werden mit der Konstanten `RAM_START` unter den `linker_section_placement_macros` in der SES-Projekt-Datei, zu finden mit der Endung `.emProject`. Wird zu wenig Platz reserviert, startet das Programm nicht.

Der Wert, der hinter der Characteristic `chara_data.c` steht, besteht aus den Komponenten in Abbildung 5.1. Aus dem Inhalt ergibt sich eine Größe von 16 Byte. Das MSb des Zeitstempels wird nicht genutzt, da auf Endgeräten genutzte Hochsprachen oft keine vorzeichenlosen Werte nutzen können. Die Einheit ist durch die LSM6DSL IMU vorgegeben. Die IMU speichert den Zeitstempel im 6.4 ms Format. Es würde Sinn ergeben, dieses Zeitformat auf das schnellste Datenerfassungsintervall von $\frac{1\text{s}}{1600\text{Hz}} = 0.625\text{ms}$ zu normalisieren, sodass beide Sensoren dieselbe Zeiteinheit liefern können. Allerdings ist die Zeiteinheit kein Vielfaches, sodass Rundungsfehler entstehen. Der LSM6DSL arbeitet zudem mit 1666 Hz statt den 1600 Hz beim BMI160. Hierfür sollte in Zukunft eine bessere Lösung gefunden werden.

³ https://github.com/bjornspockeli/custom_ble_service_example, aufgerufen am 11.07.2019

```

1 typedef struct {
2     int16_t accel_x;
3     int16_t accel_y;
4     int16_t accel_z;
5     int16_t gyro_x;
6     int16_t gyro_y;
7     int16_t gyro_z;
8     uint32_t time;
9 } imu_data_t;

```

Abbildung 5.1: Inhalt von Characteristic chara_data.c

Schickt das Endgerät dem Wearable mit der Characteristic `chara_conf.c` eine neue Konfiguration, wird die Konfiguration versucht angewandt zu werden. Ändert sich die Konfiguration, wird sie dem Endgerät mit einem Notify mitgeteilt.

5.1.4 IMU Integration

Damit die IMU aber auch die verwendeten Protokolle leicht austauschbar sind, wurde der Code für die IMU aufgeteilt. Die Definitionen in `imu.h` soll dabei jede IMU implementieren und ist die Schnittstelle zum BLE-Service. Der Header `*_protocol.h` deklariert das Protokoll, mit dem die IMU mit MCU kommuniziert, das heißt SPI oder I2C. Mit der Konstanten `PROTOCOL_SPEED` lässt sich die Geschwindigkeit des Protokolls ändern. Der Header `*_get_data.h` deklariert die Art, wie Daten gesammelt werden. Die Implementierungen können unterschiedliche Interrupts nutzen wie FIFO-Full oder Data-Ready. Um die Implementierung auszutauschen muss eine andere .c-Datei gelinkt werden, indem die entsprechende Zeile im .emProject-Projekt geändert wird.

Die IMU soll mithilfe der Callback-Funktion `bool (*send_data)(imu_data_t)` die gemessenen Werte an die MCU zurück liefern.

Beim Testen mit dem Prototypen ist das Problem, das in Abbildung 5.2 veranschaulicht wird, aufgekommen. Wenn die Sensordatenrate zu schnell für den TX-Buffer oder dem Connection Interval ist, empfängt das Endgerät die Sensordaten mit wechselnder Verzögerung. Wurde der TX-Buffer gerade durch ein Connection Interval geleert, werden alle nächsten Sensordaten in den Buffer eingetragen, bis der Buffer voll ist. Danach werden alle Sensordaten bis zum nächsten Connection Interval verworfen.

Prinzipiell kann das Problem gelöst werden, indem der Connection Interval verkleinert, der TX-Buffer vergrößert oder die Datenmenge verkleinert wird. Da die Größe vom TX-Buffer nach der Initialisierung nicht geändert werden kann und von der veränderlichen Durchsatzrate beschränkt ist, reicht es nicht unbedingt alleine aus, diesen Parameter zu variieren. Der Connection Interval wird vom Endgerät bestimmt. Das Wearable kann eine neue Geschwindigkeit vorschlagen. Können sich Wearable und Endgerät nicht einigen, wird die Verbindung getrennt. Das würde das Nutzungserlebnis stark negativ beeinflussen. Um die Datenmenge zu reduzieren kann der Sensor langsamer gestellt oder absichtlich Daten verworfen werden. Während das Zweite eine feinere Einstellung ermöglicht, ist das Erste energiesparender.

Deswegen wird ein Algorithmus eingesetzt, der die Geschwindigkeit vom Sensor herabsetzt, wenn ein neuer Connection Interval vorliegt oder der Nutzer eine neue Geschwindigkeit setzen möchte. Das Ergebnis soll dem in Abbildung 5.3 veranschaulichte Verhalten nachkommen. Der Rückgabewert der Callback-Funktion `bool (*send_data)(imu_data_t)` gibt an, ob die Daten in den TX-Buffer eingefügt werden konnten oder der Buffer schon voll ist. Der Parameter `uint16_t buffer_size` der Funktion `imu_init` entspricht der eingestellten Größe des TX-Buffers. Die Funktion `void imu_on_new_interval(uint16_t buffer_clear_interval)` wird aufgerufen, wenn die Geschwindigkeit der BLE-Verbindung sich geändert hat. Der Parameter `buffer_clear_interval` ist dabei der Connection Interval. Die Ge-

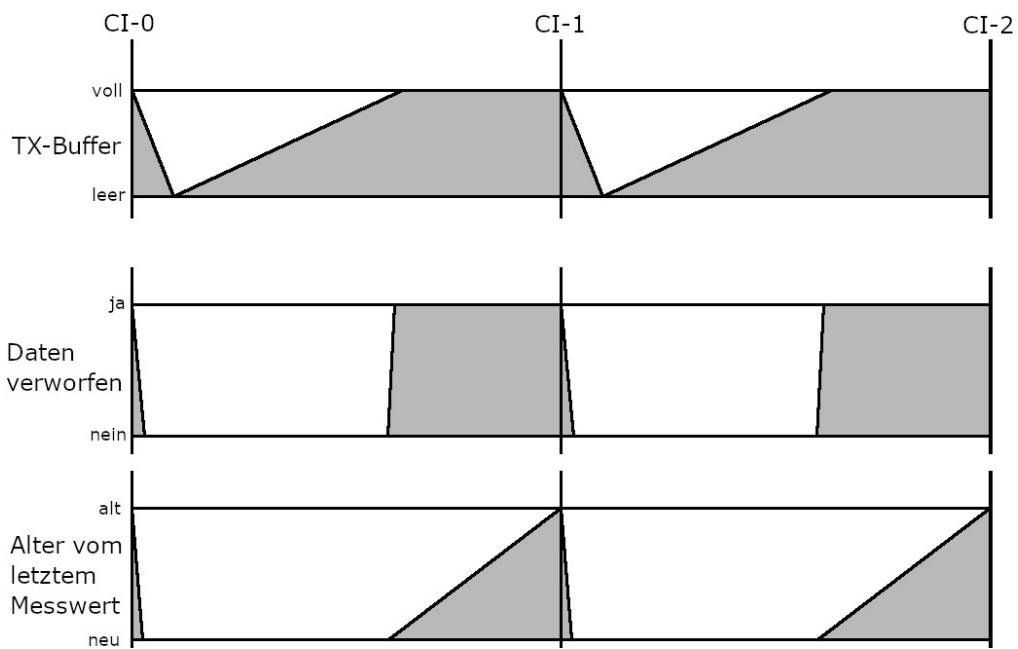


Abbildung 5.2: Darstellung des Datenstaus. CI = Connection Interval

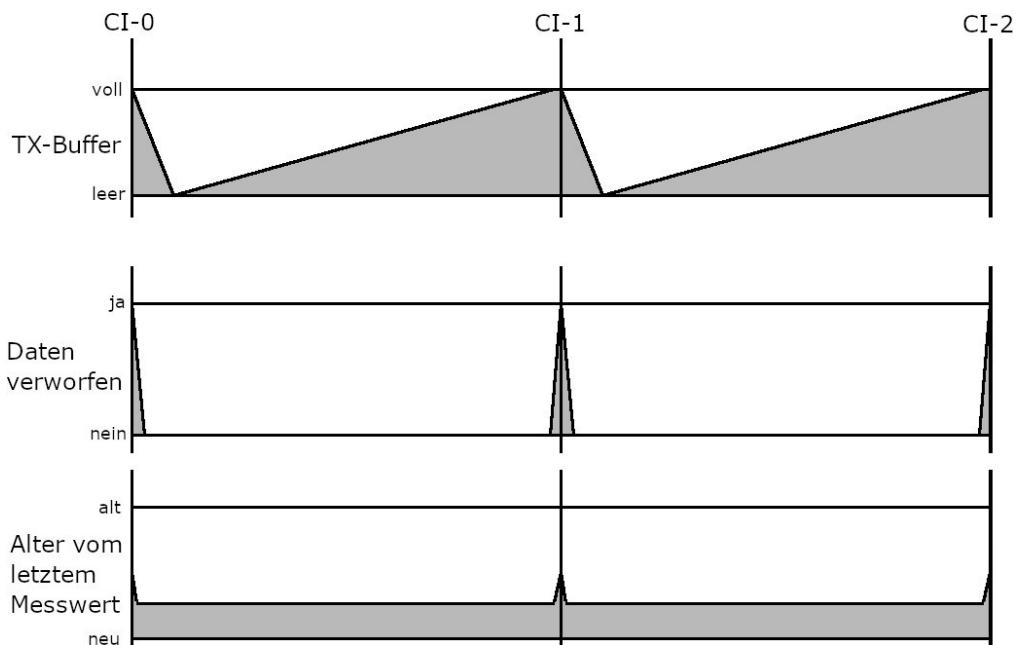


Abbildung 5.3: Lösung des Datenstaus

schwindigkeit wird gespeichert und in den weiteren Aufrufen von der Funktion `void imu_speed_set(imu_speed_t speed)` verwendet.

Der Algorithmus wird am Beispiel der LSM6DSL IMU mit FIFO-Threshold Interrupt beschrieben (`lsm_get_data_fifo.c`). Zunächst werden in Abbildung 5.4 die Konstanten für den Sensor berechnet. Die Geschwindigkeit wird in Abbildung 5.5 berechnet. Dabei wird mit jeder Iteration berechnet,

```
1 // ein Paket besteht aus 3 DATAs: Beschleunigungssensor, Gyrosensor, Zeitstempel
2 #define DATAS_PER_PACKET 3u
3 // jede DATA hat 3 Achsen mit je 2 Byte
4 #define BYTES_PER_DATA 6u
5 // die FIFO ist 4096 Byte groß. Bei 18 Byte pro Packet können 4086 Byte genutzt
   werden
6 #define FIFO_MAX_FILL_BYTES (4096u - ((4096u % (BYTES_PER_DATA *
   DATAS_PER_PACKET))))
7 // Das sind 227 Pakete in der FIFO
8 #define FIFO_MAX_FILL_PACKETS (FIFO_MAX_FILL_BYTES / BYTES_PER_DATA /
   DATAS_PER_PACKET)
9 // Per SPI können 254 Byte pro Durchgang ausgelesen werden, also 14 Pakete
10 #define MAX_PACKETS_PER_TRANSACTION ((0xFFu - 1u) / (BYTES_PER_DATA *
   DATAS_PER_PACKET))
11 // so viele Pakete werden bei jedem Durchlauf später ausgelesen. Ein Output von
   lsm_get_data_speed_set()
12 static uint16_t packets_per_transaction = MAX_PACKETS_PER_TRANSACTION;
13 // die Größe des TX-Buffers wird beim Initialisieren gesetzt
14 static uint16_t buffer_size;
```

Abbildung 5.4: Vorbereitung des Algorithmus

ob der Sensor in einem Connection Interval mehr Daten erfasst, als in den Buffer passen. Falls es zu viele sind, wird die Geschwindigkeit herunter gesetzt, wenn sie nicht auf dem kleinsten möglichen Wert ist. Zum Schluss wird die Geschwindigkeit wieder um einen Schritt erhöht. Da ein Schritt eine Halbierung oder Verdopplung der Datenrate bedeutet und damit die Lösung nicht den perfekten Punkt für die Datenrate treffen kann, wird dem Programm des Endgeräts freigestellt, ob es die Datenrate nochmals halbiert wird. Durch die Variable `bool didGoLowerOnceAndNowOk` wird sichergestellt, dass die Datenrate nicht höher als vom Endgerät eingestellt wird und, dass der Algorithmus am Ende nicht von der langsamsten Geschwindigkeit erhöht, falls die langsamste Einstellung immer noch zu schnell ist. Falls die Sensordatenrate sehr langsam im Vergleich zum Connection Interval ist, soll nicht gewartet werden, bis die FIFO mit 227 Paketen voll ist. Dann würden einige Connection Intervals gar keine und dann 227 Pakete in Einem geschickt werden. Deswegen wird der FIFO-Threshold wie in Abbildung 5.6 eingestellt. Damit soll die FIFO mindestens ein Mal pro Connection Interval ausgelesen werden.

Bei der Implementierung des Algorithmus mussten weitere Probleme gelöst werden.

Der Treiber der LSM6DSL IMU stellt die Funktion `lsm6dls_fifo_raw_data_get` zum Auslesen der FIFO bereit. Da die FIFO eine Größe von 4096 Byte hat, lässt sie sich mit der Funktion wegen dem Parameter `uint8_t len` nicht komplett auslesen. Das ergibt wenig Sinn und auch das Datenblatt begründet die Einschränkung nicht. Schaut man in die Implementierung in Abbildung 5.7, stellt man fest, dass sie nur eine weitere Funktion aufruft. Der vierte Parameter der Funktion `lsm6dls_read_reg` ist vom Typ `uint16_t`. Somit lässt sich die allgemeinere Funktion stattdessen nutzen, um die komplette FIFO auszulesen. Es ist davon auszugehen, dass es sich um einen Fehler im Treiber handelt.

Unabhängig vom vorherigen Problem gibt es ein Ähnliches bei der Nutzung von EasyDMA (vgl. Abschnitt 4.4.1). Für eine einfache Transaktion von SPI oder I2C ist der Buffer durch die Größe vom 8 Bit Register

```

1 // Pakete die in einem Connection Interval generiert werden
2 #define packets_generated_per_connection_interval(speed_param)
    ((uint16_t)(connection_interval_ms / (1000.0f / ((float)(25 * (0b1 <<
        speed_param))))))
3
4 bool didGoLowerOnceAndNowOk = false;
5 while (packets_generated_per_connection_interval(speed) > buffer_size) {
6     if (speed > 0) {
7         didGoLowerOnceAndNowOk = true;
8         speed--;
9     } else {
10        didGoLowerOnceAndNowOk = false;
11        break;
12    }
13 }
14 if (didGoLowerOnceAndNowOk) speed++;

```

Abbildung 5.5: Berechnung der Geschwindigkeit

```

1 // Es sollen mindestens 1 und maximal 227 Pakete aus der FIFO geholt werden
2 uint16_t packets_till_fifo_full_or_buffer_cleared = MAX(1u,
    MIN(FIFO_MAX_FILL_PACKETS, packets_generated_per_connection_interval(speed)));
3 uint16_t bytes_till_fifo_full_or_buffer_cleared = BYTES_PER_DATA *
    DATAS_PER_PACKET * packets_till_fifo_full_or_buffer_cleared;
4
5 // Setze Threshold. >> 1 wegen verschiedener Einheiten
6 LSM_ERROR_CHECK(lsm6dsl_fifo_watermark_set(dev_ctx,
    bytes_till_fifo_full_or_buffer_cleared >> 1));
7 // Pro Durchgang können maximal 14 Pakete geholt werden
8 packets_per_transaction = MIN(packets_till_fifo_full_or_buffer_cleared,
    MAX_PACKETS_PER_TRANSACTION);
9 // Setze Geschwindigkeit. + 2 wegen verschiedener Einheiten
10 LSM_ERROR_CHECK(lsm6dsl_fifo_data_rate_set(dev_ctx, speed + 2));

```

Abbildung 5.6: Berechnung des Thresholds

```

1 int32_t lsm6dsl_fifo_raw_data_get(lsm6dsl_ctx_t *ctx, uint8_t *buffer, uint8_t
    len) {
2     int32_t ret;
3     ret = lsm6dsl_read_reg(ctx, LSM6DSL_FIFO_DATA_OUT_L, buffer, len);
4     return ret;
5 }

```

Abbildung 5.7: Auslesen der FIFO

MAXCNT begrenzt (siehe Abschnitt 10, 31 und 33 in [Nor17]). Da ein Byte für die Adresse reserviert ist, können 254 Datenpakete pro Transaktion verschickt werden. Diese Einschränkung wurde in Abbildung 5.4 in Zeile 9ff und Abbildung 5.6 in Zeile 7f beachtet. Beim Auslesen der FIFO werden dafür mehrere Transaktionen durchgeführt, bis die FIFO leer ist. Alternativ hätte man die List-Funktion der EasyDMA oder die nRF52840 MCU, die 16 Bit für den Buffer bereitstellt [Nor19], nutzen können.

5.2 Wearable Hardwareumsetzung

Zur Entwicklung der Software und des Hardwareprototypen wurde das nRF52 DK mit der IMU auf den Evaluations Platinen mit Jumperkabeln verbunden. Zur Evaluation sollen die Platinen durch kleinere Alternativen ersetzt werden. Da ein Reflowofen bereitgestellt wurde, konnten selbst kleinste QFN-Bauteile verlötet werden. Die LSM6DSL IMU wurde als QFN-Chip und die MCU in Form der Laird BL651 verbaut. Die Platine von Laird bietet den Vorteil gegenüber der MCU als Chip, dass kein Vorwissen über Antennen nötig ist. Dieser Abschnitt entnimmt seine Informationen den Datenblättern der Bauteile.

An die MCU kann optional ein 32.768 kHz Taktgeber angeschlossen werden. Ohne den Taktgeber wird das Taktignal von anderen Taktgebern synthetisiert. Der Taktgeber muss innerhalb der von der MCU vorgegebenen Spezifikation liegen und benötigt zwei Kondensatoren zum schwingen. Die Kapazität der Kondensatoren C lässt sich mit den Formeln und Werten 5.1 bis 5.5 auf 6 pF bestimmen. [Nor17]

$$CL = \frac{C'^2}{2 * C'} \quad (5.1)$$

$$C' = C + C_{pcb} + C_{pin} \quad (5.2)$$

mit den Werten

$$CL = 7\text{pF}: \text{Lastkapazität des ausgewählten Taktgebers} \quad (5.3)$$

$$C_{pcb} \simeq 4\text{pF}: \text{Kapazitäten durch die Leiterbahnen und Platine} \quad (5.4)$$

$$C_{pin} = 4\text{pF}: \text{Kapazität des Pins der MCU} \quad (5.5)$$

Die Pins P0.22 bis P0.31 der MCU dürfen nicht hochfrequent betrieben werden, da es die BLE-Kommunikation negativ beeinflussen kann. [Nor17] Im Design der Platine wurde deswegen auf diese Pins verzichtet. Der Interrupt2 Pin der IMU konnte dadurch leider platzbedingt nicht verbunden werden. In zukünftigen Versionen der Platine sollte er angeschlossen werden, um der Softwareentwicklung mehr Möglichkeiten offen zu lassen.

An den zwei Spannungspins der IMU und dem Spannungspins der MCU sind 100 μF Kondensatoren angebracht. Sie dienen zur Abkopplung der Komponenten bei Lastschwankungen und sind auch in den Referenzimplementierungen vorhanden. Die ausgewählten LEDs haben eine Vorwärtsspannung von 1.8 V und einen Nennstrom von 2 mA. Um diese Vorgaben bei einem Batterielevel von 2 V bis 3 V einzuhalten, sind jeweils 191 Ohm Widerstände vorgeschaltet. Als Farben der LEDs wurden rot und gelb gewählt, da die Farben eine hohe Wellenlänge haben und dadurch energieärmer sind. Wird die rote LED der Gelben in der Nutzung präferiert steigert sich aus dem selben Grund die Energieeffizienz des Wearables minimal. Zur Programmierung des Chips wurde ein Cortex-M-Debug⁴ Anschluss angebracht. Damit lässt er sich vom nRF52 DK mit dem passenden Kabel programmieren. Um den Energieverbrauch zu evaluieren ist die Spannungsversorgung der Platine über zwei Pins, die 2.54 mm voneinander entfernt sind, offen zugänglich. Alle Komponenten wurden soweit verfügbar in der Größe 0603 gewählt, damit sie mit einem Lötkolben eingelötet oder ausgetauscht werden können. Eine vollständige Liste der Komponenten

⁴ http://infocenter.arm.com/help/topic/com.arm.doc.faqs/attached/13634/cortex_debug_connectors.pdf, aufgerufen am 18.07.2019

ist in Tabelle 5.1 zu finden.

Tabelle 5.1: Bill of Materials (BOM)

Komponente	Typ	Anzahl	Name im Schaltplan
Laird BL652-SA-01	nRF52832 MCU	1	IC1
STMicroelectronics LSM6DSL	LSM6DSL IMU	1	U1
ECS Inc ECS-.327-7-34R	32.768 kHz Taktgeber mit 7 pF Ladekapazität und 50 kOhm ESR	1	Y1
Yageo CC0603CRNPO9BN6R0	6 pF Kondensator für Taktgeber	2	C2, C3
Yageo CC0603JRX7R9BB104	0.1 μ F Kondensator zur Abkopplung	3	C1, C4, C5
Vishay TLMY1000-GS08	Gelbe LED	1	LED1
Vishay TLMS1000-GS08	Rote LED	1	LED2
Vishay CRCW0603191RFKEA	191 Ohm Widerstand für LEDs	2	R1, R2
Keystone Electronics 3008	CR2450 Knopfzellenhalter	1	CR2450
Panasonic EVQ-Q2B03W	Taster	1	S1
CNC-Tech 3220-10-0300-00	Cortex-M-Debug Buchse	1	J2

Bei der Erstellung des Schaltplans, in Abbildung 5.8 zu sehen, musste bereits beachtet werden, wo sich die Pins physisch an den Komponenten befinden, damit sich die Verbindungen nicht überschneiden. Die Komponenten lassen sich mit der letzten Spalte von Tabelle 5.1 zuordnen. Die Komponente VCC.GND sind die besagten Pins zur externen Stromversorgung.

Die Platine wurde nach dem Entwurf in Abbildung 5.9 bei JLCPCB⁵ angefertigt. Die roten und blauen Stellen sind Kupferverbindungen. Die größeren Felder sind die Masseflächen. Die grauen Vias verbinden die Unterseite mit der Oberseite. Während sich auf der Unterseite nur die Batterie befindet, sind die restlichen Komponenten auf der Oberseite. Das freie Rechteck in der linken unteren Ecke enthält kein Metall, da sich hier die Antenne befindet und sonst negativ beeinflusst werden würde. An der Rückseite erkennt man, dass die Größe der Platine durch die Batterie und der Antenne vorgegeben sind. Die Verbindungen zur Stromversorgung sind dicker ausgeführt, damit sie weniger Widerstand bieten. Normalerweise wird die Unterseite der Platine genutzt, wenn Verbindungen sich überschneiden. Die Unterseite ist fast komplett durch die Batterie belegt. Über den Kupferleitungen ist eine isolierende Lackschicht, die vor Kurzschlüssen durch die Batterie schützen würde. Da die Knopfzelle allerdings beim Einlegen und Entfernen über die Oberfläche reibt, ist zu erwarten, dass die Lackschicht sich abträgt und dann nicht mehr schützt. Deswegen mussten Überschneidungen komplett vermieden werden.

Die fertige Platine ist in Abbildung 5.10 zu finden. Zum Größenvergleich sind CR2032 und CR2450 Knopfzellen und das nRF52 DK mit abgebildet.

5.3 Android App

Die App wurde mit dem Android SDK geschrieben, das speziell für Android Smartphones entwickelt wurde. Obwohl Java weiter verbreitet und auch vom SDK unterstützt wird, ist die App in der Sprache Kotlin geschrieben, da sie sinnvolle Erweiterungen zu Java bietet wie Datenklassen und funktionelle Programmierung und der Code gleichzeitig mit Java-Code substituierbar ist.

Die App wurde für Android 9.0 kompiliert und getestet und soll bis Android 5.0 abwärtskompatibel sein. Ältere Versionen werden nicht unterstützt, da die Methode `no.nordicsemi.android.ble.BleManager#`

⁵ <https://jlcpcb.com/>, aufgerufen am 18.07.2019

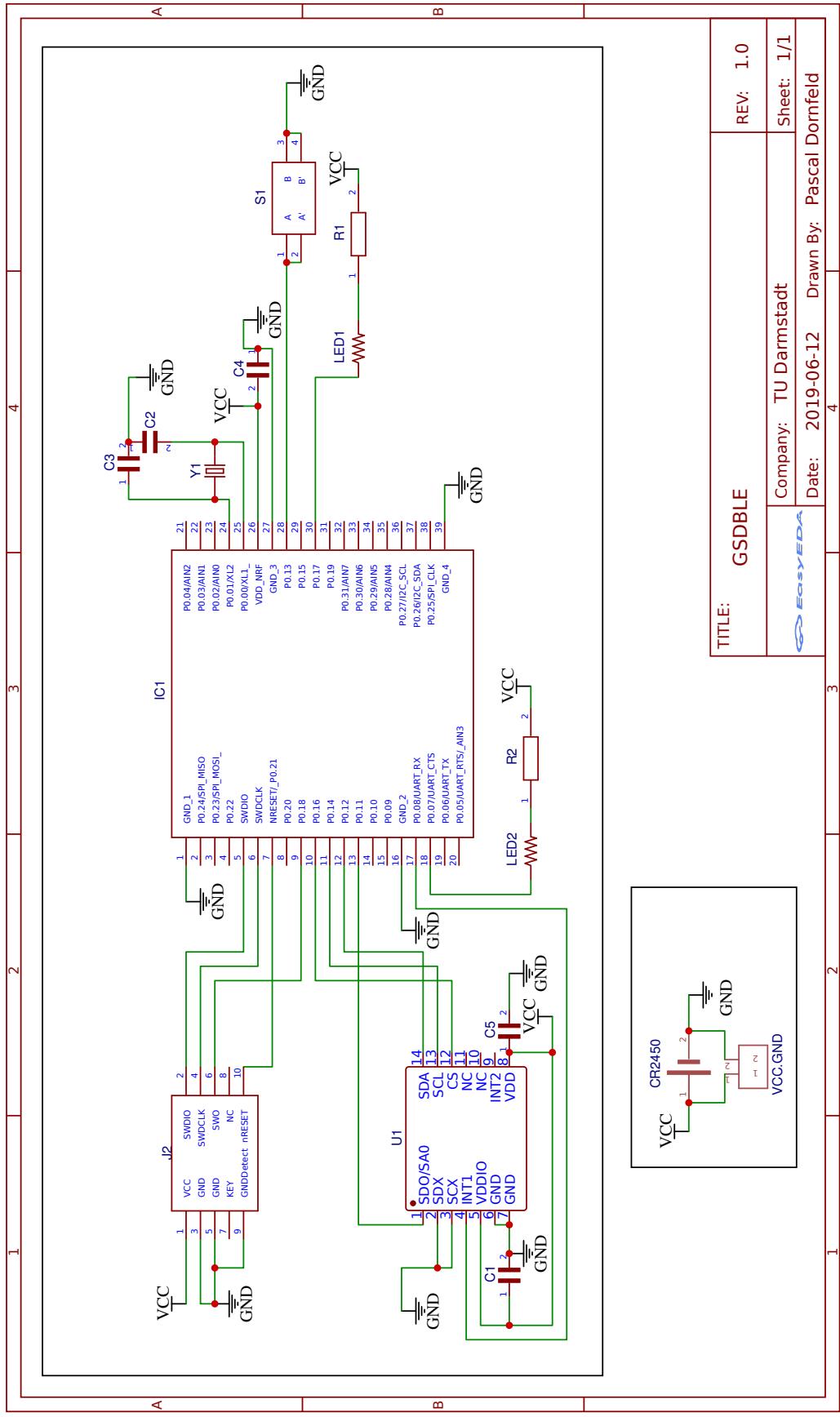
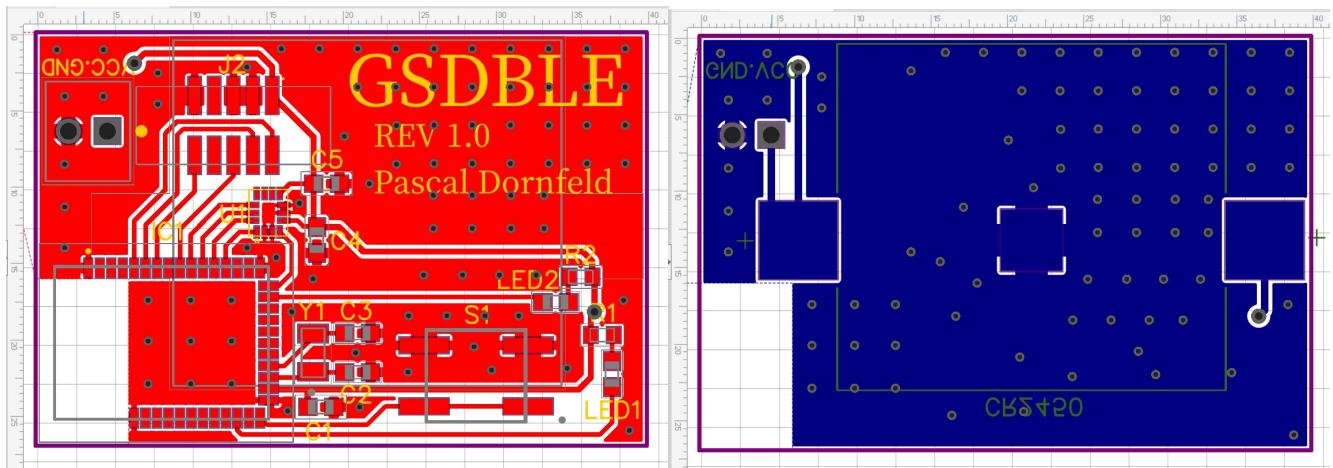


Abbildung 5.8: Schaltplan der Platine



(a) Vorderseite

(b) Rückseite

Abbildung 5.9: Platinenlayout

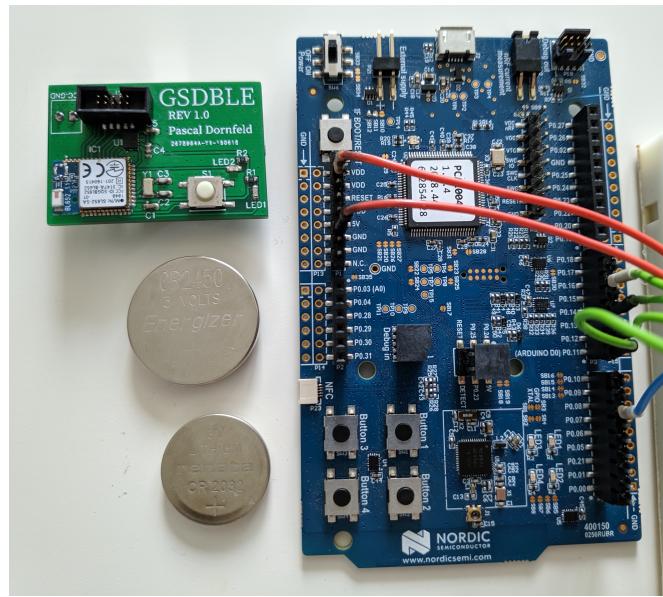


Abbildung 5.10: Fertig bestückte Platine im Größenvergleich

`requestConnectionPriority` Android 5.0 erfordert und gebraucht wird, um den Connection Interval des BLE-Stacks zu setzen.

Beim Start der App wird die `ConnectActivity` gestartet. Sie enthält nur einen Knopf. Drückt der Nutzer auf den Knopf sucht das Smartphone nach Geräten, die mit dem Namen 'GSDBLE' advertise. Der Knopf ist ausgegraut, wenn das Smartphone kein BLE unterstützt oder der Nutzer der App keine Rechte für die Nutzung von Bluetooth gegeben hat. Wenn ein Wearable gefunden wurde, wechselt die App auf die `ConnectionOverviewActivity`. Dabei wird das gefundene Wearable im Intent `EXTRA_DEVICE` übergeben.

Die Activity `ConnectionOverviewActivity` verbindet sich mit dem Wearable, sucht nach den Services und Characteristics und setzt das Notify-Bit bei den Characteristics. Das Android SDK bietet Klassen zur Verbindung mit BLE. Hier hat sich herausgestellt, dass alle Befehle asynchron laufen. Dabei ist problematisch, dass die Befehle nicht in einer Schlange verwaltet werden, sondern ein Befehl ignoriert wird, wenn der Vorherige noch nicht abgeschlossen ist. Deswegen wurde auf die Bibliothek von Nordic⁶ zugegriffen, die die Klassen des SDK besser zugänglich macht.

Die Klasse `Manager` verwaltet die Verbindung. Die Activity `ConnectionOverviewActivity` dagegen verwaltet die graphische Oberfläche. In der Activity werden verschiedene Statistiken graphisch angezeigt. Darunter sind die Sensordaten, aber auch z.B. die empfangenen Daten pro Sekunde. Über Slider lassen sich Parameter einstellen. Die Parameter sind die MTU, der Connection Interval und die Sensordatenrate. Jedes Teil ist davon ein eigenes Fragment aus den Dateien `ConfigFragment` oder `Graphfragment`. Die abstrakte Klasse `Graphfragment` empfängt Daten und fügt sie in die Liste `internalData` ein. Wenn die Liste zu voll ist, werden die ältesten Daten entfernt. Die Daten sollen dann von der Graphenbibliothek angezeigt werden. Da sie in einem anderen Thread arbeitet und es sonst zu Race Conditions kommt, wird von der Liste `internalData` ein Snapshot in `data` für die Graphenbibliothek erstellt.

Die Werte der Characteristics sind in den Modelklassen `ImuConfig` und `ImuData` realisiert. Sie bieten die Möglichkeit die Werte in ein `ByteArray` umzuwandeln und wieder zurück.

5.4 Zusammenfassung

Die Einrichtung der Entwicklungsumgebung der MCU ist leider kompliziert und bedarf viele Umwege. Die Installation von Android Studio dagegen lief einwandfrei. Von den Entwickler der MCU und Drittanbieter sollte hier nachgebessert werden.

Um den Datenstau zu umgehen, der durch die drei Buffer in MCU und IMU entsteht, berechnet ein Algorithmus die Parameter zur korrekten Umfüllung. Der Hardwareentwurf war erfolgreich verlaufen, sodass ein Wearable zur Evaluation bereit steht. In der Android App werden dafür Daten gesammelt, die statistisch ausgewertet werden können.

⁶ <https://github.com/NordicSemi/Android-BLE-Library>, aufgerufen am 18.07.2019



6 Evaluation

6.1 Goal and Methodology

ziel ist lange batterielebensdauer durch geringe stromaufnahme. vielleicht ein paper zur erforderlichen sampling rate, dass gesten korrekt erkannt werden.

6.2 Evaluation Setup

6.3 Evaluation Results

das hier ist der stromverbrauch. das hier ist die erreichte sampling rate. so skaliert das mit der anzahl der sensoren

6.4 Analysis of Results



7 Conclusions

7.1 Summary

7.2 Contributions

7.3 Future Work

energieverbrauch beim smartphone

andere unkonventionelle arten zur datenübertragung untersuchen. durch vibrierende knochen reden.
btmesh implementieren und vergleichen, wie stark der energieverbrauch sich ändert. es gibt da die library von nordic

nRF52811

project zephyr oder zukünftig mbed

list feature für easydma

mix aus daten verwerfen und sensor langsamer machen

eInk um verbindung anzuzeigen

7.4 Final Remarks



Literaturverzeichnis

- [ABCO12] Fatemeh Abyarjoo, Armando Barreto, Jonathan Cofino, and Francisco Ortega. Implementing a sensor fusion algorithm for 3d orientation detection with inertial/magnetic sensors. 01 2012.
- [aco17a] aconno. *Datenblatt ACN52832*, August 2017. Online erhältlich unter https://1897079276.rsc.cdn77.org/wp-content/uploads/2018/08/ACN52832_data_sheet_v1.3.pdf; abgerufen am 09. Juli 2019.
- [aco17b] aconno. *Datenblatt ACNSENSA*, November 2017. Online erhältlich unter https://1897079276.rsc.cdn77.org/wp-content/uploads/2018/08/ACNSENSA_datasheet.pdf; abgerufen am 07. Juli 2019.
- [Ada19] Adafruit Industries. Introduction to Bluetooth Low Energy, März 2019. Online erhältlich unter <https://cdn-learn.adafruit.com/downloads/pdf/introduction-to-bluetooth-low-energy.pdf?timestamp=1562415841>; abgerufen am 06. Juli 2019.
- [ALE16] ALEX THE GIANT auf learn.sparkfun.com. Reducing Arduino Power Consumption, November 2016. Online erhältlich unter <https://learn.sparkfun.com/tutorials/reducing-arduino-power-consumption/all>; abgerufen am 09. Juli 2019.
- [ARD] ARDUINO. ARDUINO PRIMO CORE. Online erhältlich unter <https://store.arduino.cc/arduino-primo-core>; abgerufen am 09. Juli 2019.
- [Ard16] Arduino. *Datenblatt Primo Core*, September 2016. Online erhältlich unter https://www.arduino.cc/en/uploads/Main/ARDUINO_PRIMO_CORE_V02_SCH.pdf; abgerufen am 09. Juli 2019.
- [Arm19] Arm Limited. Network connectivity in Mbed OS, 2019. Online erhältlich unter <https://os.mbed.com/docs/mbed-os/v5.13/reference/networking.html>; abgerufen am 11. Juli 2019.
- [Ben19] Bendel, Prof. Dr. Oliver. Wearables, Januar 2019. Online erhältlich unter <https://wirtschaftslexikon.gabler.de/definition/wearables-54088/version-368816>; abgerufen am 21. April 2019.
- [Blu] Bluetooth SIG. Qualified Mesh Products. Online erhältlich unter <https://www.bluetooth.com/bluetooth-technology/topology-options/le-mesh/mesh-qualified/>; abgerufen am 01. März 2019.
- [Bos] Bosch. Produktseite BMI260. Online erhältlich unter https://www.bosch-sensortec.com/bst/products/all_products/bmi260; abgerufen am 09. Juli 2019.
- [Bos18a] Bosch. *Datenblatt BMI160*, Oktober 2018. Online erhältlich unter https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BMI160-DS000.pdf; abgerufen am 09. Juli 2019.
- [Bos18b] Bosch. *Datenblatt BMP280*, Januar 2018. Online erhältlich unter https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BMP280-DS001.pdf; abgerufen am 09. Juli 2019.

- [Bos19] Bosch. *Datenblatt BMM150*, April 2019. Online erhältlich unter https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BMM150-DS001.pdf; abgerufen am 09. Juli 2019.
- [Cho] Chordata. Motion Capture system that you can build yourself. Online erhältlich unter <https://hackaday.io/project/27519-motion-capture-system-that-you-can-build-yourself>; abgerufen am 09. Juli 2019.
- [dig] digikey.com. TMP007 Infrared Thermopile Sensor. Online erhältlich unter <https://www.digikey.com/en/product-highlight/t/texas-instruments/tmp007-infrared-thermopile-sensor>; abgerufen am 09. Juli 2019.
- [Goo14] Google. Android-Quellcode bt_target.h, Dezember 2014. Online erhältlich unter https://android.googlesource.com/platform/external/bluetooth/bluedroid/+/master/include/bt_target.h#1428; abgerufen am 13. Juli 2019.
- [Int19] International Data Corporation. Absatz von Wearables weltweit in den Jahren 2014 bis 2018 (in Millionen Stück), März 2019. Online erhältlich unter <https://de.statista.com/statistik/daten/studie/515723/umfrage/absatz-von-wearables-weltweit>; abgerufen am 21. April 2019.
- [Inv16] InvenSense. *Datenblatt MPU-9250*, Juni 2016. Online erhältlich unter <https://www.invensense.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf>; abgerufen am 09. Juli 2019.
- [Lai19] Laird. *Datenblatt BL651*, Februar 2019. Online erhältlich unter https://connectivity-staging.s3.us-east-2.amazonaws.com/2019-02/CS-DS-BL651%20v1_1.pdf; abgerufen am 09. Juli 2019.
- [mad19] madcoffee auf home-automation-community.com. ARDUINO LOW POWER - HOW TO RUN ATMEGA328P FOR A YEAR ON COIN CELL BATTERY, Februar 2019. Online erhältlich unter <http://www.home-automation-community.com/arduino-low-power-how-to-run-atmega328p-for-a-year-on-coin-cell-battery/>; abgerufen am 09. Juli 2019.
- [Mau19] Mauricio Farina auf devzone.nordicsemi.com. NRF_LOG not working on Segger Embedded Studio, April 2019. Online erhältlich unter https://devzone.nordicsemi.com/f/nordic-q-a/45985/nrf_log-not-working-on-segger-embedded-studio; abgerufen am 11. Juli 2019.
- [mic] microcontroller.net. Serial Peripheral Interface. Online erhältlich unter https://www.mikrocontroller.net/articles/Serial_Peripheral_Interface; abgerufen am 07. Juli 2019.
- [mou] mouser.com. acnSENSA. Online erhältlich unter <https://www.mouser.de/ProductDetail/aconno/acnSENSA?qs=sGAEpiMZZMve4%2FbfQkoj%252BGMseiRPTSjeOSSloCsufA0%3D>; abgerufen am 08. Juli 2019.
- [MT12] K. Mikhaylov and J. Tervonen. Evaluation of power efficiency for digital serial interfaces of microcontrollers. In *2012 5th International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5, May 2012.
- [Nor] Nordic Semiconductor. Explore the nRF52 Series. Online erhältlich unter <https://www.nordicsemi.com/Products/Low-power-short-range-wireless>; abgerufen am 09. Juli 2019.

- [Nor16] Nordic Semiconductor. Bluetooth low energy Characteristics, a beginner's tutorial, März 2016. Online erhältlich unter <https://devzone.nordicsemi.com/nordic/short-range-guides/b/bluetooth-low-energy/posts/ble-characteristics-a-beginners-tutorial>; abgerufen am 06. Juli 2019.
- [Nor17] Nordic Semiconductor. *nRF52832 Product Specification v1.4*, Oktober 2017. Online erhältlich unter https://infocenter.nordicsemi.com/pdf/nRF52832_PS_v1.4.pdf; abgerufen am 14. Juli 2019.
- [Nor19] Nordic Semiconductor. *nRF52840 Product Specification v1.1*, Februar 2019. Online erhältlich unter https://infocenter.nordicsemi.com/pdf/nRF52840_PS_v1.1.pdf; abgerufen am 14. Juli 2019.
- [NXP14] NXP Semiconductors. *I2C-bus specification and user manual*, April 2014. Online erhältlich unter <https://www.nxp.com/docs/en/user-guide/UM10204.pdf>; abgerufen am 07. Juli 2019.
- [Pat19a] Patrick Schnabel. Bluetooth Mesh, Juni 2019. Online erhältlich unter <https://www.elektronik-kompendium.de/sites/kom/2210201.htm>; abgerufen am 13. Juli 2019.
- [Pat19b] Patrick Schnabel. DMS - Dehnungsmessstreifen, Juni 2019. Online erhältlich unter <https://www.elektronik-kompendium.de/sites/bau/1404151.htm>; abgerufen am 30. Juni 2019.
- [Pat19c] Patrick Schnabel. Lithium-Ionen-Akkus, Juni 2019. Online erhältlich unter <https://www.elektronik-kompendium.de/sites/bau/0810281.htm>; abgerufen am 07. Juli 2019.
- [Pat19d] Patrick Schnabele. MEMS - Micro-Electro-Mechanical Systems, Juni 2019. Online erhältlich unter <https://www.elektronik-kompendium.de/sites/bau/1503041.htm>; abgerufen am 30. Juni 2019.
- [PPLA17] G. D. Putra, A. R. Pratama, A. Lazovik, and M. Aiello. Comparison of energy consumption in wi-fi and bluetooth communication in a smart building. In *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 1–6, Jan 2017.
- [sen18] sensoren.info. MEMS - Micro-Electro-Mechanical Systems, Januar 2018. Online erhältlich unter <https://www.sensoren.info/mikromechBeschl.png>; abgerufen am 30. Juni 2019.
- [Spa16] SparkFun. Datenblatt *nRF52832 Breakout*, August 2016. Online erhältlich unter https://cdn.sparkfun.com/assets/learn_tutorials/5/4/9/sparkfun-nrf52832-breakout-schematic-v10.pdf; abgerufen am 09. Juli 2019.
- [STM15] STMicroelectronics. Datenblatt *LSM9DS1*, März 2015. Online erhältlich unter <https://www.st.com/resource/en/datasheet/lsm9ds1.pdf>; abgerufen am 09. Juli 2019.
- [STM16] STMicroelectronics. Datenblatt *HTS221*, August 2016. Online erhältlich unter <https://www.st.com/resource/en/datasheet/hts221.pdf>; abgerufen am 09. Juli 2019.
- [STM17a] STMicroelectronics. Datenblatt *LIS3MDL*, Mai 2017. Online erhältlich unter <https://www.st.com/resource/en/datasheet/lis3mdl.pdf>; abgerufen am 09. Juli 2019.
- [STM17b] STMicroelectronics. Datenblatt *LSM6DS3*, August 2017. Online erhältlich unter <https://www.st.com/resource/en/datasheet/lsm6ds3.pdf>; abgerufen am 07. Juli 2019.
- [STM17c] STMicroelectronics. Datenblatt *LSM6DSL*, September 2017. Online erhältlich unter <https://www.st.com/resource/en/datasheet/lsm6ds1.pdf>; abgerufen am 09. Juli 2019.

- [Tex] Texas Instruments. CC2650STK. Online erhältlich unter <https://www.ti.com/tool/CC2650STK>; abgerufen am 07. Juli 2019.
- [Tex15] Texas Instruments. *Datenblatt TPS2291xx*, November 2015. Online erhältlich unter <https://www.ti.com/lit/ds/symlink/tps22913.pdf>; abgerufen am 09. Juli 2019.
- [Tex16a] Texas Instruments. *Datenblatt HDC1000*, Januar 2016. Online erhältlich unter <https://www.ti.com/lit/ds/symlink/hdc1000.pdf>; abgerufen am 09. Juli 2019.
- [Tex16b] Texas Instruments. Generic Access Profile (GAP), 2016. Online erhältlich unter http://dev.ti.com/tirex/content/simplelink_cc2640r2_sdk_1_40_00_45/docs/blestack/ble_user_guide/html/ble-stack-3.x/gap.html#; abgerufen am 06. Juli 2019.
- [Tex16c] Texas Instruments. Logical Link Control and Adaptation Layer Protocol (L2CAP), 2016. Online erhältlich unter http://dev.ti.com/tirex/content/simplelink_cc2640r2_sdk_1_35_00_33/docs/ble5stack/ble_user_guide/html/ble-stack/l2cap.html#sec-l2cap; abgerufen am 13. Juli 2019.
- [Tex17] Texas Instruments. *Datenblatt OPT3001*, November 2017. Online erhältlich unter <https://www.ti.com/lit/ds/symlink/opt3001.pdf>; abgerufen am 09. Juli 2019.
- [Var14a] Varta. *Datenblatt DS6032*, September 2014. Online erhältlich unter https://products.varta-microbattery.com/applications/mb_data/documents/data_sheets/DS6032.pdf; abgerufen am 13. Juli 2019.
- [Var14b] Varta. *Datenblatt DS6430*, September 2014. Online erhältlich unter https://products.varta-microbattery.com/applications/mb_data/documents/data_sheets/DS6430.pdf; abgerufen am 13. Juli 2019.
- [Var14c] Varta. *Datenblatt DS6450*, September 2014. Online erhältlich unter https://products.varta-microbattery.com/applications/MB_DATA/DOCUMENTS/DATA_SHEETS/DS6450.pdf; abgerufen am 13. Juli 2019.
- [VGC19] VGChartz. Verkaufszahlen der weltweit meistverkauften Spielkonsolen bis Dezember 2018 (in Millionen Stück), März 2019. Online erhältlich unter <https://de.statista.com/statistik/daten/studie/160549/umfrage/anzahl-der-weltweit-verkaerten-spielkonsolen-nach-konsolentypen/>; abgerufen am 21. April 2019. Bearbeitet.
- [Vis15] Vishay. *Datenblatt sip32401a*, Juni 2015. Online erhältlich unter <https://www.vishay.com/docs/63705/sip32401a.pdf>; abgerufen am 07. Juli 2019.
- [VPdN⁺¹⁰] Benedetto Vigna, Fabio Pasolini, Roberto de Nuccio, Macro Capovilla, Luciano Prandi, and Fabio Biganzoli. Low cost silicon coriolis' gyroscope paves the way to consumer imu. In Evgeni Gusev, Eric Garfunkel, and Arthur Dideikin, editors, *Advanced Materials and Technologies for Micro/Nano-Devices, Sensors and Actuators*, pages 67–74, Dordrecht, 2010. Springer Netherlands.
- [ZJM17] Longhao Zou, Ali Javed, and Gabriel-Miro Muntean. Smart mobile device power consumption measurement for video streaming in wireless environments: Wifi vs. lte. pages 1–6, 06 2017.