**HEIDELBERG UNIVERSITY**
Institute of Computer Engineering
Chair on Application Specific Computing

# Sound Glove

## A Wireless Hand Motion Capturing Device for Audio Applications

Henning Lohse

Supervisor: Dr. Andreas Kugel
Second Assessor: Prof. Dr. Robert Strzodka

A thesis submitted in partial fulfillment for the degree of
Master of Science in Computer Engineering
Granstedt, August 23 2015

## Abstract

This work presents a wireless sensor glove used to control audio software via finger flexing and the hand's orientation and position. Musicians using this glove are no longer restricted to work in the vicinity of their audio devices but can move around freely.
The glove hardware was designed from scratch. It includes a microcontroller, a Bluetooth module, a circuit to recharge the battery using USB, as well as multifunctional three-axis sensors that were calibrated. The PC application receives sensor data via Bluetooth and computes the hand's orientation and position. Together with finger flexing, these values are used to control audio software.
The aspects explored are orientation and position accuracy during different types of motion, latency between motion and changes in the audio software, as well as the device's power consumption and hardware size. Results show that without additional work, measurements are accurate only during either slow or fast motions. Latency is low if the Bluetooth module does not use power-saving features. These are however necessary for a reduced power consumption allowing the use of a compact battery for multiple hours of runtime. The hardware fits into a textile glove and can be downsized even further.



## Zusammenfassung

Diese Arbeit stellt einen kabellosen Handschuh mit Sensoren vor, welche Fingerbiegungen und die Lage und Position der Hand messen, um Audio-Software zu steuern. Musiker können sich hiermit frei bewegen und sind nicht an stationäre Geräte gebunden.
Die Hardware wurde von Grund auf neu entworfen. Sie beinhaltet einen Microcontroller, ein Bluetooth-Modul, eine Schaltung zum Laden der Batterie per USB, sowie 3-achsige Multifunktions-Sensoren, welche kalibriert wurden. Die PC-Applikation empfängt die Sensordaten und berechnet die Lage und Position der Hand. Diese Werte werden, zusammen mit der Fingerbiegung, zur Steuerung der Audio-Software verwendet.
Untersucht wurden die Genauigkeit von Lage und Position für verschiedene Bewegungen, die Verzögerung zwischen Bewegungen und Änderungen in der Audio-Software, sowie der Energieverbrauch und die Größe der Hardware. Die Ergebnisse zeigen, dass die Messungen, ohne zusätzliche Arbeit, nur entweder für langsame oder schnelle Bewegungen genau genug sind. Verzögerungen sind niedrig, solange das Bluetooth-Modul keine Stromsparfunktionen verwendet. Diese sind allerdings notwendig, um den Stromverbrauch so zu senken, dass kompakte Batterien für mehrere Stunden Laufzeit genügen. Die Hardware passt in einen Textil-Handschuh und kann noch weiter verkleinert werden.

# *Acknowledgements*

I would like to acknowledge the support and input from a number of people who helped me to bring this thesis into being.

Firstly, I would like to thank my supervisor, Dr. Andreas Kugel, for offering me the opportunity to work on this self-proposed topic. His guidance in hardware component selection, as well as his quick feedback regarding latency, power and AHRS issues, have been substantial for me in identifying and exploring important aspects of this work.

I would furthermore like to thank Prof. Dr. Robert Strzodka for taking the time to examine and criticize my work.

Special thanks go to Dr. Andreas Wurz. His extensive experience with embedded systems aided me greatly in solving hardware problems and debugging the firmware. Without his skillful soldering techniques, this work would have been difficult to realize.

I especially thank Jan-Lukas Tirpitz. In 2014, he approached me to work with him on the DJ Glove project, the spiritual predecessor and origin of this work. During our countless fruitful discussions and working sessions, I learned about the components, technologies and requirements involved. These insights are fundamental for this thesis.

I would also like to thank Diane Le Naour, Katrin Lohse and Felicitas Hackmann for supporting me and providing valuable proofreading feedback.

# Contents

# Abbreviations

| | |
|---|---|
| **DAW** | **D**igital **A**udio **W**orkstation |
| **MIDI** | **M**usical and **I**nstrument **D**igital **I**nterface |
| **MEMS** | **M**icro-**e**lectro**m**echanical **S**ystem |
| **LiPo** | **Li**thium **Po**lymer battery |
| **PCB** | **P**rinted **C**ircuit **B**oard |
| **GPS** | **G**lobal **P**ositioning **S**ystem |
| **IMU** | **I**nertial **M**easurement **U**nit |
| **AHRS** | **A**ttitude and **H**eading **R**eference **S**ystem |
| **MARG** | **M**agnetic, **A**ngular **R**ate and **G**ravity sensor |
| **FPU** | **F**loating **P**oint **U**nit |
| **AES** | **A**dvanced **E**ncryption **S**tandard |
| **FEC** | **F**orward **E**rror **C**orrection |
| **HCI** | **H**ost **C**ommand **I**nterface |
| **L2CAP** | **L**ogical **L**ink **C**ontrol and **A**daptation **P**rotocol |
| **SDP** | **S**ervice **D**iscovery **P**rotocol |
| **IC** | **I**ntegrated **C**ircuit |
| **I2C** | **I**nter-**I**ntegrated **C**ircuit |
| **SPI** | **S**erial **P**eriphal **I**nterface |
| **UART** | **U**niversal **A**synchronous **R**eceiver **T**ransmitter |
| **TI** | **T**exas **I**nstruments |
| **FLL** | **F**requency-**L**ocked **L**oop |
| **DMA** | **D**irect **M**emory **A**ccess |
| **ISR** | **I**nterrupt **S**ervice **R**outine |
| **ADC** | **A**nalog-to-**D**igital **C**onverter |
| **JTAG** | **J**oint **T**est **A**ction **G**roup |
| **BSL** | **B**oot**s**trap **L**oader |
| **FFC** | **F**lexible **F**lat **C**able |
| **CTS** | **C**lear **T**o **S**end |
| **RTS** | **R**eady **T**o **S**end |
| **LDO** | **L**ow-**D**ropout **R**egulator |
| **PWM** | **P**ulse-**W**idth **M**odulation |
| **LSB** | **L**east **S**ignificant **B**it |
| **RGB** | **R**ed, **G**reen, **B**lue |
| **RTT** | **R**ound-**t**rip **T**ime |

# Chapter 1

# Introduction

The digital production of music is assisted by digital audio workstation (DAW) software. This software, as depicted in figure 1.1, provides functionality to record, manipulate and combine audio tracks.


FIGURE 1.1: Digital audio workstation (DAW) [1].

Features include common control elements, like play and pause (a). Waveforms of audio tracks are visualized (b), as well as notes of virtual instruments (c). Furthermore, effects and filters can be configured to manipulate the audio tracks (d).

However, audio track control is less practical using mouse and keyboard compared to using a mixing desk. Additionally, real instruments often need to be recorded. For these purposes, the Musical Instrument Digital Interface (MIDI) has been developed. This protocol defines how, amongst others, instruments, notes and control signals have to be encoded and interpreted.

MIDI allowed the development of control and recording devices. Figure 1.2 shows a MIDI controller board (left) and a MIDI cable with ports (right).


FIGURE 1.2: MIDI controller board (left) and MIDI cable with ports (right) [1].

MIDI controller boards can be used as mixing desks. They provide a set of buttons, sliders and potentiometers (a). Specific identifiers are assigned to these controls. This allows the DAW to map the controls' values to specific elements in the software. For example, pushing a slider upwards can be mapped to increase the volume. In addition, a keyboard is provided to synthesize virtual instruments (b).

MIDI devices require special hardware ports and cables (c). They are found in e.g. MIDI guitars for guitar note recording, as well as sound cards in computers. However, MIDI ports can also be virtual. This means DAW software can communicate with other software that uses MIDI ports via virtual, software-only MIDI ports.

For live performances, especially for electronic music, DAWs with MIDI controller boards are common. Their disadvantage is the limit on the artist's freedom of motion. The artist must stay in front of his equipment to control the audio. This led to the development of special devices providing mobility by using virtual MIDI ports.

## 1.1 Kinect Depth Camera

Microsoft's Kinect depth camera comes with a software development kit. This allows the development of arbitrary applications using the system's body and feature detection. Feature detection includes "full head tracking simultaneously with body tracking (...) finger tracking (...)", as well as skeleton detection [2]. Implementations exist to make the Kinect features available as MIDI signals [3].

However, depending on the model, latency is about 20ms to 102ms. For a musician, this is a noticeable delay between motion and audio output. Especially since additional steps, like effect manipulation, occur afterwards and add to total latency.

Furthermore, as typical for computer vision applications, feature detection is not always accurate. For example, an open hand might not be detected immediately. The Kinect is also stationary. The artist must not leave the field of view and no obstructions must enter the line of sight between camera and artist.

## 1.2 Mi.Mu Gloves

Mi.Mu developed the glove depicted in figure 1.3. It utilizes micro-electromechanical system (MEMS) sensors to determine the hand's orientation. Flex sensors are placed on most fingers' joints to determine finger flexing. Capacitive fabrics allow the detection of touching fingers and can simulate buttons by pressing on certain areas of the fabric. An LED and a vibration motor provide feedback to the user. The glove operates wireless using WiFi and a 2000mAh lithium polymer battery (LiPo). Software exists to use the data for audio applications. E.g. rotating the hand can control effects in a DAW [4].

FIGURE 1.3: Mi.Mu Glove [5].

This solution is portable in the sense that the artist is not restricted to some space in front of the equipment or camera. The artist can move around freely. Features like finger flexing or open and close hands can be determined precisely.

The MEMS sensors, however, are placed on the wrist below the forearm. This means the hand's orientation can only be determined for motions including forearm rotation. Placing the hardware directly on the hand is not possible due to board and LiPo sizes.

There is no possibility to determine the hand's position. The hand's position can be of interest to the user, as it provides additional degrees of freedom to control DAW software. Furthermore, the LiPo requires an external charging device.

## 1.3 Design Goals

Inspired by the Mi.Mu Glove, the goal of this work is to realize a custom printed circuit board (PCB) design and corresponding software to implement and explore the following features for a wireless MIDI controller sensor glove:

1. Absolute 3D orientation detection of the hand using MEMS sensors on the PCB.
2. Hand 3D position detection using MEMS sensors on the PCB. May be relative to a body part, but must not use external reference systems.
3. Hand acceleration and velocity detection using MEMS sensors on the PCB.
4. Finger flex detection for per-finger control and detection of an open or closed hand.
5. PC application to translate sensor data to MIDI signals.
6. A readout rate of up to 500Hz, making updates hardly noticeable by the user.
7. A low-power mode with a readout rate of 100Hz, increasing device run-time.
8. PCB size small enough to fit into a glove on the back of the hand.
9. Wireless operation.
10. Low total latency, which is the delay between sensor readouts and availability of MIDI signals. Must be below 20ms to be suitable for audio applications.
11. LiPo rechargeable via USB.
12. Low LiPo capacity requirement, allowing for small-sized LiPos fitting into a glove on the back of the hand. The run-time must be at least six hours, which the author considers a useful time to work with the glove.
13. Two gloves usable via a single wireless interface without a significant increase in total latency. Must be below 20ms for both gloves to be suitable for audio applications.
14. Vibration motor integration.
15. Capacitive touch evaluation.

## 1.4 Chapter Overview

The following chapter 2 explores which sensors are required for motion detection and how their data need to be processed. Chapter 3 evaluates wireless technologies with respect to the design goals.

In chapter 4, the process of hardware selection is described. Chosen components and their implications on the design goals are presented. Consequently, chapter 5 introduces the software architecture and how the glove and computer software interact. Necessary sensor calibrations are discussed in chapter 6.

With a working system, chapter 7 provides experiment setups and results. Chapter 8 discusses experiment results to verify whether the design goals have been met.

Finally, chapter 9 summarizes this work and provides options for future work.

# Chapter 2

# Motion Sensors

The design goals state it is desired to know the hand's orientation and position. This would provide the user with six degrees of freedom to control audio applications. The acceleration and velocity of linear and angular motions is desired, too. Additionally, the fingers' flex' should be provided.

This chapter present the flex sensor, as well as concepts and sensors used to determine the hand's orientation. Furthermore, the hand's position can be determined by additional sensors placed on the arm. Lastly, acceleration and velocity values can be calculated.

## 2.1 Finger Flex

As described in section 1.1, optical solutions have a high latency and are erroneous. Reliable solutions are thin conductive sensor stripes that can be placed on the fingers. When flexing the sensor stripe, their resistance changes immediately. To determine the flex, only an ADC is required to measure the voltage between input and output of the sensor stripe. The flex sensor stripe used in this work is the Spectra Symbol 4.5" Flex Sensor, as depicted in figure 2.1.

FIGURE 2.1: Spectra Symbol 4.5" Flex Sensor [6].

## 2.2 Hand Orientation

The orientation of an object can be described using Tait–Bryan angles, as seen in figure 2.2. These three angles Ψ, Θ and Φ define the angles of rotation per axis. They describe the object's orientation as the frame with the axes $X$, $Y$ and $Z$ relative to a reference frame with the axes $x$, $y$ and $z$.

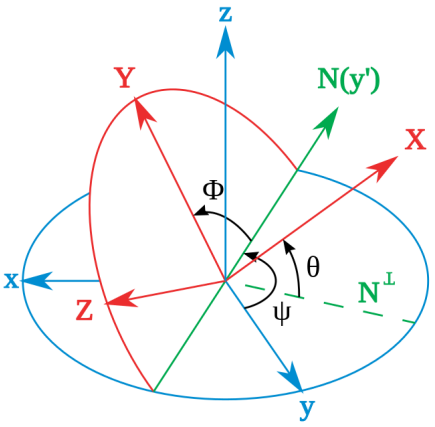


FIGURE 2.2: Tait-Bryan angles Ψ, Θ and Φ in z–y'–x" order [7].

In flight dynamics, Tait–Bryan angles are commonly referred to as yaw (Ψ), pitch (Θ) and roll (Φ) if applied in z–y'–x" order. That means that yaw is applied first. This rotates the object's y axis away from the principal y axis to y'. With applying pitch, the object's x axis rotates to x". Roll then rotates the objects around the x" axis. This leaves the object's frame with the axes X, Y and Z, as shown in figure 2.2.

Figure 2.3 visualizes this behaviour for a hand. The z–y'–x" order ensures that yaw always describe a "wiping", pitch a "flapping" and roll a rolling rotatory motion.

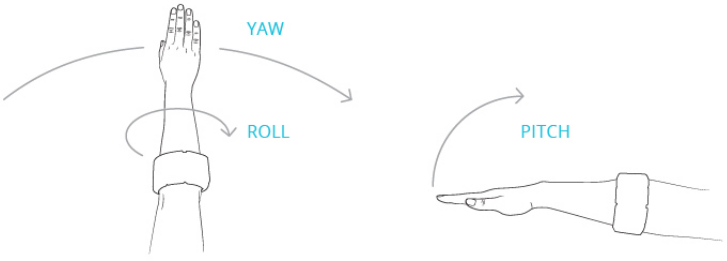


FIGURE 2.3: Hand orientation by roll, pitch and yaw angles [8].

In this work, these three angles are determined and made available to the user.

### 2.2.1 Gyroscope

Theoretically, the orientation of the hand can be determined by continuously integrating and summing up angular velocities provided by a gyroscope sensor. However, such sensors are inherently affected by static and dynamic errors. Static errors are offsets from the actual velocity when stationary. Dynamic errors are the same if not stationary but in motion. While, depending on the sensor, they might be small, they will never be zero. When integrating angular velocity, errors accumulate over time. Resulting angles would be off by multiple degrees after just a few seconds [9].

It is, however, possible to adjust the integration of angular velocity. The field of flight dynamics developed systems performing such tasks. They are called Inertial Measurement Units (IMU) and Attitude and Heading Reference Systems (AHRS).

### 2.2.2 IMU

Inertial Measurement Units typically combine a gyroscope with an accelerometer. The latter is used to measure linear acceleration. Since the Earth's gravitation $G$ applies as linear acceleration, the system can use the accelerometer to determine where down (Earth's surface) and up (sky) are. This can be used as a reference.

For this work, if the hand rests parallel to the Earth's surface with the palm facing down, roll and pitch are expected to be zero. In this case, the Earth's gravitation only applies to the up-down-axis. Other axes will be affected once roll and pitch change. IMUs reference $G$ to adjust the integration of angular velocity for roll and pitch angles. Yet further references are required to adjust the yaw angle.

### 2.2.3 AHRS with MARG

Attitude and Heading Reference Systems are IMUs with a magnetometer. The resulting sensor array is commonly called a magnetic, angular rate and gravity sensor (MARG). The magnetometer is used to measure the direction and magnitude of magnetic fields. The Earth's magnetic field's poles can be used as references.

To determine the poles' directions, an AHRS must take the Earth's magnetic field's lines into account. The lines' inclinations, the vertical component relative to the Earth's surface, and declinations, the horizontal component, depend on the location. Figure 2.4 illustrates this. Therefore, proximity to the magnetic north or south pole influences angles to magnetic east and west. By referencing gravity using the accelerometer, an AHRS is able to determine the poles' directions. This delivers the magnetic field line's inclination and declination as references in the vertical and horizontal plane. Using these, the integration of angular velocity can be adjusted for roll, pitch and yaw. Roll and pitch have two sources of references for adjustments, improveing their accuracy. Yaw, however, relies solely on the magnetometer for corrections.

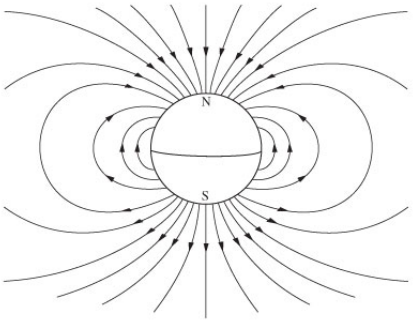


FIGURE 2.4: Varying directions of the Earth's magnetic field lines [10].

### 2.2.4 Madgwick's AHRS

For this work, an AHRS developed by S. Madgwick is used [11]. It combines measurements from a three-axis MARG. The result is a set of quaternions, an orientation representation which is converted to the Tait–Bryan angles roll, pitch and yaw.

Madgwick shows that his algorithm exceeds accuracy of a proprietary one while requiring less computational power. Noteworthy are the static and dynamic errors as a function of sampling rate, as seen in figure 2.5. By using a sampling rate of 10Hz, a static error of $< 2°$ and dynamic error of $< 7°$ is achieved. At a 100Hz sampling rate, both errors drop to less than $1°$. The low errors at low sampling rates allow the construction of a low-power embedded AHRS, which is explored experimentally in chapter 7 of this work.

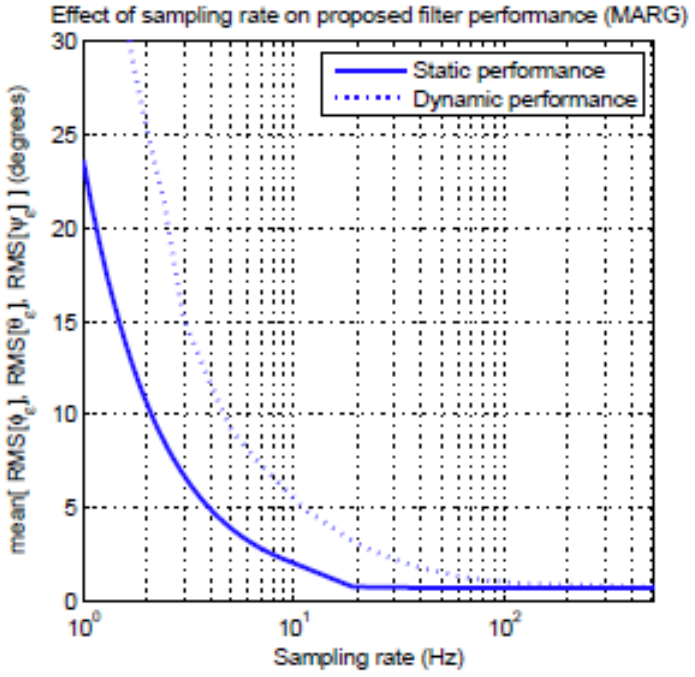


FIGURE 2.5: Static and dynamic orientation error as a function of sampling rate [11].

It is important to note that the resulting orientation is relative to a "reference frame". In this case, this means that the degrees of roll and pitch are relative to the Earth's surface. For example, a roll of 0° and pitch of 0° occurs if the object is parallel to the surface. Increasing roll to 180° rotates the object to be upside-down, whereas increasing pitch to 90° aligns the object to the Earth's gravity direction, perpendicular to the surface. Yaw on the other hand equals 0° if the object is pointing to the Earth's magnetic north. Steps of 90° makes the object point to east, south and west respectively.

The algorithm also compensates gyroscope bias drift. This is important, as gyroscope measurements may drift away from actual values over time. Furthermore, magnetic distortions on the vertical plane, affecting pitch and roll, can be detected and compensated by referencing the Earth's gravity. Magnetic distortions on the horizontal plane however, affecting yaw, can not be compensated.

Two filter gain parameters, $\beta$ and $\zeta$, can be configured in the algorithm. The "filter gain $\beta$ represents all mean zero gyroscope measurement errors (...). The sources of error include: sensor noise, signal aliasing, quantisation errors, calibration errors, sensor misalignment, sensor axis non-orthogonality and frequency response characteristics. The filter gain $\zeta$ represents the rate of convergence to remove gyroscope measurement errors which are not mean zero (...). These errors represent the gyroscope bias." [11] Furthermore, there "is a clear optimal value of $\beta$ for each filter implementation; high enough to minimises errors due to integral drift but sufficiently low enough that unnecessary noise is not introduced (...)" [11]. This means that, to achieve best results, $\beta$ must be chosen for each MARG to compensate their individual mean zero errors, while $\zeta$ must be chosen to compensate for individual bias drift.

As the algorithm relies on the direction of gravity and the Earth's magnetic field, "(...) accelerations due to motion will result in an erroneous observed direction of gravity and so a potentially corrupt the estimated attitude and local magnetic distortions may still corrupt the estimated heading." [11] This means that during motions, any calculated orientation is potentially off and converges to its actual value once the motion stops. Furthermore, the Earth's magnetic field might be distorted by ferromagnetic objects in the vicinity of the user: "Sources of interference in the earth frame, termed soft iron, cause errors in the measured direction of the earth's magnetic field. Declination errors, those in the horizontal plane relative to the earth's surface, cannot be corrected without an additional reference of heading. Inclination errors, those in the vertical plane relative to the earth's surface, may be compensated for as the accelerometer provides an additional measurement of the sensor's attitude." [11] This means that the yaw angle, relying solely on the Earth's magnetic field for adjustments, is expected to show the greatest error due to magnetic distortions and the lack of references.

It is important to note that AHRS are typically used in aircrafts that operate outdoors, where the impact of magnetic distortions due to ferromagnetic objects in the vicinity is limited. This work therefore experimentally evaluates, in chapter 7, the suitability of this AHRS for indoor usage where ferromagnetic objects like metal frames can be found.

## 2.3 Hand Position

Theoretically, the position of the hand can be determined by continuously integrating and summing up linear acceleration provided by an accelerometer twice. However, the same error sources as described in section 2.2.1 for angular velocity integration apply here. Worse, integrating twice accumulates errors over time even faster.

Technologies like GPS (Global Positioning System) aid in tracking positions by using satellites as reference points. But GPS is impractical since a resolution of centimetres can not be achieved for non-military usage. Additionally, the update rate is in the single or double digit Hz range, resulting in a high latency for the user.

This work therefore uses the arm's orientation to determine the hand's position relative to the shoulder by placing additional MARGs on the forearm and upper arm.

### 2.3.1 Arm Orientation

The orientations of the upper and forearm, $\overrightarrow{o_u}$ and $\overrightarrow{o_l}$ respectively, corresponding to equation 2.1, can be determined using an AHRS in the same way like the hand's.

$$\overrightarrow{o} = \begin{pmatrix} roll \\ pitch \\ yaw \end{pmatrix} \tag{2.1}$$

The directional vector of an orientation $\overrightarrow{o}$ can be calculated by function $d$ as shown in equation 2.2. It is assumed that roll, pitch and yaw angles are zero if the hand is parallel to the Earth's surface, palm facing down and fingers facing to magnetic north. Since rolling the arm part around itself does not change the arm part's directional vector, the roll angle is not required for calculation.

$$d(\overrightarrow{o}) = n\left(\begin{pmatrix} cos(yaw) \times cos(pitch) \\ sin(yaw) \times cos(pitch) \\ sin(pitch) \end{pmatrix}\right) = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \; with \; n(\overrightarrow{v}) = \frac{\overrightarrow{v}}{||\overrightarrow{v}||} \tag{2.2}$$

Together with their lengths $l_u$ and $l_l$, the position $\overrightarrow{p}$ of the hand can be determined as the sum of arm part vectors according to equation 2.3.

$$\overrightarrow{p} = l_u \times d(\overrightarrow{o_u}) + l_l \times d(\overrightarrow{o_l}) \tag{2.3}$$

Placing sensors on upper and forearm allows the user to track hand position in a sphere relative to and centred to his shoulder. This provides the user with the freedom to move around independently from a reference system.

### 2.3.2 Error

The accuracy of the hand's position depends on the accuracy of each arm part's orientation. Errors in the arm parts' orientation angles, as described in section 2.2.4, lead to errors in the hand's position $\overrightarrow{p_e}$. The latter can be calculated as the difference between the error-free hand position and one influenced by errors of $n$ degrees. See equation 2.4.

$$\overrightarrow{p_e} = l_u \times \left( d(\overrightarrow{o_u}) - d\left( \begin{pmatrix} roll_u + n \\ pitch_u + n \\ yaw_u + n \end{pmatrix} \right) \right) + l_l \times \left( d(\overrightarrow{o_l}) - d\left( \begin{pmatrix} roll_l + n \\ pitch_l + n \\ yaw_l + n \end{pmatrix} \right) \right) \tag{2.4}$$

Assume an upper arm length of 30cm and a forearm length of 25cm. Errors in orientation provided in section 2.2.4 have been used. Equation 2.5 shows static and dynamic positional errors, $\overrightarrow{p_{e,10Hz,s2}}$ repectively $\overrightarrow{p_{e,10Hz,d7}}$, at 10Hz sampling rate. The static error is 2° and the dynamic error is 7° per axis. Equation 2.6 shows static and dynamic positional errors combined as $\overrightarrow{p_{e,100Hz}}$ at 100Hz sampling rate. The error is 1° per axis.

$$\overrightarrow{p_{e,10Hz,s2}} = \begin{pmatrix} -0.07cm \\ 1.92cm \\ 1.92cm \end{pmatrix}, \quad \overrightarrow{p_{e,10Hz,d7}} = \begin{pmatrix} -0.82cm \\ 6.65cm \\ 6.70cm \end{pmatrix} \tag{2.5}$$

$$\overrightarrow{p_{e,100Hz,1}} = \begin{pmatrix} -0.02cm \\ 0.96cm \\ 0.96cm \end{pmatrix} \tag{2.6}$$

It can be seen that for sub-cm accuracy, a sampling rate of at least 100Hz must be used. Experiments in chapter 7 show whether or not this level of accuracy can be reached.

## 2.4 Acceleration & Velocity

The gyroscope and accelerometer in a MARG already deliver angular velocity respectively linear acceleration. Linear velocity $v_l$ is the positional difference between $\overrightarrow{p_1}$ and $\overrightarrow{p_2}$ per elapsed time $\Delta t$. See equation 2.7. Angular acceleration $a_a$ is calculated similarly as the difference between angular velocities $\overrightarrow{v_{a1}}$ and $\overrightarrow{v_{a2}}$ per elapsed time $\Delta t$. See equation 2.8. Note that both $v_l$ and $a_a$ are also affected by orientation errors.

$$\overrightarrow{v_l} = (\overrightarrow{p_2} - \overrightarrow{p_1}) \times (\Delta t)^{-1} \tag{2.7}$$

$$\overrightarrow{a_a} = (\overrightarrow{v_{a2}} - \overrightarrow{v_{a1}}) \times (\Delta t)^{-1} \tag{2.8}$$

# Chapter 3

# Wireless Technologies

The design goals state that operation is desired to be wireless. Wireless technology candidates that can be found in embedded systems and computers are WiFi, Bluetooth, Bluetooth Low Energy and ZigBee. The author disregards ANT, as it is only proprietary. The wireless technology has to be chosen depending on the motion sensors' data transmission bandwidth and latency, as well as general power consumption requirements.

This chapter presents requirements for wireless technologies. The candidates are evaluated and the stack of the one used in this work, Bluetooth, is described in more detail.

## 3.1 Bandwidth & Latency

Typically, compact MEMS MARGs with an area of up to $1cm^2$ that are inexpensive produce measurement values with up to 16 bits resolution. Assume 16 bits resolution for each of the three axes for each of gyroscope, accelerometer and magnetometer, as well as the temperature sensor. A complete readout would be 16 bits x (3 x 3 + 1) = 160 bits = 20 bytes in size. By using two additional MARGs on the arm, a complete readout for all MARGs would be 3 x 20 bytes = 60 bytes in size. Additionally, using a flex sensor to determine the fingers' flexs increases the total data size. Assuming 1 byte per finger, a total of 65 bytes results.

Algorithms like Madgwick's AHRS utilize floating point operations. These heavily impact latency and power consumption on a low-power embedded system, since they are not optimized for such workloads and often lack floating point units (FPU). Therefore, the 65 data bytes have to be sent to the computer application for processing. At the minimum sampling rate of 100Hz (see 2.3.2), this results in a required transmission bandwidth of 100Hz x 65 bytes = 6.35kB/s. Yet, higher sampling rates are beneficial to decrease errors in orientation and reduce the readout delay. The latter decreases the possibility of the user recognizing a delay, especially for rapid changes in motion. By evaluating a 500Hz sampling rate with a packet period of only 2ms, required transmission bandwidth increases to 31.74kB/s.

This work therefore assumes an approximate maximum required transmission bandwidth of 32kB/s or 256kb/s per glove.

As pointed out previously, a low readout delay hides delays for rapid motion changes. Additionally, a low latency between MARGs' readout and availability of hand position and orientation also contributes to hiding delays. A high bandwidth decreases the transmission time per bit. For example, using a wireless technology with 500kb/s bandwidth results in a latency of about 1.04ms for transmitting 65 bytes, excluding protocol overhead. This bandwidth latency however does not include i.e. processing time.

A lower single digit ms latency is desirable for bandwidth latency. This could potentially allow the whole path from readout to data availability to be below 10ms in latency, which is hardly noticeable by the user. Thus, for two gloves, the achievable bandwidth of the wireless technology must at least be in the middle to upper hundreds of kb/s.

## 3.2 ZigBee

ZigBee, as defined in the standard IEEE 802.15.4, operates with a bandwidth of 250kb/s. While this is barely sufficient to transmit the data of a single glove, two gloves can only be used at low sampling rates. Since this work is about to evaluate latency and motion accuracy, higher sampling rates are required. Therefore, ZigBee is no candidate.

## 3.3 WiFi

WiFi is a wireless technology defined in multiple IEEE 802.11 standards over the years. Today, the author assumes that most private households and workplaces are equipped with routers using WiFi to connect devices with each other and to the internet. The achievable gross bandwidth is 54Mb/s for devices supporting standards IEEE 802.11a/g or later. Even older devices supporting IEEE 802.11b can achieve 11Mb/s. This satisfies bandwidth requirements.

Since sensible information is regularly exchanged in WiFi networks, they are typically secured by Advanced Encryption Standard (AES) encryption. Hence, the glove's embedded system needs to handle AES encryption and decryption if it wants to participate in a WiFi network. This increases latency and especially power consumption, as the algorithms are computationally expensive in software or require extra hardware modules.

Furthermore, the range of WiFi is intended to be e.g. a complete house. To achieve this, hardware amplifiers require additional energy to amplify the signal. Such a signal could penetrate even multiple walls. However, in the glove's use cases imagined by the author, range is already sufficient if it covers a single room. Wall penetration is not required. While the output power can be adjusted, it is typically higher than for e.g. Bluetooth and thus requires more energy [12].

For continuous transmissions at maximum bandwidth, WiFi consumes more power than Bluetooth [12]. One could argue that by using high bandwidths, the transmission time of small, fixed size data is shorter so that, in comparison, power could be saved. It is true that WiFi at high bandwidths is about factor two to three more energy efficient than Bluetooth [13]. However, at low bandwidths like 64kb/s = 8kB/s, the inverse is true. WiFi is actually two to three times less energy efficient than Bluetooth, even without using AES encryption [12].

The author expects that for a bandwidth of about 64kB/s for two gloves, Bluetooth still has the advantage. Therefore, WiFi is no candidate and Bluetooth is to be evaluated.

## 3.4 Bluetooth

Bluetooth is a wireless technology initially standardized as IEEE 802.15.1. Today, it is maintained by the Bluetooth Special Interest Group. The intended transmission range is "personal area", typically limited to e.g. a room. Different versions have been developed over the years. The most common version is Bluetooth 2.1 + EDR, published in 2007. It supports net bandwidths up to 2.1Mb/s, satisfying bandwidth requirements. In 2010, Bluetooth 4.0 was approved and devices have been made available over the last years. It supports net bandwidths up to 0.7Mb/s, also satisfying bandwidth requirements. Version 4.0 aims at reducing power consumption and is marketed as "Bluetooth Smart" or "Bluetooth Low Energy". In this work, the latter is used and Bluetooth 2.1 + EDR is referred to as "Bluetooth Classic". While Classic as well as Low Energy satisfy bandwidth requirements, the differences in transmission behaviour impact latency.

### 3.4.1 Bluetooth Classic Transmission

Bluetooth divides its frequency band into 79 channels. On each one, only a single transmission is allowed to occur at a time. To avoid collisions, frequency hopping is utilized. Communication partners switch ("hop") channels after sending or receiving data, minimizing the impact of external noise sources, like microwaves, as communication quickly takes place on a different, more likely noise-free channel. Each partners use different sequences to avoid collisions. Additionally, a forward error correction (FEC) can add redundancy, potentially allowing data to be repaired without resending them.

Communication takes place between a master and one or more slaves, as depicted in figure 3.1. For this work, the master is the PC running the application and a glove is a slave. "Transmission/reception takes place in timeslots that are only 625 microseconds in duration. The Master uses even-numbered slots to address each slave in turn, and each addressed slave has the opportunity to answer in the following odd-numbered timeslot. Or it can wait for it turn next time around. In addition to this, some timeslots are used for broadcasts and as logical channels for synchronization and other control signals." [14] The latter are shown as "dedicated timeslots" in figure 3.1.

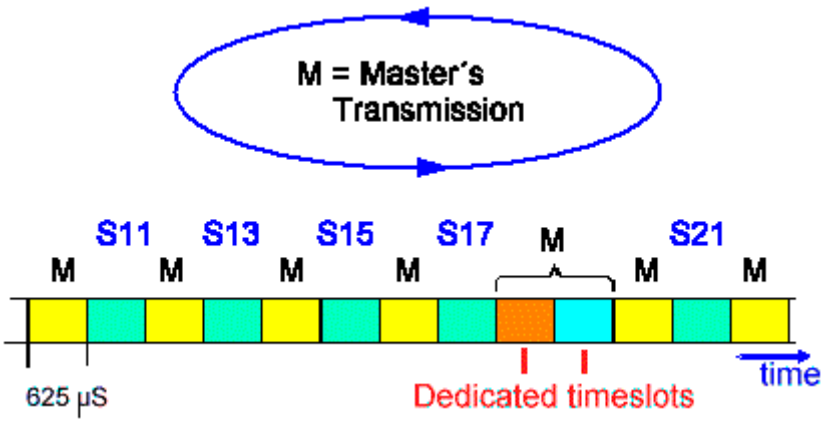


FIGURE 3.1: Bluetooth communication timeslots between master and slaves [14].

Slaves have the opportunity to send their data packets not just in one, but also three or five timeslots. This depends on the size of data available for transmission and FEC usage. Figure 3.2 illustrates this as a table.

| Type | Payload Header (bytes) | User payload (bytes) | FEC |
|---|---|---|---|
| DM1 | 1 | 0-17 | Yes |
| DH1 | 1 | 0-27 | No |
| DM3 | 2 | 0-121 | Yes |
| DH3 | 2 | 0-183 | No |
| DM5 | 2 | 0-224 | Yes |
| DH5 | 2 | 0-339 | No |

FIGURE 3.2: Bluetooth packet sizes [15].

Assume a noise-free environment where each slave glove is always ready to send 65 bytes of sensor data. Packets of type DM3 or DH3 occupying three timeslots can be used. Therefore, the transmission latency per glove is at best four timeslots = 2.5ms. It can be seen that a non-empty transmit buffer, small data sizes and noise-free environments are important to utilize timeslots efficiently for low latency.

### 3.4.2 Bluetooth Low Energy Transmission

The idea behind Bluetooth Low Energy is that many devices don't need to communicate e.g. every two timeslots. For example, it is sufficient for a heart rate monitor to transmit the heart rate every second. But if the master constantly initiates a communication with its slaves, the slaves waste energy by answering that they don't have new data.

To solve this, Bluetooth Low Energy introduces connection intervals and slave latency. The slave specifies the connection interval $i$ in milliseconds and the slave latency $l$ as an

integer. The master is allowed to initiate a communication with the slave every $l \times i$ milliseconds. For the rest of the time, the slave is sleeping to save energy. Communication, once started, works the same as for Bluetooth Classic. The issue is that the connection interval is at minimum 7.5ms and the slave latency 1. This results in a maximum sampling rate of 1/7.5ms = 133Hz. As seen in section 2.3.2, the author wants to evaluate errors in the hand's position at higher sampling rates.

It is possible to transmit multiple sensor readouts every connection interval via DM5 or DH5 packets. The receiver could insert a delay between processing each readout to simulate a higher sampling rate. However, this approach increases latency. Furthermore, Bluetooth Low Energy does not come with a serial port implementation to effortlessly exchange data. While Bluetooth Classic provides the Serial Port Profile (SPP) via its stack, a developer must implement such a profile on its own for Bluetooth Low Energy.

Because of the sampling rate issue and SPP, this work is realized with Bluetooth Classic.

### 3.4.3 Bluetooth Sniff Mode

Devices connected using Bluetooth Classic have to regularly exchange link maintenance data to keep the connection alive. This increases the power consumption for the devices, since e.g. the antenna can not be deactivated. A solution is the Sniff mode, "designed to save power by reducing the amount of link maintenance data exchanged" [16].

This works as follows: A minimum and maximum sniff interval has to be defined, where the minimum must be smaller than the maximum. The devices can agree upon an interval in this range during which to stop exchanging link maintenance data. The minimum however must be at least 6 slots, or 3.75ms. Therefore, Sniff mode may not be utilized for a 500Hz sampling rate, but only for 100Hz. An attempt parameter specifies the number of master-to-slave slots the slave will listen to, distributed over the interval. The timeout parameter defines how many master to slave slots the slave will listen to after receiving data during a slot specified by the attempt parameter.

Figure 3.3 depicts an example communication using Sniff mode. Timeslots are yellow if master-to-slave and the slave listens, turquoise if slave-to-master performed by the slave, and white if the slave does not listen or send at all. The sniff interval has been set to 16 by setting e.g. the minimum to 14 and the maximum to 16. By setting attempt to 1, the slave listens to master-to-slave slots only at the first timeslot 0 within the interval and answers at the following timeslot 1. The timeout of 2 appends two additional timeslot pairs to the communication, from timeslot 2 to 5. This allows the device to send multiple data packets before going into a power-saving mode from timeslot 6 to 15. This procedure repeats at each multiple of 16 timeslots.

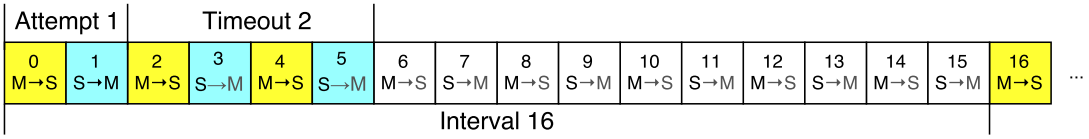


FIGURE 3.3: Timeslots using Sniff mode with attempt = 1, timeout = 2, interval = 16.

### 3.4.4 Bluetooth Protocol Stack

Figure 3.4 illustrates the basic components of a Bluetooth stack. It consists of the physical data transport layer, also called baseband. On top of that are the Host Command Interface layer (HCI), Logical Link Control and Adaptation Protocol layer (L2ACP) and the Service Discovery Protocol service (SDP) [17]:

- *Physical Data Transport Layer.* Handles physical transmissions between the baseband transceivers of participating Bluetooth modules.
- *HCI Layer.* A uniform set of commands implemented by all modules. Basic parameters like output strength can be read and configured. Manages physical connections, also known as physical links.
- *L2CAP Layer.* Manages logical links of upper layers on top of physical links by multiplexing data. Handles packet segmentation and reassembly. L2CAP is required to establish multiple logical links from, for example, different applications on one device by utilizing the single physical link between two devices.
- *SDP Service.* Services, also known as profiles, must be registered at the SDP. Afterwards, other Bluetooth modules can read the profiles provided by the device to determine whether certain profile links can be established.

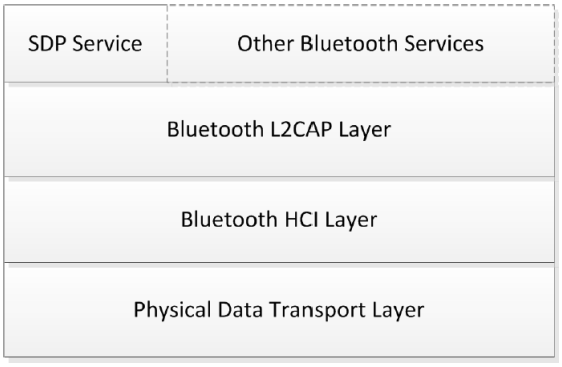


FIGURE 3.4: "Basic components that form a typical Bluetooth Stack" [17].

As stated above, Bluetooth services are commonly referred to as profiles. The standard defines dozens of them for a wide range of use cases. This includes, amongst others, real-time audio transmission, blood pressure monitoring and human interface device profiles

for e.g. wireless keyboards. The relevant profiles for this work are the Generic Access Profile (GAP) and the Serial Port Profile:

- *GAP.* Most importantly, GAP handles the authentication when connecting to a device. This can be done, for example, password-based or by pressing a button on the device during authentication. Devices can be "paired" by storing address information to allow future connection attempts to be allowed if the addresses match. For details on how GAP is used in this work, see section 5.2.
- *SPP.* The glove needs to transmit arbitrarily formatted sensor data at time intervals dependent on the sampling rate being used. No special profile exists for this use case, except for SPP. Data input is a serialized stream of bytes which is received in the same order on the opposite side. The transmission is transparent and requires no control flow management at application level, except for making sure that the desired number of bytes has been received.

Note that while real-time profiles exists, they are purposed for audio and video transmissions and make use of lossy compression algorithms. This would damage sensor values and lead to increased orientation errors, which is not desired. Since a real-time transmission is favourable for audio application control via glove, it is necessary to evaluate SPP to see whether transmission intervals occur regular and periodical.

# Chapter 4

# Hardware Selection & Design

The previous chapters 2 and 3 discussed requirements for motion sensors and Bluetooth modules with respect to the design goals. In this chapter, the process of selecting adequate hardware components is illustrated. This includes the motion sensors and Bluetooth module, as well as a suitable micro-controller to control the hardware and additional necessary integrated circuits (IC). Schematics and layouts are provided to show how the components are integrated. Furthermore, the power circuit running from a LiPo or USB connection is presented. Hardware design files are found in appendix A.

## 4.1 Motion Sensors

As shown in section 2.1, a Spectra Symbol 4.5" Flex Sensor is used to determine finger flexing. To reduce hardware costs, only a single one is used on a single finger. Its flex value however is transmitted and used for every finger to simulate a complete set.

MEMS gyroscopes, accelerometers and magnetometers can be found in single MARG ICs. Many also include a temperature sensor, which is required for temperature drift compensation. Such ICs are preferred to keep the board layout small, since less IC area and wiring is required compared to using three different ICs.

To keep errors in the hand's orientation and position small, each MARG must be able to provide readouts with at least 100Hz, preferably up to 500Hz.

Furthermore, the MARG must be able to measure all of the hand's and arm parts' motions. This means that the gyroscope must have a resolution of at least 2000dps to not miss fast rotations, like twisting the arm. The accelerometer must support at least 16G to not miss rapid changes in motion, like hits into the air.

The addressing of a certain sensor inside the MARG is typically realized by addressing one of the MARG's registers. This implies that it is not possible to read out all three sensors in parallel. Therefore, fast connections to the MARGs are required.

Typical connection candidates are Inter-Integrated Circuit (I2C) with a 400kHz bandwidth and Serial Periphal Interface (SPI) with a bandwidth in the Mhz range. As described in section 3.1, a complete readout is approximately 20 bytes = 160 bits in size. For I2C at 400kHz, this results in a latency of 160/400kHz = 0.4ms just for the data, excluding slave and register addressing. For SPI at e.g. 1MHz, the data latency is 160/1MHz = 0.16ms. This scales linearly with an increased SPI bandwidth. The obvious choice is a MARG capable of SPI with a bandwidth of multiple MHz.

The initial MARG candidate was the InvenSense MPU-9250 [18]. It is supposed to draw a maximum current of 3.5mA. It contains all required sensors with the necessary bandwidths and resolutions, except for the magnetometer, which operators at only 8Hz. However, the package's pads are placed with a distance of 0.4mm which is difficult to solder with the author's equipment.

The STMicroelectronics LSM9DS0 is therefore used for this work [19]. Its package's pads are placed with a distance of 0.5mm. The total package area is $4\times4mm^2$. The maximum bandwidths are 760Hz for the gyroscope, 1600Hz for the accelerometer and 100Hz for the magnetometer. The latter is not optimal for a desired 500Hz sampling rate, but the author did not find inexpensive, compact MARGs with a higher magnetometer bandwidth. The sensors support the required resolutions and can be read with up to 10MHz via SPI. While the magnetometer provides a significantly increased bandwidth compared to the MPU-9250, it is still below 500Hz. The author however was not able to find inexpensive MARGs with higher magnetometer bandwidths. Separate magnetometers are not desired to keep the board size small. Furthermore, current consumption is about at least 6.5mA during operation, almost twice as much as for the MPU-9250. These factors need to be considered when evaluating orientation errors and device runtime.

## 4.2 Bluetooth Module

A wide variety of Bluetooth modules exist. In a previous work, the author utilized a HC-06 module, controlled by an Arduino Nano [1]. The module supported only Bluetooth SPP and the authentication was completely transparent to the Arduino application. Communication took place via Universal Asynchronous Receiver Transmitter (UART) at a maximum baud rate of 115,200. Such modules are quite common.

Their downside however is the limited baud rate. At 115,200bps, transmitting 65 bytes of sensor values via UART to the modules requires 65×8b / 115200bps = 4.5ms, excluding any protocol overhead and additional processing or buffering delays.

Another downside is the transparency of the connection since no HCI commands are supported. No information is available whether a connection has been established or is in progress. Pairing occurs without any notification and without any means of modification. Only a limited set of parameters can be configured via vendor-specific commands.

The author desires a module capable of communicating via HCI commands to build a robust system with full control over the connection. This would serve the design goals by aiding at reducing latency as well as being aware of when to put the microcontroller into a sleep mode to save power.

The Bluetooth module used for this work is the Panasonic PAN1325B [20]. This module with an area of 9.5×9mm$^2$ and integrated antenna supports Bluetooth Classic and Low Energy. Communication takes place via UART with up to 4Mbaud. The transmission latency for 65 bytes therefore is only 65×8b / 4,000,000bps = 0.13ms, excluding any overhead. Furthermore, HCI is fully supported.

The latter however requires a microcontroller capable of handling a Bluetooth software stack to communicate with the PAN1325B. Since implementing a stack using HCI commands would exceed the scope of this work, a suitable implementation for a capable microcontroller has been identified.

## 4.3 Microcontroller

The PAN1325B contains the Texas Instruments (TI) CC2560B wireless controller. TI offers the Bluetopia Bluetooth Stack SDK royalty-free if used with such a wireless controller and one of their MSP430 or TM4C12x microcontrollers [21]. The MSP430 series is designed for low-power applications and devices. For the use of the Bluetooth SDK, one of the recommended models, which is used in this work, is the MSP430F5438A. Its specification and suitability for this work is described as followed [22]:

- 1.8V to 3.6V supply voltage for 8MHz to 25MHz clock frequency. 1.8V is required by the PAN1325B and is the minimum for the LSM9DS0. However, the hardware should be designed with a supply voltage of at least 2.7V to allow the microcontroller to run at the maximum of 25MHz for further evaluations.
- The internal clock can be sourced from external crystals. Furthermore, a total of three different clock signals, generated from clock frequency division or a frequency-locked loop (FLL), can be used internally or for peripheral connections. This is necessary to e.g. run the microcontroller with 25MHz while communicating with the LSM9DS0 at 10MHz.
- 256kB of programmable flash memory. Other models with e.g. 128kB could be problematic since the SPP demo provided by the Bluetooth SDK already requires about 120kB of flash memory. 256kB is sufficient to implement SPP and additional functionality in the glove's firmware.
- 512 bytes of information flash memory. This is a special segment of the 256kB programmable flash memory typically not used for program code. This allows storage of arbitrary values during program execution without affecting the latter.

For example, Bluetooth pairing information can be stored here to be available after a device reset.

- 16kB of RAM. Again, other models with e.g. 8kB RAM are potentially undersized, as the SPP demo requires 6.4kB of RAM and lacks sensor readouts and other functionality. 16kB is sufficient.
- UART with 1Mbaud. Unfortunately, this microcontroller does not fully satisfy the possible UART baud rate to the PAN1325B of 4MBaud. $1 MBaud$ however results in a transmission latency of 0.52ms, excluding any overhead, which is acceptable.
- Up to eight SPI ports are usable. This allows all three LSM9DS0 to be connected to their own SPI port. Since some of the ports support Direct Memory Access (DMA) transfers, and all of them can be executed asynchronously via interrupt service routines (ISR), sensor readouts can be done asynchronously in parallel. This further reduces latency.
- Analog-to-digital converter (ADC) with 12 bits resolution. This provides 4,096 steps to digitalize the voltage running through the flex sensor to determine the finger's flex, which is satisfying. The voltage of the LiPo can also be measured in the millivolt range.
- Multiple execution modes to save power. Depending on which parts of the clock system are deactivated, they save different amounts of power.
- The package PZ100 has an area of $1.4 \times 1.4 cm^2$ with a total of 100 pins.

It can be seen that the TI MSP430F5438A satisfies memory and connectivity requirements. The final power consumption and achieved sensor readout rate however are still subject to evaluation with a finished device.

The MSP430F5438A can be programmed and debugged via the Joint Test Action Group's "Standard Test Access Port and Boundary-Scan Architecture", also known as JTAG. As a fallback, an FTDI FT232R IC can be used to provide a serial port via USB connection [23]. This FT232R can be specifically connected (see 4.4.5) to the microcontroller to allow the programming by using the Bootstrap Loader (BSL) [24]. This is a program pre-programmed into the microcontroller's flash memory that loads data received by the serial port of the FT232R into the program memory. The BSL can be started by sending a special control sequence over the serial port.

For this work, only JTAG was used for programming and debugging. BSL has not been tested since the former works as desired.

## 4.4 Microcontroller Connections Circuit

Figure 4.1 gives an overview of the microcontroller connections, placed on the board on the hand. Highlighted are the major components, which are explained in more detail during the following subsections: *(a)* The LSM9DS0 MARG with *(b)* the port for the flexible flat cable (FFC) connecting to the arm parts' boards' MARGs. *(c)* The PAN1325B Bluetooth module with the level converter required to convert the microcontroller's 2.7V signals to 1.8V and vice-versa. *(d)* Four LEDs. *(e)* The JTAG port. *(f)* The FTDI FT232R for BSL programming with an activation switch. *(g)* Power supply and LiPo. *(h)* Two pins for the Spectra Symbol 4.5" flex sensor. *(i)* Pin and flyback diode for the vibration motor. *(j)* Pin for capacitive touch materials.

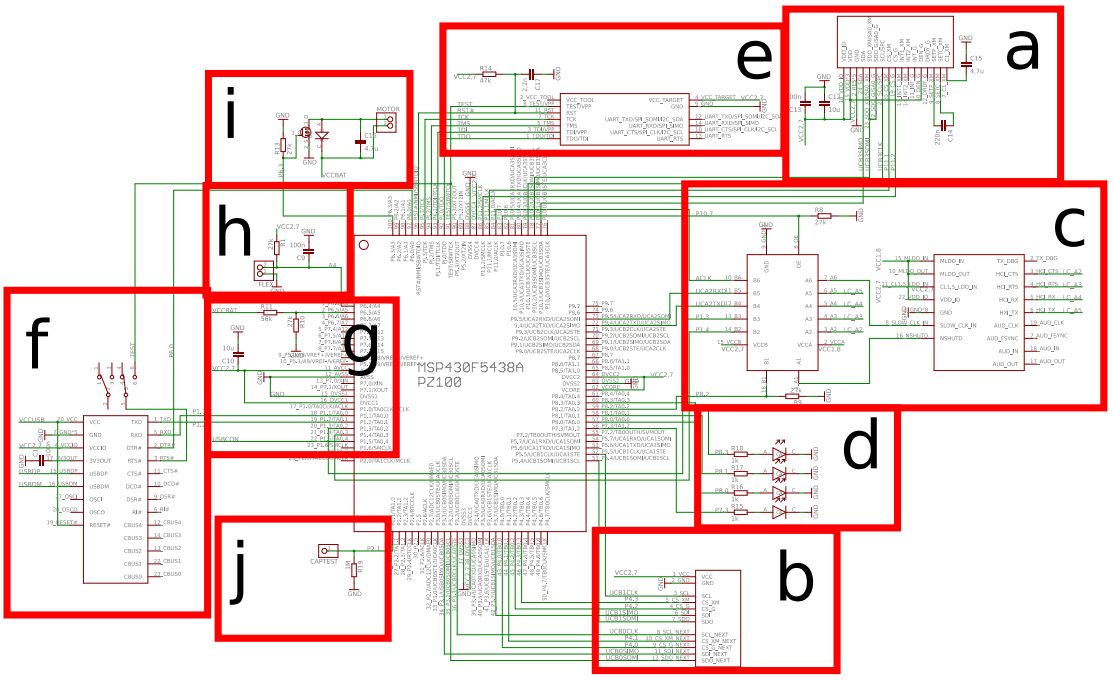


FIGURE 4.1: Microcontroller connections schematic overview

### 4.4.1 LSM9DS0 & FFC

Figure 4.2 depicts the LSM9DS0 connection to the microcontroller. Capacitors have been chosen according to datasheet recommendations [19]. Supply and IO voltage are both 2.7V, just like for the microcontroller.

One must notice that two SPI chip select pins exist: One for the gyroscope (CS_G), and one for the accelerometer and magnetometer (CS_XM). Both these groups also have different output pins, SDO_G and SDO_XM. They are however connected to the same microcontroller pin since both outputs are supposed to never happen at the same time. The MARG's SPI pins are connected to the microcontroller's SPI pins at port UCB3. The chip select pins have to be controlled manually via P11.1 and P11.2.

Since the datasheet provides no further information on the DEN_G pin except for "Function: Gyroscope data enable", the author decided to connect it to the supply voltage to presumably always enable the gyroscope. Interrupt pins and DRDY_G ("Function: Gyroscope data ready") are not used, to reduce wiring and because the readout occurs periodically anyway.

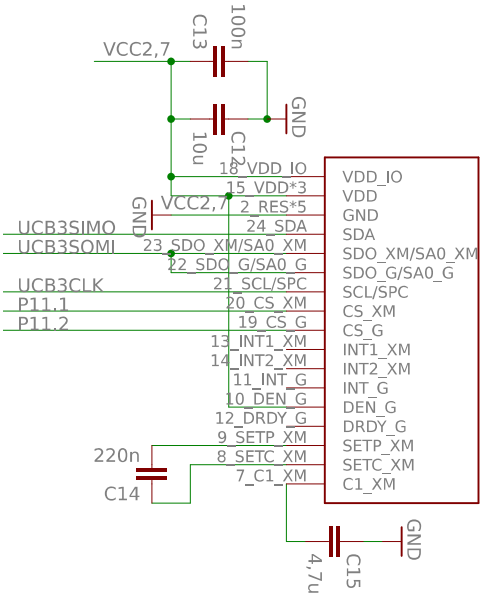


FIGURE 4.2: LSM9DS0 connections to the microcontroller (rotated)

Figure 4.3 shows the FFC port's connections. The FFC port is intended to connect the forearm's and upper arm's MARG boards to the main board on the hand. Therefore, two sets of five wires each are used to connect the MARGs' SPI ports to the microcontroller's. The forearm MARG is connected to SPI port UCB1 with chip select controlled via pins P4.3 and P4.2. For the upper arm, it's UCB0 with P4.1 and P4.0. 2.7V supply voltage and ground are provided, too. The FFC therefore requires a total of twelve wires.

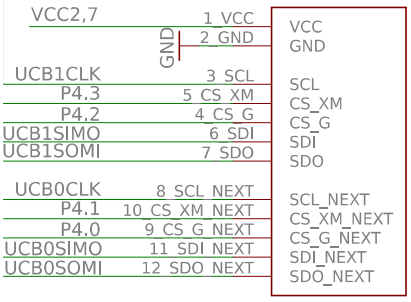


FIGURE 4.3: FFC connections to the microcontroller

### 4.4.2 PAN1325B & Level Converter

The PAN1325B Bluetooth module only operates with an IO voltage of 1.8V [20]. Since the microcontroller runs at 2.7V, a level converter is required.

For this purpose, the TI TXB0106 has been chosen [25]. It supports the required voltages and has a latch-up performance exceeds of 100mA, which is sufficient to run the PAN1325B at maximum performance (approximately 40mA during transmit plus IO). Switching times are low enough to allow 1Mbaud UART communication.

Furthermore, as depicted in figure 4.4, the six channels A1/B1 to A6/B6 provided by the TXB0106 are satisfying. The PAN1325B requires six connections to the microcontroller: Four for the UART, including hardware control signals clear-to-send (CTS) and ready-to-send (RTS), a NSHUTD connection to signal module shutdown, as well as SLOW_CLK_IN. The latter is an input expecting a clock signal with 32,678Hz and an accuracy of 250ppm [20].

The UART transmit and receive pins are connected to the microcontroller's UART at port UCA2. CTS and RTS are connect to the interrupt-enabled pins P1.3 and P1.4. Handling them has to be done in software. NSHUTD can be controlled manually via pin P8.2. SLOW_CLK_IN is connected to ACLK, which is the auxiliary clock output generated by the microcontroller. These connections have been chosen according to a developer's guide for pin-compatible ICs [26]. Additionally, the TXB0106 needs to be enabled via pin 10.7. Pull-down resistors R8 and R9 are used to disable the level converter at startup and disable shutdown for the Bluetooth module after enabling the level converter. 27kΩ are sufficient to reduce 2.7V at any possible current to a level equivalent to logical 0.

The PAN1325B's power supply pins CL1.5_LDO_IN, MLDO_IN and MLDO_OUT can be connected in three different configurations [20]. Each one impacts the "RF output power" and "system power" differently. By connecting CL1.5_LDO_IN to 2.7V and MLDO_IN and MLDO_OUT to 1.8V, the module is supposed to have "maximum RF output power and optimum system power". This means that the wireless signal strength can be increased to cover greater distances, and the module itself draws less current compared to other configurations. This is the optimal configuration for this work to evaluate wireless range and power consumption.

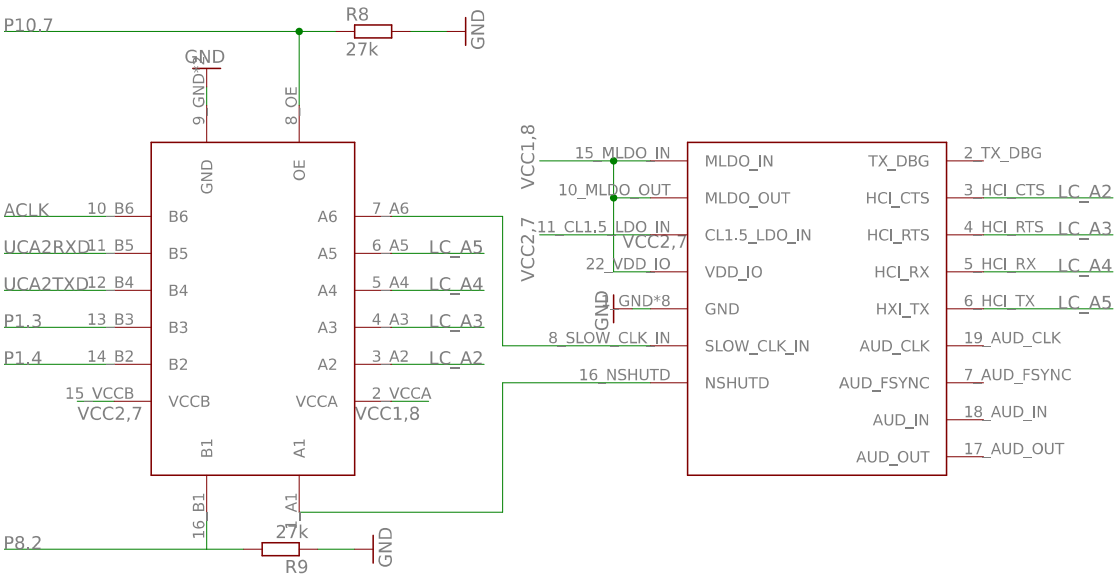


FIGURE 4.4: PAN1325B (right) connections to TXB0106 (left) and to microcontroller

### 4.4.3 LEDs

The board on the hand contains four LEDs for feedback purposes. For example, blinking LEDs could indicate a working device with software making progress.

Figure 4.5 shows the schematic. The LEDs are connected to pins P7.3, P8.0, P8.1 and P8.3 and can be controlled manually in software. The chosen model is the Osram LG L29K-F2J1-24 [27]. It generates a bright green light at 2mA current consumption. The necessary current limiting resistors R15 to R18 are chosen based on the maximum forward voltage of 2.2V: R = (2.7V - 2.2V)/2mA = 1kΩ.

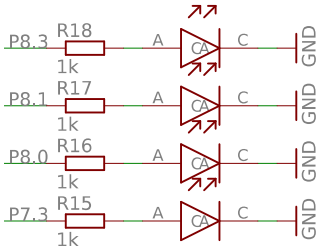


FIGURE 4.5: LED connections to microcontroller

### 4.4.4 JTAG

The chosen MSP430F5438A microcontroller can be programmed and debugged via JTAG. Debugging this way is advantageous as e.g. all memory locations can be viewed and breakpoints can be set. Connecting a device via JTAG port to a PC's USB port

requires special programmer hardware. The Olimex MSP430-JTAG-TINY-V2 programmer has been chosen for this work. It is fully compatible to TI devices.

The programmer features a 14-pin JTAG connector [28]. Figure 4.6 illustrates the connection to the microcontroller. 4-wire JTAG is used, connecting the pins TCK, TMS, TDI and TDO to their counterparts. The available TEST pin is connected this way, too. Additional wiring, resistor and capacitor for the reset pin RST have been chosen according to user guides [28, 29]. By leaving VCC_TOOL unconnected, the programmer draws its power from the USB port. VCC_TARGET is connected to the 2.7V supply voltage and can sense the voltage to identify whether the device is powered or not.

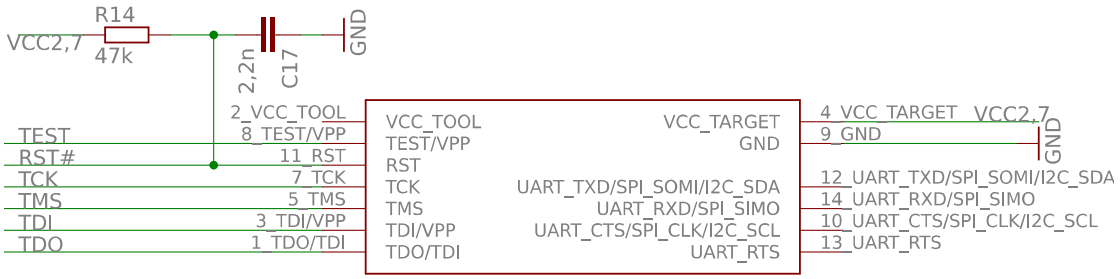


FIGURE 4.6: JTAG connections to microcontroller

### 4.4.5 FTDI FT232R

As described in section 4.3, the MSP430F5438A can be programmed via BSL. To start the BSL, the TEST pin must be pulled up twice before resetting the device [24]. Afterwards, P1.1 becomes the BSL UART's transmit and P1.2 the receive pin. P6.0 becomes the BSL UART's data terminal ready (DTR#) and TEST the RTS# pin. Programming is done by sending the serialized program via UART to the BSL.

To program the microcontroller via USB, the FTDI FT232R IC has been chosen. It directly connects to the USB port's USBDP and USBDM pins. Software drivers ensure that the IC connected to the PC is usable as a regular serial port. The software also allows to manually change the RTS pin required to start the BSL by pulling TEST up twice [30]. Afterwards, RTS# can be used as a UART pin again.

For all this to work, the switch connecting RTS# and TEST must be closed. This is a safety measure, since the JTAG port is also connected to TEST. However, because programming and debugging via JTAG worked flawlessly during this work, programming by using the FT232R and BSL approach has never been tested.

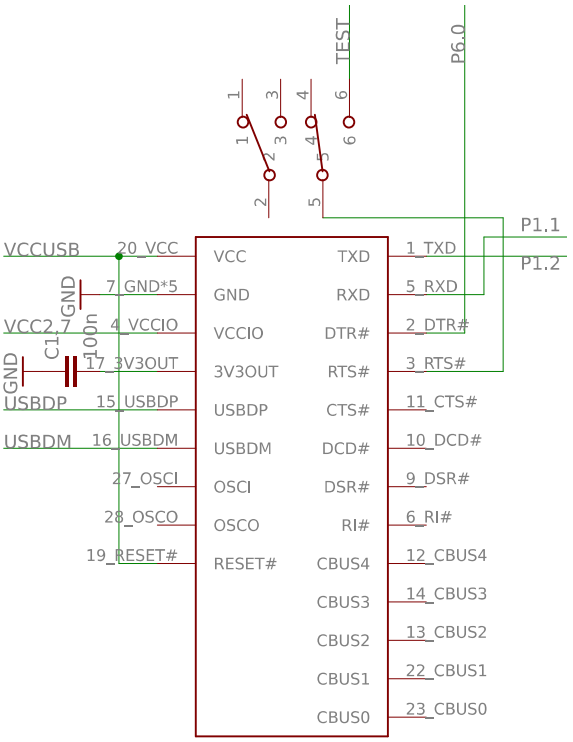


FIGURE 4.7: FTDI FT232R and switch connections to microcontroller

### 4.4.6 Power Supply & LiPo

Figure 4.8 shows how the power supply, LiPo and USB information are connected to the microcontroller.

The analog reference voltages, AVCC for power and AVSS for ground, are connected to the supply voltage of 2.7V respectively ground. The capacitor C10 decouples dynamic current to reduce noise when using the ADC. Digital power and ground, DVCC1 and DVSS1, are also connected to 2.7V and ground. The microcontroller has a total of four DVCC and DVSS pins each, which are not shown here. They are connected identically.

Additionally, USBCON provides the logical signal to P1.5 whether a USB cable has been plugged in or not. Furthermore, VCCBAT, the LiPo voltage, is connect to the ADC pin A5. This, together with USBCON, allows the software to determine the voltage and whether the LiPo is discharging, charging or full.

As described in section 4.5, the power supply runs from a 3.7V LiPo. The voltage of a LiPo in use however is not constant, but drops over time. A fully charged one typically has a voltage of 4.2V, which can not be measured by the ADC running with a reference of 2.7V. Therefore, a voltage divider consisting of resistors R10 and R11 has been implemented. At 4.2V, the ADC's input at pin A5 would be $4.2\text{V} \times 27\text{k}\Omega / (27\text{k}\Omega + 56\text{k}\Omega) = 1.37\text{V}$. The software can now quantify all of the LiPo's voltage levels.

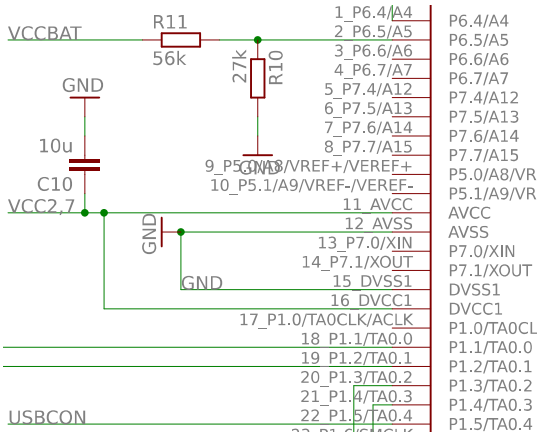


FIGURE 4.8: Power supply, LiPo and USB connections

### 4.4.7 Flex Sensor

The flex sensor can be connected at two pins on the board. The flat resistance of about 10kΩ increases to up to 110kΩ when being bent [31].

Figure 4.9 depicts the implemented voltage divider. For a flat flex sensor, the ADC's input at pin A4 equals 2.7V×27kΩ/(27kΩ + 10kΩ) = 1.97V. When bending the sensor by flexing the finger, the measured voltage drops to up to 2.7V×27kΩ/(27kΩ + 110kΩ) = 0.53V. These limits can be used to determine how much the sensor has been bent. The capacitor C9 removes noise due to unsteady resistance changes from the flex sensor.

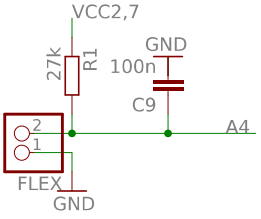


FIGURE 4.9: Flex sensor connection and voltage divider

### 4.4.8 Vibration Motor

The schematic is designed to provide pins to connect a vibration motor. Figure 4.10 depicts the schematic. The following description explains how the vibration motor is supposed to be controlled and how the schematic is supposed to work according to the author's understanding of the latter. On the finished PCB however, the vibration motor is always driven, independently of whether the control pin has been pulled down or not. The author did not find the fault.

The motor is controlled via pulse-width modulation (PWM) at pin P6.3, which is pulled down after device reset. The pin controls the n-channel MOSFET transistor

IRLML6402PbF by International Rectifier [32]. If the pin output is set to high, the transistor connects VCCBAT to GND. Otherwise, the connection to GND is closed. The intensity of the vibration is therefore dependent on how fast the pin switches between low and high to control the power received by the motor.

The capacitor C15 smooths the PWM's voltage to reduce variations in the vibration's intensity during pin switching. Additionally, a flyback diode is placed in parallel to the motor. Since the motor acts as an inductor, Lenz's law states that after removing the input voltage by closing the transistor, current flows in the voltage's opposite direction. This current could damage the transistor. The flyback diode prevents this by routing the current to itself [33].

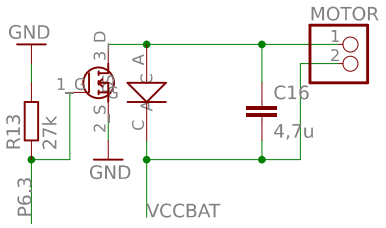


FIGURE 4.10: Vibration motor connection to microcontroller

### 4.4.9 Capacitive Touch

Figure 4.11 depicts the schematic for a single-touch capacitive sensor design [34]. The capacitive sensor, like a button or fabric, increases its capacity depending on e.g. how much of its area is covered by a finger. This is useful for features like a slider build from multiple buttons or like the detection of touching fingers via capacitive fabric.

The microcontroller software was intended to use pin P2.1 to charge the capacitor, i.e. a finger, and measure the time it takes for it to discharge. The amount of time correlates to the style of touch. The resistor R19 with 1MΩ has been chosen to allow a quick discharge for fast measurements.

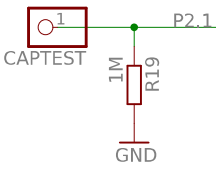


FIGURE 4.11: Capacitive touch connection to microcontroller

## 4.5 Power Supply Circuit

This section discusses the power supply circuit. The goal is to power the system via a LiPo and allow the latter to recharge using USB. USB data pins must be forwarded to the FTDI FT232R and the microcontroller should be able to read the LiPo voltage and charging state.

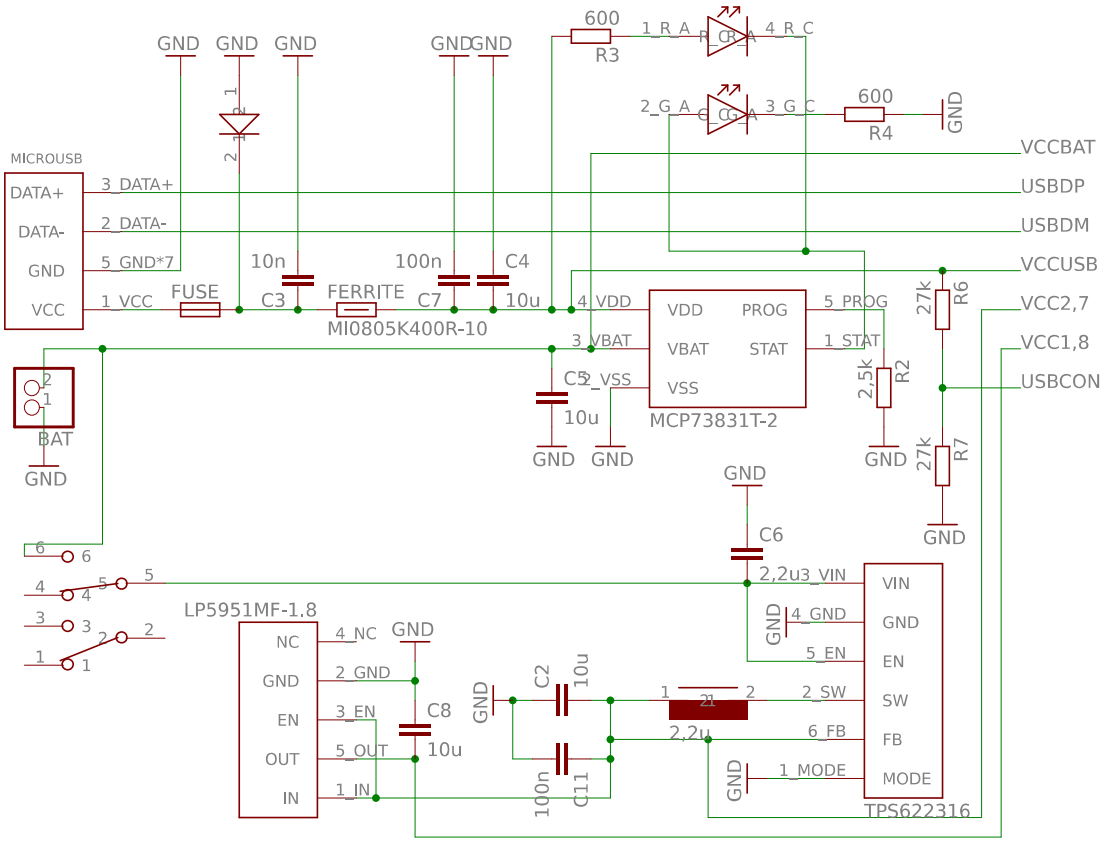


FIGURE 4.12: Power supply and USB connection

Figure 4.12 shows the power supply circuit. On the left, the Micro USB port and BAT LiPo input pins can be seen, as well as the power switch. On the right, the output signals to be used by system are found. This includes voltage signals VCCBAT for the LiPo, VCCUSB for USB, VCC2,7 for the 2.7V supply, as well as VCC1,8 for the 1.8V supply for the PAN1325B. USBCON signals whether a USB cable is plugged in. USBDP and USBDM are the positive and negative USB data lines used by the FT232R.

VCC of the Micro USB port is secured by a fuse and a suppression diode. This protects the device from damage caused by current spikes due to e.g. malfunctioning connected USB ports or electrostatic discharges. The fuse breaks at 500mA or more, which is the maximum that USB supports normally [35]. The ferrite bead suppresses electromagnetic wave disturbances caused by high-frequency signals. This is recommended, as USB operators in the MHz range. The capacitors C3, C4 and C7 smooth the voltage signal, which is then output as VCCUSB, 5V if USB is connected.

The LiPo charging IC is the Microchip MCP73831T-2 for LiPos with a maximum voltage capacity of 4.2V [36]. Its VDD input is connected to the Micro USB port's VCC. The IC supports a charge current of up to 500mA, the maximum USB can provide. However, that actual charge current depends on the resistor connected between PROG and ground. For this work, the resistor R2 is used with 2.5kΩ. This allows the IC to charge the battery with up to 400mA, which still leaves up to 100mA to be used by the device. This is a safety measure to ensure that the device always has enough current to operate.

The VBAT output of the MCP73831T-2 is regulated to charge the LiPo. VCCBAT is the LiPo voltage forwarded to the microcontroller independently of whether a USB cable is connected or not. Furthermore, this signal is the input to the power switch and therefore where the system draws its current from. Once the device is switched on, the VBAT signal serves as input to the TI TPS622316 switching regulator [37]. It generates a 2.7V DC output at 3MHz switching frequency. The MODE pin is pulled low to enable the "Power Save Mode with automatic transition from PFM (Pulse frequency mode) to PWM (pulse width modulation) mode". EN is connected to VBAT to always enable to IC once an input is available. Other pins, capacitors and the inductor are connected and chosen according to the datasheet. In general, a switching regulator wastes less power than a low-dropout regulator (LDO) and was therefore chosen for this work. The chosen IC works with up to 94% efficiency. The output serves as the 2.7V supply voltage signal VCC2,7 for the system.

The PAN1325B Bluetooth module however requires 1.8V IO signals. To ensure proper device functionality, an LDO has been chosen, like for a TI evaluation board with pin-compatible Bluetooth module and microcontroller [38]. The TI LP5951MF-1.8 serves as the LDO IC [39]. By connecting EN to IN, it is always enabled once the switch regulator provides output. GND and OUT are wired and connected via capacitor according to datasheet recommendations. The LDO's output is the 1.8V supply voltage signal VCC1,8. The author however assumes that using an LDO for the 1.8V supply was not the best choice. Such ICs reduce the voltage by dissipating power into heat, effectively wasting energy and negatively affecting total power consumption. Another linear switching regulator should have been used.

Lastly, a dual LED in red and green shows the power state. If a USB cable is connected and the LiPo is charging, it emits red light. If the LiPo is fully charged, it emits green light.

## 4.6 Hand PCB Layout

Figure 4.13 depicts the hand PCB layout with a size of $5.4 \times 5 cm^2$ All components are placed on the PCB's top layer (red). The area below the Bluetooth module's antenna, placed on the top border of the PCB to minimize distortion from other components, is not filled with copper, as this would interfer with the wireless transmission.

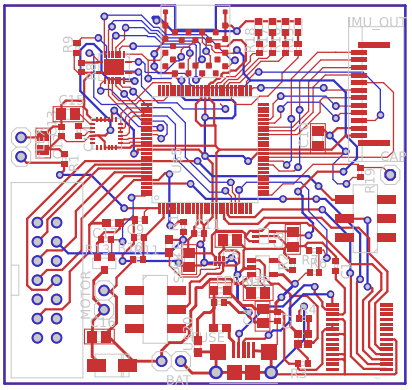

FIGURE 4.13: Hand PCB layout, sized 1:1.

Figure 4.14 shows the finished hand PCB. The FFC cable port is placed at the top right border, the JTAG 14-pin port at the bottom left border and the USB port at the bottom border. This allows easy cable access. Note that because the internal 32,768Hz crystal's accuracy is too low to drive the Bluetooth module, an additional suitable crystal has been glued on top of the microcontroller. It is connected to the external crystal input pins XT1IN and XT1OUT with two recommended 18pF capacitors [40]. The following components have been soldered by Dr. Andreas Wurz: Microcontroller, Bluetooth module and level converter, MARG, linear and switching voltage regulator, MOSFET, FTDI IC, USB port, as well as the external crystal.

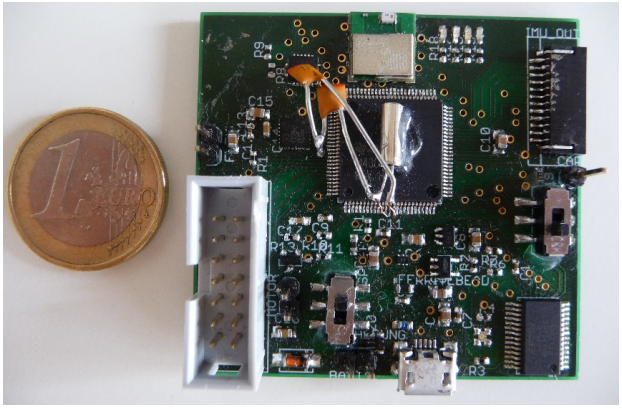

FIGURE 4.14: Hand PCB.

## 4.7 Arm PCBs

The forearm and upper arm must be provided with a MARG to allow the determination of the hand's position relative to the shoulder. This section presents the arm boards.

Figure 4.15 depicts the arm boards' schematic. Each board contains two FFC ports. The left one serves as the input for the board. The signals connected to the microcontroller

SDO, SDI, CS_G, CS_XM and SCL are used by the MARG. The other signals, denoted by the _NEXT suffix, are forwarded to the right FFC port to serve the next board as input. This was done to have a single board design for both the forearm and upper arm. The forearm board receives both input signals from the hand board and forwards the _NEXT signals to the upper arm board. Both boards make use of VCC and GND.

The capacitors and wiring of the MARGs are analog to the one on the hand board. Design recommendations from the datasheet are used [19]. The resistor R1 with 0Ω was intended to be replaced by a resistor with a greater resistance to measure the voltage drop. This could have been used as an indicator on how much power the MARG consumes, but has never been utilized in this work.

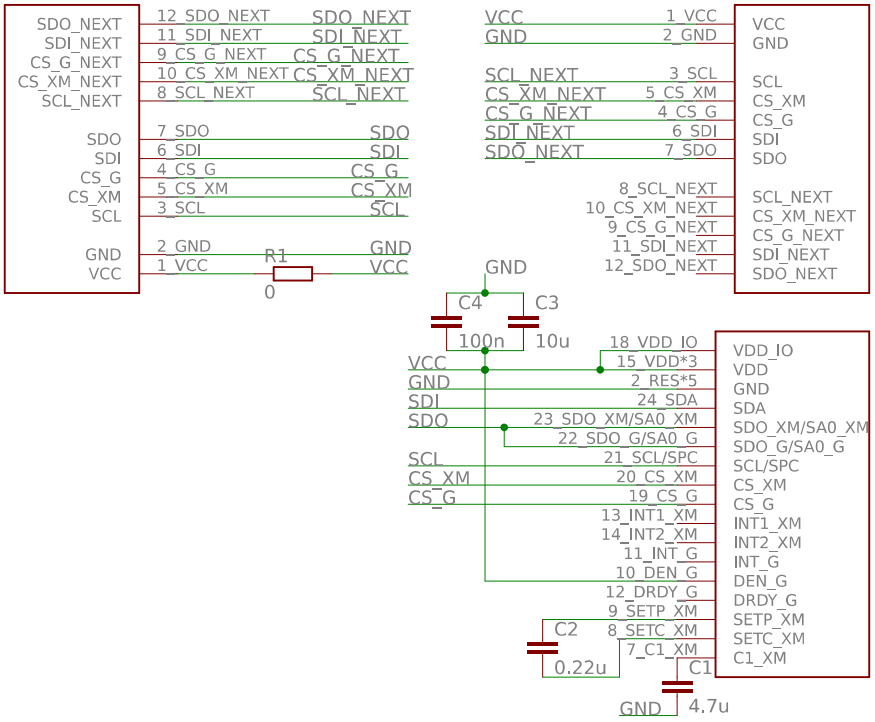


FIGURE 4.15: Arm board schematic with LSM9DS0 and FFC ports.

Figure 4.16 shows the routed arm board layout. The size is 3cm × 2cm. Noticeable is the area required to forward signals to the upper arm without intersecting signals on the board itself. All components are placed on the top layer (red).

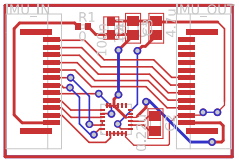

FIGURE 4.16: Arm board layout, sized 1:1.

Analog to the board layout of figure 4.16, figure 4.17 depicts the two finished arm boards. The forearm board contains two soldered FFC ports, one for its input and one to forward signals to the upper arm board. The upper arm board contains only the one for its input. The MARGs have been soldered by Dr. Andreas Wurz.

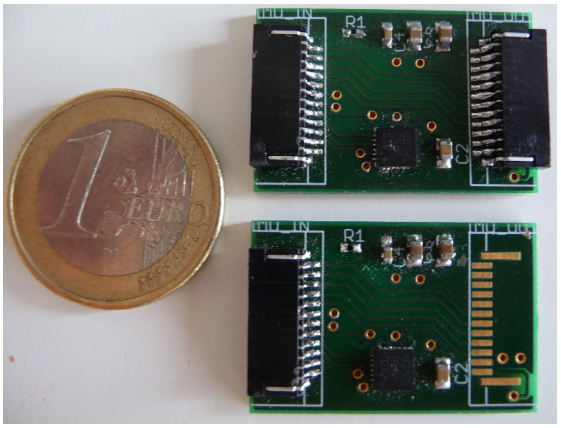


FIGURE 4.17: Forearm board (top) and upper arm board (bottom).

## 4.8 Assembled Glove

Figure 4.18 displays the complete glove with connected hand and arm parts' PCBs, as well as the glove and arm wrap fabrics. While the glove fabric itself has been bought, the pockets and arm wrap have been hand-sewn to fit the components. The glove is powered by a 180mAh LiPo. For a complete bill of materials, see appendix B.

FIGURE 4.18: Fully connected glove PCBs in glove and arm wrap fabric.

# Chapter 5

# Software Architecture

This chapter provides an overview of the important software components. Figure 5.1 depicts the architecture of the glove firmware with its two major components: The command loop, and readout and processing. The command loop reads and writes commands from and to a buffer that stores data from the UART port used to communicate with the Bluetooth module. Commands sent from the PC application include, amongst others, starting or stopping the sensor readout. LEDs are controlled to indicate state. The data readout and processing component utilizes three SPI ports to read out the three MARGs on the hand, forearm and upper arm. Furthermore, the bend of the flex sensor attached to a finger, as well as the LiPo's charge, is read via the ADC. A connected Micro USB port is determined, too. If sensor readout has been enabled by, this data is written to the command buffer for Bluetooth transmission to the PC application.

Figure 5.2 depicts the PC application software. Like the glove firmware, a command loop handles communication and execution of functionality via command exchange. Once sensor readout data is received, calculations take place to update the hand's orientation and position. Sensor values like finger flex and LiPo charge are converted to usable units, too. A visualization is then updated to show the arm and hand. Furthermore, the updated hand data is translated to a fixed set of MIDI signals, which are sent to the virtual MIDI port. DAW applications can then utilize these signals as controls.

The glove firmware has been written in C++03 using TI Code Composer Studio v6.1.0 with the TI compiler v4.4.4. The MSP Driver Library of MSPWare v2.21.00.39 and the CC256x MSP430 Bluetopia SDK v1.5 R2 have been utilized [21, 41]. The PC application has been written on Microsoft Windows 8.1 Pro in C++11 using Visual Studio Express 2013 for Windows Desktop v12.0 Update 4. Bluetooth communication utilizes sockets and the implementation is derived from a previous project. [42]. The visualization has been implemented using Irrlicht v1.8.1 and virtual MIDI ports are controlled by utilizing Tobias Erichsen's virtualMIDI SDK [43, 44]. MIDI-related code has been taken from a previous project, too, but extended to work with the values provided by this glove [42]. Source code is available in appendix A.

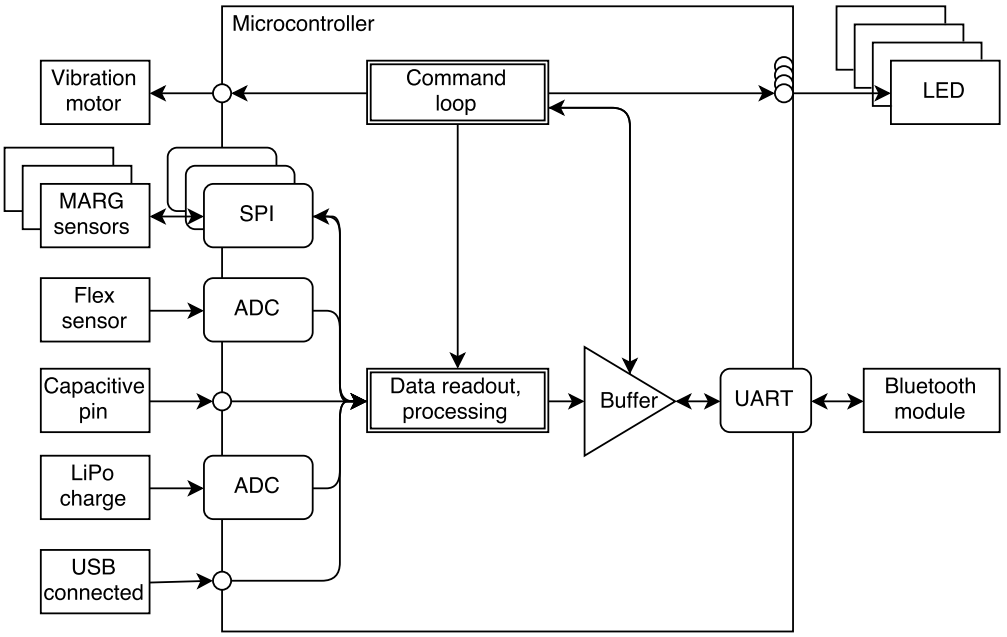


FIGURE 5.1: Overview of glove firmware.

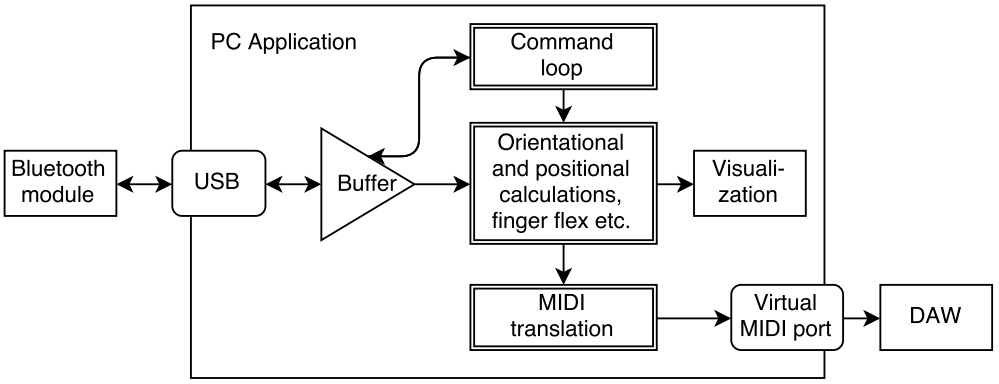


FIGURE 5.2: Overview of PC application.

## 5.1 Glove Initialization

Every time the glove is powered on, the microcontroller's clocks are set up. Since the internal crystal has a low accuracy, the external one with 32,768Hz and less than 250ppm accuracy is used to source the master clock. This clock drives program execution. To reach 25MHz, the voltage core level must be increased to the maximum, requiring at least the 2.7V provided from the power supply. Afterwards, the FLL can be used to settle the master clock at 25MHz. The subsystem clock drives peripheral interfaces like SPI. For this work, it is sourced from the master clock with a divider of four. Therefore, the subsystem clock frequency is 25MHz / 4 = 6.25MHz. As this is significantly below

the 10MHz supported by the MARG's SPI, a master clock of 20MHz with a subsystem clock divider of two to reach 10MHz is evaluated in terms of performance in chapter 7. The auxiliary clock is driven out at the ACLK pin to provide the Bluetooth module PAN1325B with the necessary 32,768Hz frequency.

Afterwards, the ADC is set up to work with its internal 5MHz clock and use a sampling period of four cycles. This configuration has been proven to generate reliable measurements. Third, the MARGs are configured to operate with the desired readout bandwidths and resolutions. Their SPI ports are set up to work with the necessary clock phase, polarity and endianness. Fourth, the timer used by the scheduler is initialized. The timer is sourced from the auxiliary clock and increments a tick variable each millisecond. This is the minimum usable period to schedule functions. The purpose is avoiding e.g. busy waiting while loops. By using the scheduler, a function is called periodically and can put the microcontroller into a power saving mode after execution. See 5.3 for the command loop as an example.

Lastly, the Bluetooth stack is initialized. This includes setting up the data structures and buffers, as well as the HCI driver handling UART communication between microcontroller and PAN1325B. L2CAP is configured to let the Bluetooth module operate as a slave device, since the PC application initiates the communication.

## 5.2 Bluetooth SPP Connection

Using an initialized Bluetooth stack, figure 5.3 depicts the sequence necessary to establish a serial connection between the glove and the PC application. First, the SPP service has to be registered at the SDP and opened. Afterwards, the GAP makes sure the module is pairable, connectable and discoverable. The user is now able to use his OS to discover and pair with the device. If the glove is already connected, pairing is disallowed. If not, pairing information is updated by writing the PC's Bluetooth device address and a generated link key into the microcontroller's flash memory, as explained in section 5.9. The OS can now discover the SPP service and install the necessary serial port drivers.

After this preparation, the application can attempt to establish an SPP serial port connection to the glove via socket. The socket makes the OS Bluetooth driver try to connect to the glove using the link key from the pairing. If the link key and address still match, the connection is established and the socket becomes connected to SPP. Calibration data of the MARGs is sent automatically first. See chapter 6 for more information on sensor calibration or section 5.9 regarding retrieving the values from the flash memory. Afterwards, serial data like commands can be exchanged.

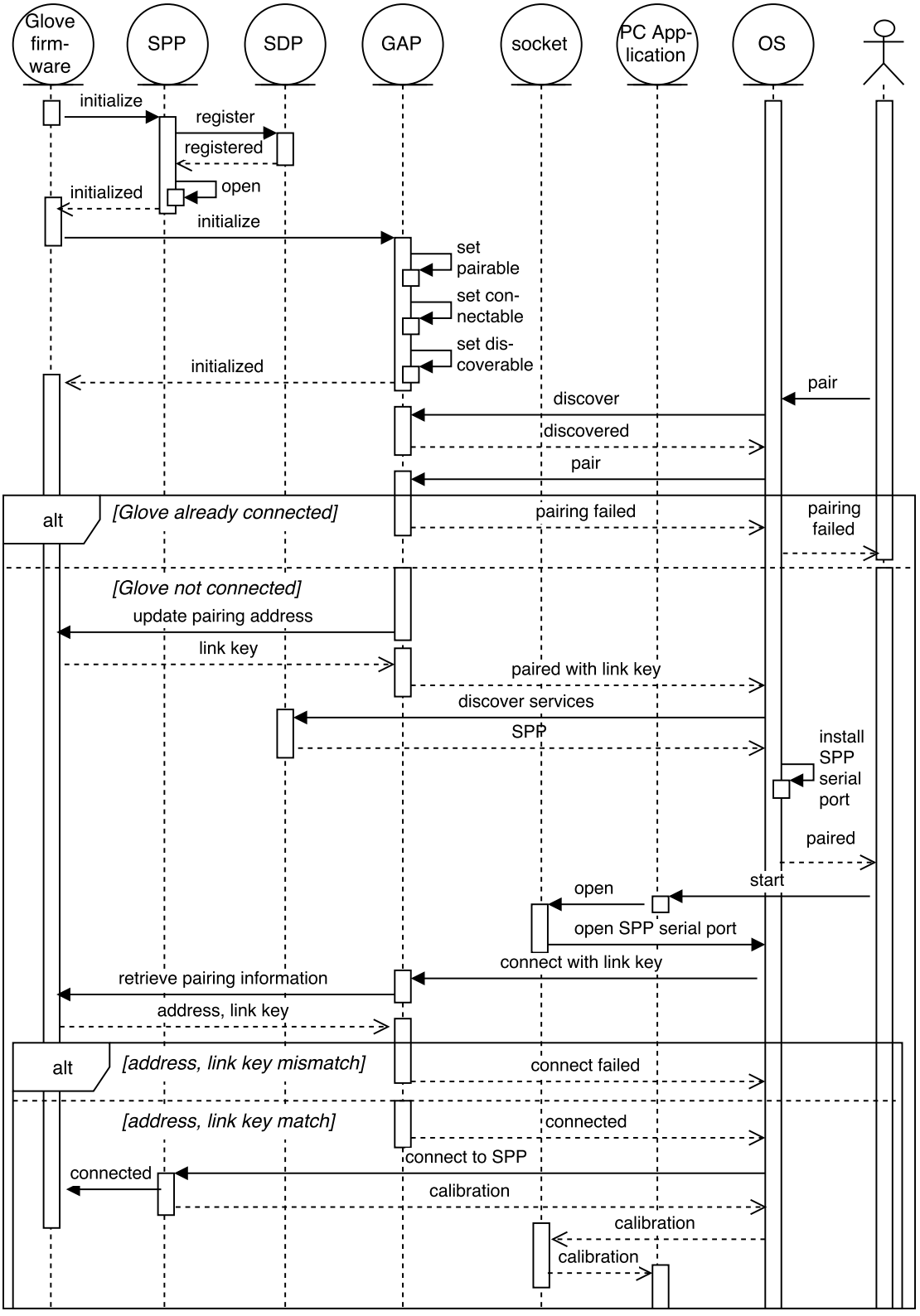


FIGURE 5.3: Sequence to establish a Bluetooth SPP connection.

## 5.3 Command Loop & Commands

The glove and PC application each implement a command loop to exchange and execute commands with data. Listing 5.1 shows the glove's command loop. The loop function is added to the scheduler to be called each millisecond. It executes commands received via SPP. Afterwards, the serial port sends and receives the Bluetooth module's data.

```
1  void loop(void* parameters)
2  {
3      if (context.serial_port->isOpen()) {
4          Command::received(context)->execute(context);
5          context.serial_port->progress();
6      }
7      LPM0;
8  }
9
10 void main()
11 {
12     Scheduler::instance().add(loop, NULL, 1);
13     Scheduler::instance().run();
14 }
```

Listing 5.1: Glove command loop.

Listing 5.2 shows the PC application's command loop: a simple infinite loop. The application sleeps during the call to waitUntilAvailable if no data is present, until the Bluetooth driver asynchronously provides data received from the glove. Special commands to e.g. start the sensor readout are executed before the main loop.

```
1  for (;;) {
2      serial_port->waitUntilAvailable(Command::prefix_size);
3      Command::received(context)->execute(context);
4  }
```

Listing 5.2: PC applicationg command loop.

The context struct, shown in listing 5.3, is used by both platforms to share command code. On the glove, commands are provided with the serial port to use for data transfers. This is analog for the PC application, where an additional glove variable is also provided. By creating multiple contexts, e.g. two gloves can be used by PC application. Both platforms' Contexts come with the readout_diff_ref variable. It contains the reference readout values for differential compression, as described in section 5.5.

```
1  struct Context {
2  #ifdef MSP430
3          ISerialPort* serial_port;
4  #else
5          std::shared_ptr<SerialPort> serial_port;
6          std::shared_ptr<Glove> glove;
7  #endif
8          SharedPointer<Readout> readout_diff_ref;
9  };
```

Listing 5.3: Context.

The execute method of the ReadoutCommand is shown in listing 5.4. On the glove's microcontroller, execute starts MARG readouts and waits using power saving mode LPM0 until they are finished. The command can then be sent to the PC application. Serialization ensures that all the captured readout data is available there, too, to be used for updating e.g. the hand position and orientation when executing the command.

Note that if the Bluetooth transmit buffer has not enough space available to fit the serialized data of a readout into it, the data is discarded. This lowers the achieved sampling rate if e.g. 500Hz is desired, but the device only manages to transmit at 300Hz. The PC application integrates values over the delay between received readouts.

```
1  void ReadoutCommand::execute(Context& context)
2  {
3  #ifdef MSP430
4      MARG::hand().startReadingOut(readout().margs[0]);
5      MARG::forearm().startReadingOut(readout().margs[1]);
6      MARG::upperArm().startReadingOut(readout().margs[2]);
7      while (!MARG::hand().isDoneReadingOut()
8      ||      !MARG::forearm().isDoneReadingOut()
9      ||      !MARG::upperArm().isDoneReadingOut()) {
10         LPM0;
11     }
12     readout().capture();
13 #else
14     context.glove->update(readout);
15 #endif
16 }
```

Listing 5.4: Execution of ReadoutCommand on both platforms.

## 5.4 Sensor Readout

Sending StartReadoutCommand to the glove starts sensor readouts, as shown in listing 5.5. readout and temperature are added to the function scheduler. readout is called every 2ms for a 500Hz rate, whereas temperature is called each second, as temperature changes less frequent. Both function then execute their corresponding commands, like the ReadoutCommand for sensor readout, and send them to the PC application.

```
1  void StartReadoutCommand::readout(void* parameters)
2  {
3          Context& context = *((Context*) parameters);
4          ReadoutCommand command;
5          command.execute(context);
6          command.send(context);
7  }
8
9  void StartReadoutCommand::temperature(void* parameters)
10 {
11         Context& context = *((Context*) parameters);
12         TemperatureCommand command;
13         command.execute(context);
14         command.send(context);
15 }
16
```

```
17  void StartReadoutCommand::execute(Context& context)
18  {
19  #ifdef MSP430
20      if (!started()) {
21          Scheduler::instance().add(readout, &context, 2);
22          Scheduler::instance().add(temperature, &context, 1000);
23          started() = true;
24      }
25  #endif
26  }
```

LISTING 5.5: Execution of StartReadoutCommand on the glove.

As seen in listing 5.4, MARGs are read out by calling their startReadingOut method. This starts the asynchronous readout using ISRs as depicted in the activity diagram of figure 5.4. The goal is to read out all sensor bytes with minimal overhead.

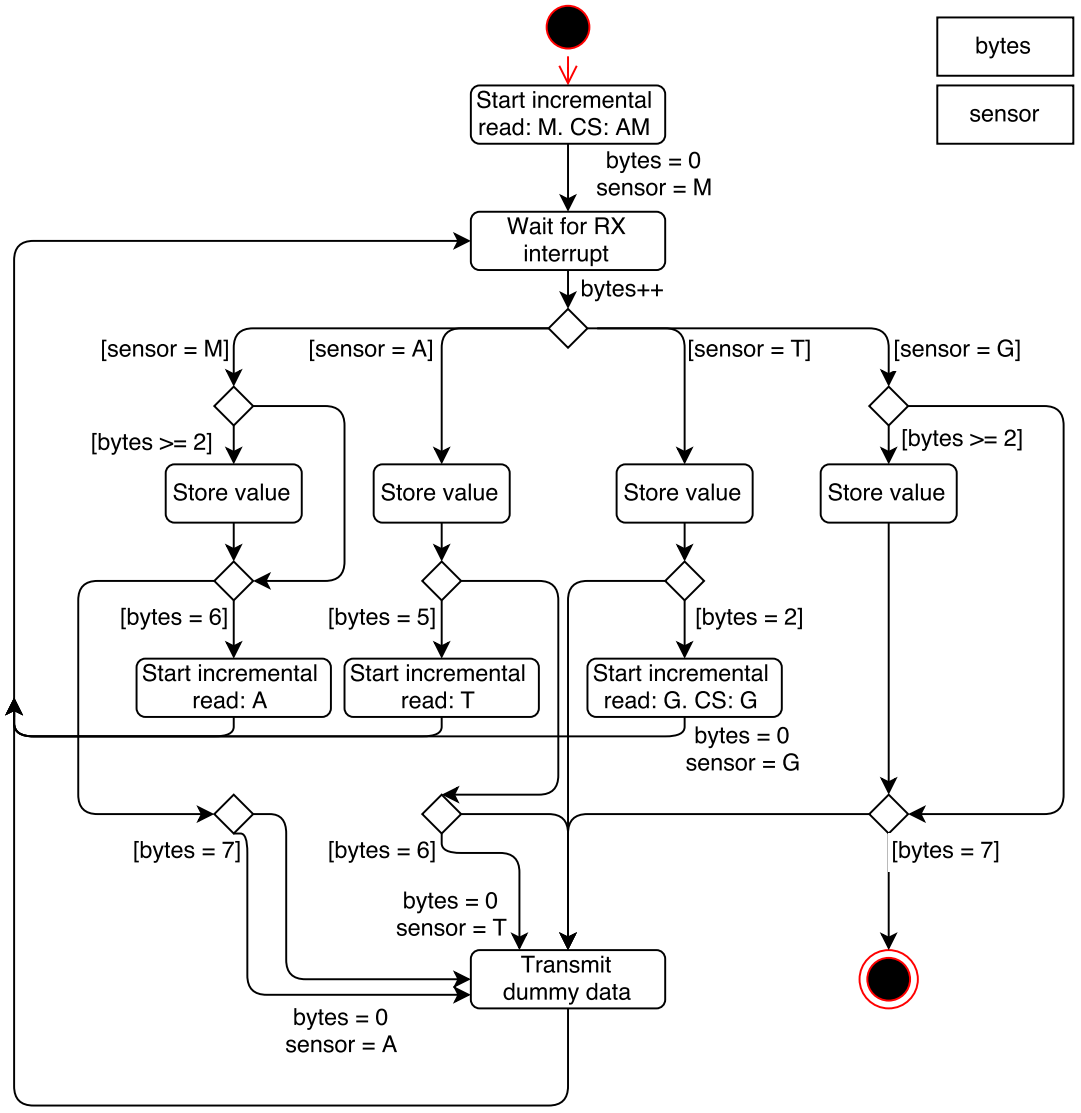


FIGURE 5.4: MARG readout activity.

The magnetometer, accelerometer and gyroscope provide six bytes, while the temperature sensor provides two. By transmitting their data register addresses to the MARG, a readout with automatically incremented register addresses is started. Every time a byte is received, the ISR is called. However, a null byte, has to be transmitted to receive the next byte from the MARG after transmitting a register address.

First, the magnetometer is read out, which requires activating the necessary chip select. The first received byte is discarded and not stored since SPI operates in full duplex mode and this byte is therefore not the first value byte. The second to seventh received bytes are stored.

The full duplex mode is the reason reading out the accelerometer is started after receiving the sixth byte, which corresponds to the second to last value byte. By receiving the last value byte, the accelerometer register address reached the MARG and the next output will by accelerometer's first value byte. No byte has to be discarded. This scheme continues until the temperature sensor has been read out.

To read out the gyroscope, the previous chip select has to be deactivated and the one for the gyroscope must be activated. Because of this, the readout it not started after receiving the first temperature byte. This means that the first received byte has to be discarded due to the full duplex mode operation. The following six bytes are then the gyroscope value bytes. After receiving them all, the readout is finished.

The described implementation reads received bytes from the SPI receive register and stores them into RAM during regular program execution. But the microcontroller supports DMA for received SPI data. However, this can only be utilized for the forearm and upper arm MARG boards, as during the hardware design, the author was not aware that not all SPI ports are supported by DMA.

The difference can be seen by comparing activity diagram figure 5.4 to 5.5. The latter requires no storing of values, as the DMA controller takes care of this. Only the DMA destination address has to be updated accordingly. Initially, the author planned to only set this to the first byte of the data buffer for each sensor once the sensor to read out changes. The automatic DMA destination address incrementation was supposed to write the data bytes to their corresponding buffer indices. Despite using the DriverLib functions and verifying via debugger that the necessary registers are set correctly, the automatic address incrementation did not work. Therefore, the address has to be incremented during program execution after each received byte. Furthermore, the DMA controller treats byte addresses with a different endianness. This makes it necessary to change the endianness for each 16 bit sensor value word after reading out all values. The effect on readout latency and power consumption is determined experimentally in chapter 7.

The readout sequence can be described as XMMMMMMAAAAAATTXGGGGGG, where X is a discarded byte and M, A, T and G are sensor value bytes. This means 22 bytes have to be transmitted to receive all 20 sensor value bytes from a MARG.

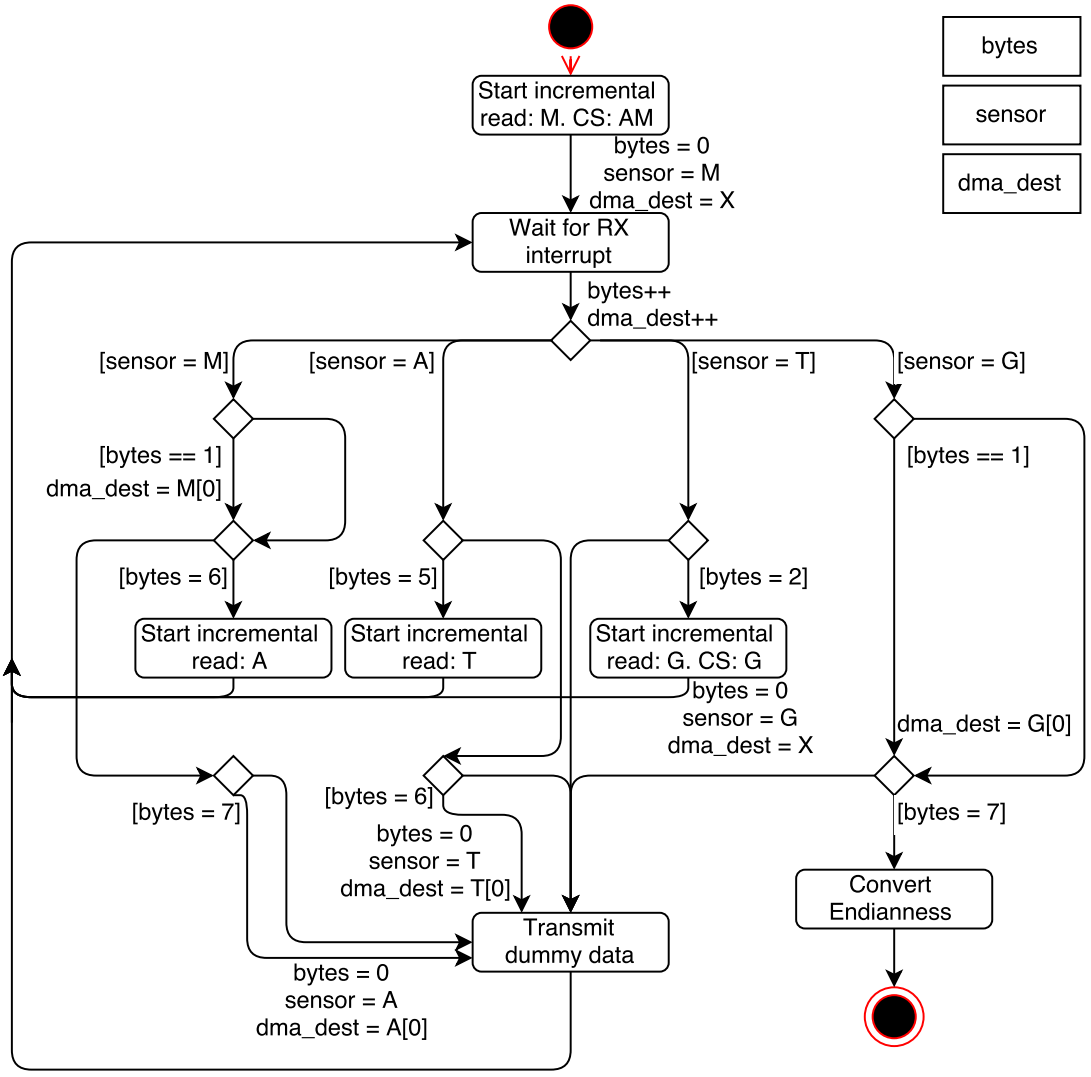


FIGURE 5.5: MARG readout activity.

## 5.5 Readout Compression

According to the datasheets, the Bluetooth module, when transmitting, consumes the most power of all components. In an attempt to reduce power consumption by reducing transmission bandwidth, a data compression technique has been developed.

The key idea is that for a sampling rate of 100Hz or 500Hz, resulting in a readout period of 10ms respectively 2ms, sensor values are expected to not change much between two consecutive readouts. For a 16 or 12 bit sensor value, most of the time, only the lowest bits are expected to change. This is especially true for continuous motions without rapid changes. Consequently, a differential compression algorithm has been developed for this work. It can be applied to gyroscope, accelerometer, magnetometer and flex sensor readouts.

| | Flag | Index | Data | Bit size |
|---|---|---|---|---|
| Compressed, minimum | 1 | 000 | X | 5 |
| Compressed, maximum | 1 | 111 | XXXX XXXX | 12 |
| Uncompressed | 0 | | XXXX XXXX XXXX [XXXX] | 13 [17] |

TABLE 5.1: Readout compression encoding.

Regarding the encoding as illustrated in table 5.1, first, a single bit, the compression flag, declares whether the following bits represent compressed or uncompressed values.

If a value is compressed, the flag is followed by three index bits encoding the values zero to seven. Starting from the least significant bit (LSB), they indicate the highest bit that has changed since the previous readout. Up to eight changed data bits are appended. Therefore, the total size for compressed readout values ranges from five to twelve bits. This results in a size decrease of 25% to 68.75% for 16 bit values, and up to 58.3% for 12 bit values. In case the value did not change, a zero bit is still appended. Otherwise, if nothing is supposed to be sent, additional bits would have been necessary to declare which sensor the bits belong to. For three three-axis gyroscopes, accelerometers and magnetometers, as well as five flex sensors, this would require five bits to provide a mapping to the $3{\times}3{\times}3 + 5 = 32$ values. Such an approach would require the same minimum number of bits, but increase the maximum number and processing time.

If the index of the highest changed bit is above seven, the value is uncompressed and the flag is followed by all 16 or twelve data bits. The total size is increased by one bit, the flag. The resulting size increase is 6.25% for 16 bit values and 8.3% for 12 bit values.

For example, the previous flex sensor value was 0000 1010 0101, and the current one is 0000 1010 1111. The highest changed bit is at index 3, counting from the LSB at in index 0. The resulting compressed data would be 1 011 1111, saving 4 bits or 1/3 of the data size.

Encoded compression bits of all sensor values are directly concatenated. Bits defining the compression of one sensor value may be distributed in multiple bytes during serialization. This eliminates empty bits and reduces the total bit size of a complete readout to the minimum. Processing time however increases due to necessary bit masking operations.

## 5.6 Glove Representation

The previous sections described how sensors are read out and how commands with data are exchanged and executed. As shown in figure 5.2 and listing 5.4, the glove data for the PC application is updated with every received sensor readout. Processed data is represented using the Glove class as shown in listing 5.6. Access to hand, forearm and upper arm data, as well as the battery is provided. The presented data values can be translated into MIDI signals.

```
1  class Glove {
2  public:
3      Hand hand() const;
4      ArmPart forearm() const;
5      ArmPart upperArm() const;
6      Battery battery() const;
7  };
```

Listing 5.6: Glove class representing glove data.

Listing 5.7 presents the classes Hand, Finger and ArmPart. Each Glove has one Hand object with information about translatory and rotatory motions, as well as fingers. The flex of a Finger object can be read and is defined as the percentage of a stretched respectively closed finger. The forearm and upper arm ArmPart objects provide rotatory motions, as well as their three-dimensional vector derived from rotation and length. The latter is used to determine the hand's position.

```
1  class Hand {
2  public:
3      Motion translation() const;
4      Motion rotation() const;
5      Finger thumb() const;
6      Finger indexFinger() const;
7      Finger middleFinger() const;
8      Finger ringFinger() const;
9      Finger littleFinger() const;
10 };
11
12 class Finger {
13 public:
14     double flex() const; // [%] (0% = stretched, 100% = closed)
15 };
16
17 class ArmPart {
18 public:
19     Motion rotation() const;
20     double length() const; // [cm]
21     Vector<double> toVector() const;
22 };
```

Listing 5.7: Hand, Finger and ArmPart classes.

Hand and ArmPart contain motion information in form of a Motion object. Listing 5.8 shows that each of such an object provides three-dimensional values for position, speed and acceleration. Depending on the usage, those value are referring to translatory or rotatory motion.

```
1  class Motion {
2  public:
3      Vector<double> position() const;     // [cm]     or [deg]
4      Vector<double> speed() const;        // [cm/s]   or [deg/s]
5      Vector<double> acceleration() const; // [cm/s^2] or [deg/s^2]
6  };
```

Listing 5.8: Motion class.

Furthermore, each Glove holds a Battery object as defined in listing 5.9. Battery provides information about the charge and usage state, i.e. whether the USB cable is plugged in and the LiPo is charging or not. The charge voltage and percentage can be read.

```
1  class Battery {
2  public:
3      enum class ChargeState {
4          Unknown,
5          Discharging,
6          Charging,
7          Full
8      };
9      enum class UsageState {
10         Unknown,
11         InUse,
12         PluggedIn
13     };
14     ChargeState chargeState() const;
15     UsageState usageState() const;
16     double charge() const;     // [V]
17     double percentage() const; // [%]
18 };
```

Listing 5.9: Battery class.

## 5.7 Alignment Correction

As described in section 2.2.4, the provided orientation in the form of the Tait-???Bryan angles roll, pitch and yaw is relative to a reference frame. For example, the hand's yaw angle equals 0° if the hand is pointing to the Earth's magnetic North. Since this is impractical for the user to work with, his alignment to the reference frame must be corrected so that his hand and arm parts' orientations are zero once the application is started. Therefore, for the first 10 seconds of the application, the user has to stretch out his arm, which must be horizontally to the Earth's surface. This time is required to let the AHRS algorithm make the orientations converge to their values relative to the reference frame. Roll, pitch and yaw of hand and arm parts will converge to some angles. These angles are then used as offsets to correct the user's alignment to the reference frame, making all angles refer to the stretched-out arm position.

## 5.8 MIDI Translation

The MIDI protocol was designed for musical instruments and describes multiple types of messages that can be exchanged in form of MIDI signals [46]. The messages used in this work are Note On and Note Off to start and stop playing notes. Each one is characterized by the key and its velocity, which is the audio volume the key should be played at. Additionally, Controller Change is used to set the value of a specified

controller number. These are used by the DAW software to e.g. control the magnitude of effect manipulations applied to a note.

Listing 5.10 presents an example. The virtual MIDI port named "Sound Glove" is opened. DAW software can retrieve MIDI signals from it. For the note to be played, the first parameter, the key, is bound to the hand's vertical position. The key range is 57 to and higher, corresponding to the notes A3 and higher. The second parameter binds the velocity to finger flexing. A closed hand leads to silence, whereas an open hand produces a loud sound. DAW software can map these played notes to arbitrary instruments. Obviously, for a complete musical project, the DAW software must be prepared and correspond to the programmed notes.

```
1 MIDIPort port("Sound Glove");
2 MIDINote note(
3     [&]() { return 57 + glove->hand().translation().position().z() / 100; },
4     [&]() { return 127 - glove->hand().indexFinger().flex() * 1.27; }
5 );
6 port.play(note);
7 // Process readout
8 port.play(note);
9 // ...
```

LISTING 5.10: Sound Glove virtual MIDI port and playing a note with key bound to the hand's vertical position and velocity bound to finger flexing.

## 5.9 Flash Memory Usage

The MSP430F5438A microcontroller contains four information memory segments A, B, C and D. Each is 128 bytes in size and can be read and written during program execution to retrieve and store arbitrary data. This data then survives power cycles.

Table 5.2 shows the content of segment A. The Magic string is used to determine whether the information flash memory has been initialize. If "DEADBEEF" can not be found here, all items in all segments are set to their default values. Magic is then set to "DEADBEEF". Furthermore, segment A contains the Bluetooth device name which defaults to "Sound Glove". During the Bluetooth pairing process, as described in section 5.2, the remote address and generated link key is also stored and retrieved from here.

Table 5.3 shows how calibration values of the MARGs are stored in different information memory segments B, C and D. The values are too large to fit those of e.g. two MARGs into a single segment. These values are transmitted to the PC application after the SPP connection is established, as seen in section 5.2. Chapter 6 provides more information in sensor calibration.

Note that the calibration value indicating an inverted MARG placement, like for the forearm and upper arm where the x and y axes are inverted compared to the hand, is logically treated as a boolean value. Calibration information in the flash memory however is treated as a packed struct to simplify serialization and save flash memory

by removing any byte pads. By default, structs on an MSP430 are word aligned to 16 bit addresses, just like 16 bit integers. If a single byte boolean value would have been the first byte of the packed struct, memory accesses to e.g. the following 16 bit integer values would have been off by 8 bits due to missing byte pads, resulting in garbage reads. Furthermore, float is used for scaling values instead of double since using the latter would have exceeded the available amount of flash memory.

| Item | Type | Size |
|---|---|---|
| Magic | char[8] | 8 |
| Bluetooth device name | char[33] | 33 |
| Bluetooth device name length | uint8_t | 1 |
| Bluetooth pair remote address | uint8_t[6] | 6 |
| Bluetooth pair link key | uint8_t[16] | 16 |
| | | 64 |

TABLE 5.2: Information memory segment A items.

| Item | Type | Size |
|---|---|---|
| Inverted placement | uint16_t | 2 |
| Gyroscope offset | uint16_t[3] | 6 |
| Gyroscope sensitivity | float[3] | 12 |
| Accelerometer offset | uint16_t[3] | 6 |
| Accelerometer sensitivity | float[3] | 12 |
| Magnetometer offset | uint16_t[3] | 6 |
| Magnetometer rotation matrix | float[9] | 36 |
| Calibration raw temperature | int16_t | 2 |
| | | 82 |

TABLE 5.3: Calibration values in information memory segments B, C and D.

# Chapter 6

# Sensor Calibration

Measurement values provided by MEMS MARGs are erroneous if used without calibration, leading to a less accurate determination of orientation. Due to microscopical differences between the internal structure of each MARG, calibration has to be performed for each one, as results for one MARG can not be used for another one of the same product line. Specifically, measurement values are affected by the following characteristics [9]:

- *Cross sensitivity.* Sensor axes might not be perpendicular to each other. When applying input to a single axis, a portion of it affects another one. These portions must be determined and moved to the actual axis.
- *Sensitivity.* When applying a known level of input to an axis, measurement values might be off by a certain factor that must be determined to calculate a scaling.
- *Zero-level offset.* When not applying any input to an axis, measurement values might not be zero. This offset must be removed.
- *Temperature drift.* Sensitivity and zero-level offset change depending on the MARG's temperature. MARG temperature must be tracked and compared to an initial value $t$ to allow adjustments. Drift values are can be found in the MARG's datasheet.

For calibration, it is recommended to take the the average of multiple measurement values when determining calibration values. This counters small variances and improves precision. Furthermore, the MARG ICs must be soldered well-aligned to the PCB borders, e.g. perfectly in parallel. Any deviations from this will lead to suboptimal calibration results when used one of the following methods.

In the following sections, calibration methods used in this work for the gyroscope, accelerometer and magnetometer are presented.

## 6.1 Gyroscope Calibration

The author assumes that the gyroscopes only show a marginal cross sensitivity, if any at all. The resulting heading error is expected to be below 1 degree [9].

The zero-level offset $\overrightarrow{s_g}$ can be determined by rotating the sensor at a constant angular velocity. A reflector is placed on the sensor, and a laser is used to measure the period between the reflections. This allows to precisely determine the actually achieved angular velocity. By comparing this to the one measured by the gyroscope, sensitivity $\overrightarrow{s_g}$ can be described accurately. However, such a calibration setup was not available to the author.

Sensitivity $\overrightarrow{s_g}$ and zero-level offset $\overrightarrow{o_g}$ have been determined by using an IKEA Kallax shelf, as depicted in figure 6.1. The author taped the PCBs in parallel, deviations however may be possible due to limited measurement means. $\overrightarrow{o_g}$ is determined by not moving the device and collecting measurement values for several minutes. Their mean is $\overrightarrow{o_g}$. For sensitivity $\overrightarrow{s_g}$, an approach derived from the work of M. Looney (2010) has been chosen [47]. The Kallax shelf is rotated by 90° around the positive and negative direction of each axis and angular velocity is integrated. This provides two degrees of rotation for each axis. The error of the integration, as described in section 2.2.1, is expected to be marginal since the integration occurs over a period of less than 5 seconds. By comparing the mean of the two rotatory degrees to the actual 90°, sensitivity $\overrightarrow{s_g}$ is determined.

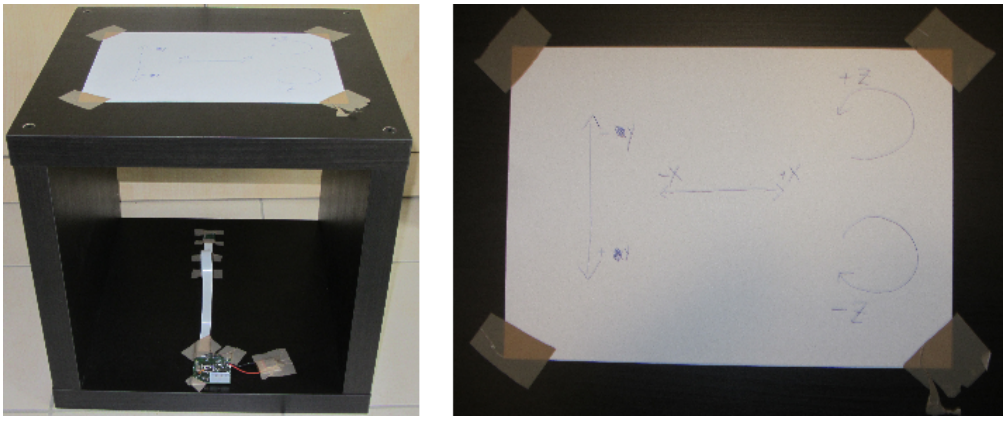


FIGURE 6.1: Glove PCBs mounted inside of IKEA Kallax shelf for gyroscope and accelerometer calibration (left). Gyroscope axes directions for shelf (right).

Finally, by taking temperature drifts $\overrightarrow{d_{g,s}}$ for sensitivity and $\overrightarrow{d_{g,o}}$ for zero-level offset into account, the calibrated gyroscope measurement values $\overrightarrow{v_g}$ can be calculated. This requires the deviation in temperature $\Delta t$ and the raw values $\overrightarrow{r_g}$. See equation 6.1.

$$\overrightarrow{v_g} = \overrightarrow{s_g} \times \overrightarrow{d_{g,s}} \times \Delta t \times (\overrightarrow{r_g} - \overrightarrow{o_g} - \overrightarrow{d_{g,o}} \times \Delta t) \tag{6.1}$$

## 6.2 Accelerometer Calibration

The author assumes that the accelerometers only show a marginal cross sensitivity. The resulting roll and pitch error is expected to be up to 1 degree [9]. When not moving the sensor, only the Earth's gravitation $G$ is measured by the accelerometer. This can be used for calibration. Sensitivity $\overrightarrow{s_s}$ and zero-level offset $\overrightarrow{o_a}$ can be determined by performing a 6-point tumble test with the aid of the IKEA Kallax shelf [9]. Position the sensor so that only a single axis $a$ measures $G$ in its positive range. ($a_{G+}$) The other two axes, $b$ and $c$, are expected to be zero. ($b_{0,1}$, $c_{0,1}$) Afterwards, turn the sensor upside down. The axis $a$ must now measure $G$ in its negative range. ($a_{G-}$) The other two axes are again expected to be zero. ($b_{0,2}$, $c_{0,2}$) Repeat this procedure for the other two axes.

In total, each axis measures $G$ in two opposite directions and no $G$ four times, during the 6-point tumble test. This yields $x_{G+}$, $x_{G-}$, $y_{G+}$, $y_{G-}$, $z_{G+}$, $z_{G-}$, $x_{0,1}$ to $x_{0,4}$, $y_{0,1}$ to $y_{0,4}$ and $z_{0,1}$ to $z_{0,4}$. Sensitivity $\overrightarrow{s_a}$ and zero-level offset $\overrightarrow{o_a}$ can be calculated from these values, see equations 6.2 and 6.3.

$$\overrightarrow{s_a} = \begin{pmatrix} (|x_{G+}| + |x_{G-}|) \times (2G)^{-1} \\ (|y_{G+}| + |y_{G-}|) \times (2G)^{-1} \\ (|z_{G+}| + |z_{G-}|) \times (2G)^{-1} \end{pmatrix} \tag{6.2}$$

$$\overrightarrow{o_a} = \begin{pmatrix} (x_{0,1} + x_{0,2} + x_{0,3} + x_{0,4}) \times 4^{-1} \\ (y_{0,1} + y_{0,2} + y_{0,3} + y_{0,4}) \times 4^{-1} \\ (z_{0,1} + z_{0,2} + z_{0,3} + z_{0,4}) \times 4^{-1} \end{pmatrix} \tag{6.3}$$

Combined with temperature drifts $\overrightarrow{d_{a,s}}$ for sensitivity and $\overrightarrow{d_{a,o}}$ for zero-level offset, deviation in temperature $\Delta t$ and raw values $\overrightarrow{r_a}$, the calibrated accelerometer measurement values $\overrightarrow{v_a}$ can be calculated. See equation 6.4.

$$\overrightarrow{v_a} = \overrightarrow{s_a} \times \overrightarrow{d_{a,s}} \times \Delta t \times (\overrightarrow{r_a} - \overrightarrow{o_a} - \overrightarrow{d_{g,a}} \times \Delta t) \tag{6.4}$$

## 6.3 Magnetometer Calibration

The magnetometers is affected by the Earth's magnetic field. The latter depends on the location and date, since it is moving. Services exist to look up the Earth's magnetic field values at a given latitude, longitude, elevation and date [48]. Unfortunately, these values can not be used as references for calibration due to the following error sources:

- *Hard-iron errors* "represent magnetic field sources, which add (or subtract) to the earth's magnetic field. Examples of this type of error are with permanent magnets, power supply currents or when local ferromagnetic material retains residual magnetic fields (become magnetized) from exposure to large magnetic fields." [49]

- *Soft-iron errors.* "Soft iron errors represent the magnitude and direction change that the earth's magnetic field experiences when near ferromagnetic objects." [49]

The magnitude of both error sources depends on the board layout and surrounding materials. This means magnetometer calibration has to be performed while the sensor is mounted on the finished device. Ideally, a 3-axis Helmholtz Coil should be used for calibration [9]. Such a device can apply a nearly uniform magnetic field to the magnetometer axes. This allows to precisely determine axes sensitivities and cross-sensitivity, as well as cancelling the Earth's magnetic field to determine any zero-level offsets. A Helmholtz Coil however was not available to the author.

For this work, to calibrate the magnetometer, the finished boards were taped to a piece of cardboard. Sensor packages were aligned by their axes. This setup, as seen in figure 6.2, has been used by the author to rotate the device, which is further explained in the following subsections. While this approach is less precise than using a Helmholtz Coil, it did increase magnetometer data quality and orientation precision.

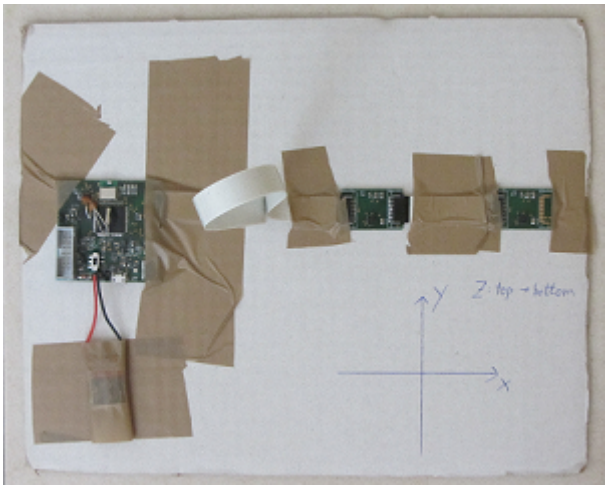


FIGURE 6.2: Glove PCBs on a piece of cardboard used for magnetometer calibration.

### 6.3.1 Raw Data Ellipsoid

When rotating a magnetometer around all axes and plotting the data as 3D points, a calibrated magnetometer will produce a sphere concentric to the principal axes. This is because by rotating the sensor, the Earth's magnetic field is measured in virtually all directions, forming the radius of a sphere. Figure 6.3 visualizes the read raw data for the sensor placed on the hand. The grey lines represent the rotation around all axes, while circles in light RGB represent the rotations around single axis. The lines in dark red, green and blue (RGB) show the principal axes and lines in light RGB display the principal axes as measured by the magnetometer.

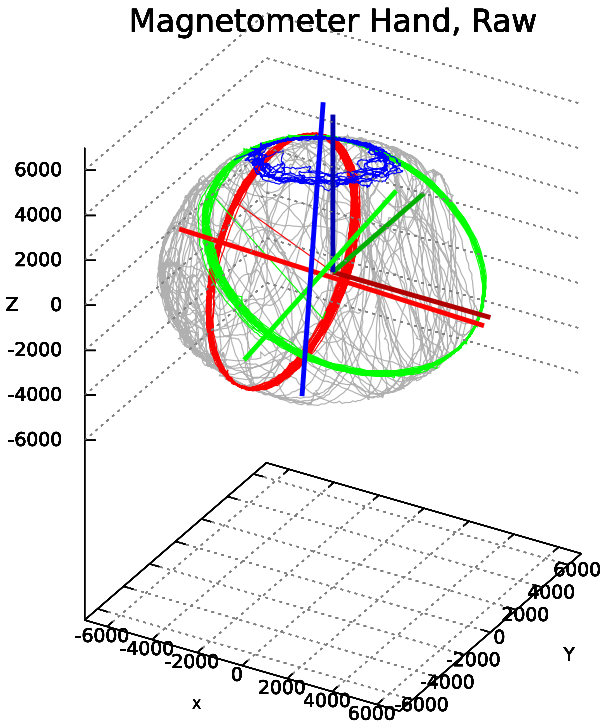


FIGURE 6.3: Magnetometer raw values of hand sensor.

From the figure and data can be seen that the axes are not perpendicular due to significant cross sensitivity. The body is an ellipsoid, indicating different axes sensitivities. The centre is not in the origin of the coordinate system due to zero-level offsets. A rotatory matrix $R_m$ is required to correct the cross sensitivity. Since the orientation filter algorithm, as described in section 2.2.4, relies solely on the Earth's magnetic field's direction and disregards its magnitude, sensitivity is not required. Sensitivity drift due to temperature can therefore be neglected. The zero-level offset $\overrightarrow{o_m}$ however must be determined to correctly calculate the direction. But, according to the sensor's datasheet, the zero-level offset is not affected by temperature changes. [19]

### 6.3.2 Rotation Matrix & Cross Sensitivity

The previous section visualized the principal axes as measured by the magnetometer. The axes are the surface normals of the surfaces fitted to the circles when rotating the device around each single axis. The surface normals can be fitted approximately as the mean of all pair-wise cross products between the circles' 3D data point vectors. Ideally, these surface normals describing the measured axes are aligned to their corresponding axis. For example, when rotating the device around its x axis, the measured axis $\overrightarrow{x}$ is expected to be equal to $\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$. For e.g. the right hand glove's hand magnetometer

however, $\overrightarrow{x}$ is equal to $\begin{pmatrix} -0.998469 \\ 0.054639 \\ -0.008570 \end{pmatrix}$. These values show that input on the y and z axes affect measurements on the x axis, and that input on the x axis affects measurements on the y and z axes by about 5% and –1% respectively. Such vectors exist for all axes.

To remove cross sensitivity for a value with inherent perpendicular axes $\overrightarrow{p_m}$, the raw value $\overrightarrow{r_m}$ must be multiplied by the rotation matrix $R_m$. The latter is formed by using the surface normal axes vectors as columns, assuming that $\overrightarrow{p_m}$ is linearly dependent from $\overrightarrow{r_m}$. See equation 6.5. This way, each axis' value of $\overrightarrow{p_m}$ is the sum of portions of input for this particular axis measured on all axes, just like the magnetometer actually measures.

$$\overrightarrow{p_m} = \overrightarrow{r_m} \times R_m = \overrightarrow{r_m} \times \begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{pmatrix} \tag{6.5}$$

Figure 6.4 visualizes measurement values multiplied by $R_m$ for the hand's magnetometer on the right hand. It can be seen that the axes are perpendicular, meaning that cross sensitivity has been removed. The body though is still not a sphere, but an ellipsoid with less deviances between the axes' radii. This implies that the calculated magnitude of the Earth's magnetic field is incorrect, which is acceptable since the magnitude is not relevant. The direction on the other hand can now be determined correctly once the ellipsoid's offset has been adjusted.

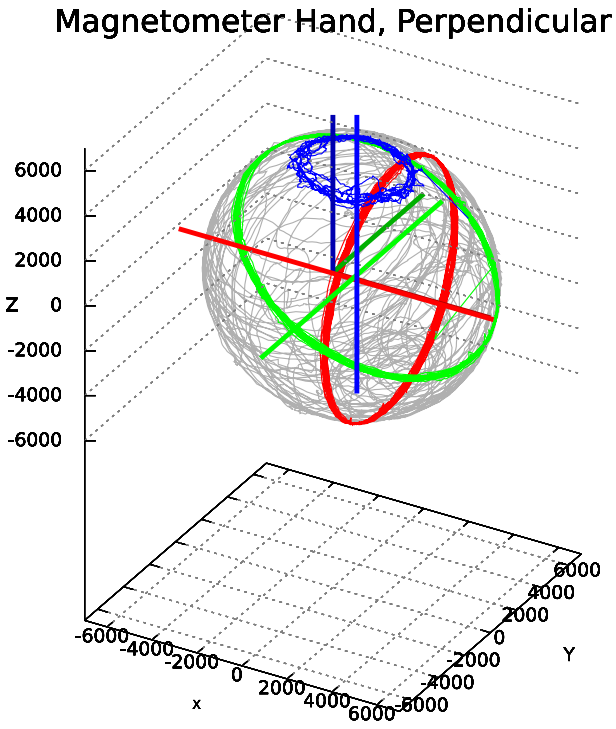


FIGURE 6.4: Magnetometer values of hand sensor with perpendicular axes.

### 6.3.3 Zero-level Offset

An ellipsoid can be fit to the 25,000 raw data 3D points using a MATLAB script by Yury Petrov [50]. The script returns, amongst others, the centre of the ellipsoid. The zero-level offset $\overrightarrow{o_m}$ of the magnetometer equals the centre of the ellipsoid. This is because the latter is expected to be centred in the origin of the coordinate system for a calibrated magnetometer. By using $\overrightarrow{o_m}$ and the rotation matrix $R_m$, the calibrated magnetometer value $\overrightarrow{v_m}$ can be calculated as seen in equation 6.6.

$$\overrightarrow{p_m} = \overrightarrow{r_m} \times R_m - \overrightarrow{o_m} \tag{6.6}$$

Figure 6.5 visualizes the fully calibrated magnetometer values. It can be seen that the axes are perpendicular and only minimally misaligned to the coordinate system's principal axes. The Earth's magnetic field's direction can now be determined.

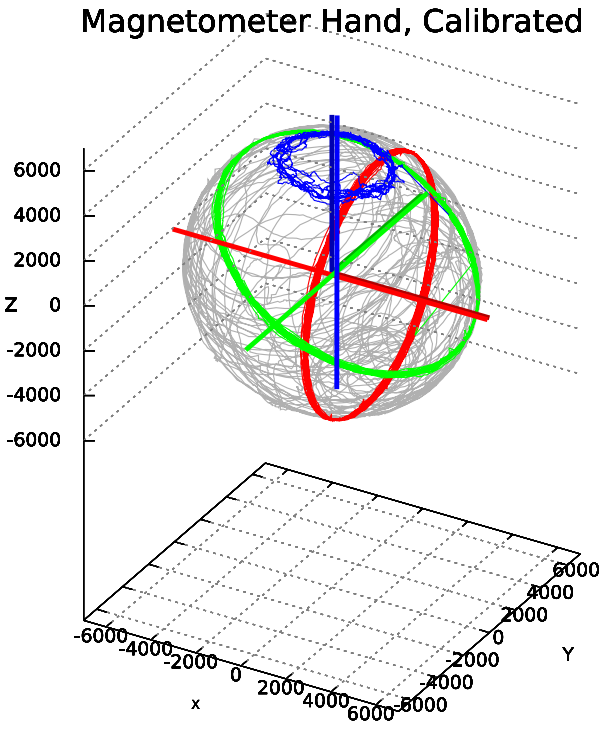


FIGURE 6.5: Magnetometer values of calibrated hand sensor.

## 6.4 Results

Table 6.1 presents calibration results. 250.000 gyroscope samples have been used to determine $o_g$. For $s_g$, about 1.500 samples over the 5 seconds rotation period have been taken. 50.000 samples have been taken for each of the 6-point tumble test's orientations used for accelerometer calibration. For magnetometer calibration, 25,000 measurement

values have been read when rotating the device around all axes. Additionally, 10,000 measurement values have been read when rotating the device around single axis'. $t$ is the approximate raw temperature value of the MARG during calibration.

| | **Hand** | **Forearm** | **Upper Arm** |
|---|---|---|---|
| $\boldsymbol{t}$ | $-500$ | $-500$ | $-400$ |
| $\boldsymbol{o_g}$ | $\begin{pmatrix} -0.21°/s \\ 0°/s \\ 0°/s \end{pmatrix}$ | $\begin{pmatrix} 0°/s \\ 0.14°/s \\ -0.35°/s \end{pmatrix}$ | $\begin{pmatrix} -0.56°/s \\ 0°/s \\ 0.91°/s \end{pmatrix}$ |
| $\boldsymbol{s_g}$ | $\begin{pmatrix} 0.98 \\ 1.02 \\ 1 \end{pmatrix}$ | $\begin{pmatrix} 1.07 \\ 1.03 \\ 1.05 \end{pmatrix}$ | $\begin{pmatrix} 1.21 \\ 1.22 \\ 0.8 \end{pmatrix}$ |
| $\boldsymbol{o_a}$ | $\begin{pmatrix} -0.31G \\ -0.28G \\ 0.11G \end{pmatrix}$ | $\begin{pmatrix} -0.32G \\ -0.27G \\ 0.1G \end{pmatrix}$ | $\begin{pmatrix} -0.23G \\ -0.28G \\ 0.18G \end{pmatrix}$ |
| $\boldsymbol{s_a}$ | $\begin{pmatrix} 1.14 \\ 1.14 \\ 1.17 \end{pmatrix}$ | $\begin{pmatrix} 1.13 \\ 1.11 \\ 1.11 \end{pmatrix}$ | $\begin{pmatrix} 1.22 \\ 1.21 \\ 1.18 \end{pmatrix}$ |
| $\boldsymbol{R_m}$ | $\begin{pmatrix} -1 & 0.05 & -0.01 \\ -0.09 & 0.99 & 0.01 \\ 0.01 & 0.13 & 0.99 \end{pmatrix}$ | $\begin{pmatrix} 1 & -0.09 & -0.01 \\ 0.06 & -1 & -0.01 \\ 0.06 & -0.07 & 0.99 \end{pmatrix}$ | $\begin{pmatrix} 1 & -0.08 & 0.02 \\ 0.12 & -0.99 & -0.04 \\ 0.03 & -0.02 & 1 \end{pmatrix}$ |
| $\boldsymbol{o_m}$ | $\begin{pmatrix} 0.05gauss \\ 0.03gauss \\ -0.01gauss \end{pmatrix}$ | $\begin{pmatrix} -0.06gauss \\ -0.03gauss \\ 0.04gauss \end{pmatrix}$ | $\begin{pmatrix} -0.06gauss \\ -0.02gauss \\ 0.04gauss \end{pmatrix}$ |

TABLE 6.1: Glove calibration results.

Noteworthy are the accelerometer offsets, where most axes measure about a quarter to a third of $G$ if no acceleration is applied. Furthermore, accelerometer values require scaling of 11% to 22%. Using raw values would result in a miscalculation of gravitation direction, which is crucial for the correction of angular velocity integration. Regarding the gyroscopes, the upper arm one's nearly measures a 1°/s angular velocity on its z axis and requires significant scaling. Without the latter, there would be a noticeable difference between the actual and measured rotation. The magnetometers show significant cross-axis sensitivities, where e.g. 13% of input on the hand MARG's y axis are measured on the z axis, leading to erroneous integration corrections and orientations.

These values demonstrate the necessity of MEMS MARG calibration. The reader must keep in mind that the used methods were not optimal and that calibration results for more precise orientation calculations are possible with proper equipment.

# Chapter 7

# Experiments

In this chapter, experiments regarding the design goals are described and their results presented. Section 7.1 provides an overview on different glove firmware configurations and resulting readout rates, latencies and power consumption. In section 7.2, the round-trip time of readouts and delays between them are presented. Sections 7.3 and 7.4 show achieved errors in static orientation and position depending on the AHRS configuration, as well as accuracy and convergence delays in determining the hand's position during motions. All experiments have been carried out using the PC specified in table 7.1.

| | |
|---|---|
| Manufacturer | Apple Inc. |
| Model | MacBook Pro (13-inch, Early 2011) |
| CPU | Intel Core i5-2415M @ 2.30GHz |
| RAM | 2x 4GB PC3-10600 DDR3 1333MHz |
| SSD | Samsung SSD 840 Pro 256GB |
| Bluetooth Adapter | Broadcom BRCM2070 |
| OS | Microsoft Windows 8.1 Pro |
| Bluetooth Driver | Apple Broadcom Built-in Bluetooth v5.0.4.0 |

TABLE 7.1: Specification of PC used for experiments.

## 7.1 Firmware Configuration Differences

In this section, table 7.2 presents readout delay and rate, minimum latency excluding data transmission and power consumption for different glove firmware configurations. The readout delay is the time between received readout data, correlated to the readout rate. 25,000 samples have been taken for each configuration to determine the mean readout delay. The minimum latency consists of delays in the glove firmware and PC application, but does not include data transmission between the Bluetooth modules. Power consumption includes the actual current consumption and resulting device runtime. All configuration use increased SPP buffer sizes, as explained in section 7.2.

| Configuration ID | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| System frequency [MHz] | 20 | 25 | 25 | 25 | 25 | 8 |
| SPI frequency [MHz] | 10 | 6.25 | 6.25 | 6.25 | 6.25 | 8 |
| UART speed [bps] | 1M | 1M | 1M | 1M | 1M | 250k |
| Pre-set readout rate [Hz] | 500 | 500 | 500 | 500 | 100 | 100 |
| DMA | | | ✓ | | | |
| Compression | | | | ✓ | | |
| Sniff mode | | | | | ✓ | ✓ |
| Achieved readout delays and rates | | | | | | |
| Delay, mean [ms] | 2.45 | 1.97 | 2.36 | 2.6 | 9.97 | 10.04 |
| Rate, mean [Hz] | 408 | 508 | 424 | 384 | 100 | 100 |
| Min. achieved delays affecting latency during stationary positioning | | | | | | |
| Readout [ms] | 0.92 | 0.76 | 0.95 | n/a | 0.76 | 2.41 |
| Serialization [ms] | 0.33 | 0.26 | 0.26 | n/a | 0.26 | 0.82 |
| UART [ms] | 0.56 | 0.56 | 0.56 | n/a | 0.56 | 2.26 |
| Deserialization & AHRS [ms] | 0.19 | 0.19 | 0.19 | n/a | 0.19 | 0.19 |
| Min. latency excl. transmission [ms] | 2 | 1.8 | 2 | n/a | 1.8 | 5.7 |
| Power consumption | | | | | | |
| Current consumption [mA/h] | n/a | 91.2 | n/a | 90.5 | 69.4 | 78.3 |
| Runtime, 180mAh LiPo [h:min] | n/a | 1:58 | n/a | 1:59 | 2:36 | 2:18 |

TABLE 7.2: Readout delays, latency and power consumption for glove firmwares.

As explained in section 5.1, the achieved readout rate when using a 20MHz system and 10MHz SPI frequency should be compared to using 25MHz and 6.25MHz respectively. Configurations A and B provide these results for a pre-set readout rate of 500Hz at the maximum UART frequency of 1Mbaud. Furthermore, the effects of using DMA during sensor readout and applying differential data compression on readout data, as described in sections 5.4 and 5.5 respectively, are shown by configurations C and D. Configurations E and F present low-power mode results. This means that the pre-set readout rate has been reduced to 100Hz to allow the usage of the Sniff mode (see section 3.4.3). Minimum interval = 14, maximum = 16, attempt = 1 and timeout = 2. Configuration F additionally reduces the system frequency to 8MHz and increases the SPI frequency the 8MHz, too. The UART speed however must be lowered to 250kbaud to ensure proper communication between the microcontroller and the Bluetooth module.

Configuration A achieves a mean readout rate of about 408Hz, which is lower than the desired 500Hz. Power consumption has not been measured, as this configuration is irrelevant for the design goals. Configuration B fulfils the design goals by achieving a mean 508Hz readout rate. The author assumes that being over 500Hz is due to statistical errors. Latency is at least 1.8ms and the device can run for 1:58 hours.

Configuration C shows that using DMA during sensor readout reduces the mean achieved readout rate. Power consumption has not been measured, as this configuration is irrelevant for the design goals. Configuration D, utilizing compression, also comes with a reduced mean achieved readout rate, making it irrelevant. But the device runtime increased by a minute. The minimum latency has not been determined for D since it varies with the data size resulting from the compression. Configuration E realizes a low-power mode at 100Hz readout rate and using Sniff mode, and decreases current consumption by 21.8mA/h compared to B. Setting the frequencies the 8MHz with configuration F on the other hand increases current consumption by 8.9mA/h compared to E. Furthermore, the minimum latency increases by 3.9ms.

## 7.2 Readout Round-trip Time & Delays

Figure 7.1 presents round-trip time (RTT) results for readouts. The PC application starts a timer before sending 65 bytes of data, which equals the size of a readout, to the glove. After the glove received all bytes, it sends 65 bytes back. The timer in the PC application is stopped once all 65 bytes from the glove have been received. 10,000 samples have been taken. The minimum equals about 1ms, maximum 43ms and mean 8ms. Note that almost all round-trips require either about 1ms, 8ms or 15ms ± 1ms.

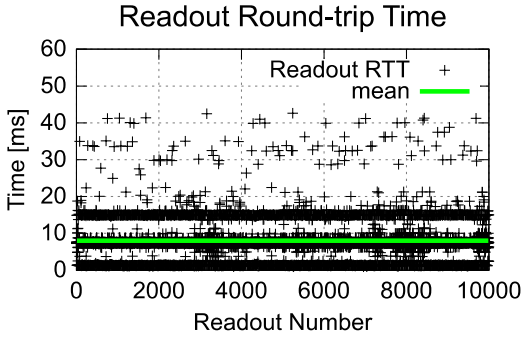


FIGURE 7.1: Round-trip time of readout data between PC application and glove.

Figure 7.2 shows measurements for the delay between received readouts for configuration B. While the pre-set readout rate in the glove firmware is set to 500Hz in this case, readouts may arrive at the PC application with different delays. This depends on the Bluetooth transmission and whether or not multiple readouts can be completely or partially transferred during slave-to-master (glove-to-PC-application) timeslots of different packet sizes. The left plot provides results for using the default SPP buffer sizes of 58 bytes in the glove firmware. The mean readout delay equals 3.77ms, indicating that the pre-set readout rate of 500Hz can not be achieved. For measurements shown in the right plot, the SPP receive and write buffer sizes have been increased to 256 and 1024 bytes respectively. The mean readout delay has been reduced to 1.97ms, allowing a 500Hz rate. 10,000 samples have been taken each.

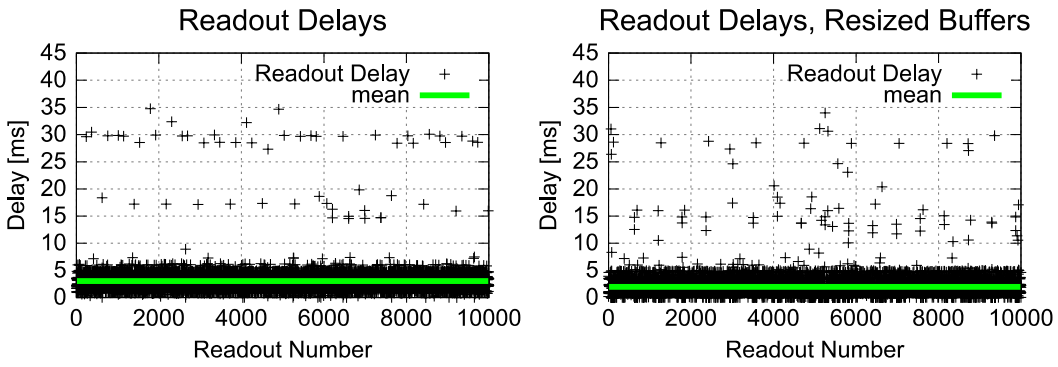


FIGURE 7.2: Delays between received readouts, without (left) and with (right) increased SPP buffer sizes.

Noteworthy is the distribution of readout delays, as shown in the left plot of figure 7.3 using increased SPP buffer sizes, too. 10,000 samples have been taken. Almost all readouts arrive with a delay that is a multiple of roughly 1.25ms, equal to two timeslots. This corresponds to the communication scheme, as explained in section 3.4.1. After one master to slave timeslot, the slave answers during one, three or five timeslots, depending on the data size. The resulting transmission time is therefore always a multiple of two timeslots. The author assumes that the high delays, like the maximum delay of about 35ms, occur due to protocol mechanisms like frequency hopping or retransmissions.

The right plot of figure 7.3 presents the readout delays distribution for configuration E. 10,000 samples have been taken. The pre-set readout rate has been lowered to 100Hz to allow the usage of Sniff mode, and a corresponding mean delay of 9.97ms has been achieved. About 80% of readouts arrive with a delay of 10ms or less, which corresponds to the pre-set rate. About 20% on the other hand are affected by a delay of about 33ms. The author assumes that this configuration allows the Bluetooth module to regularly enter some low-power mode which requires about 30ms to exit from it.

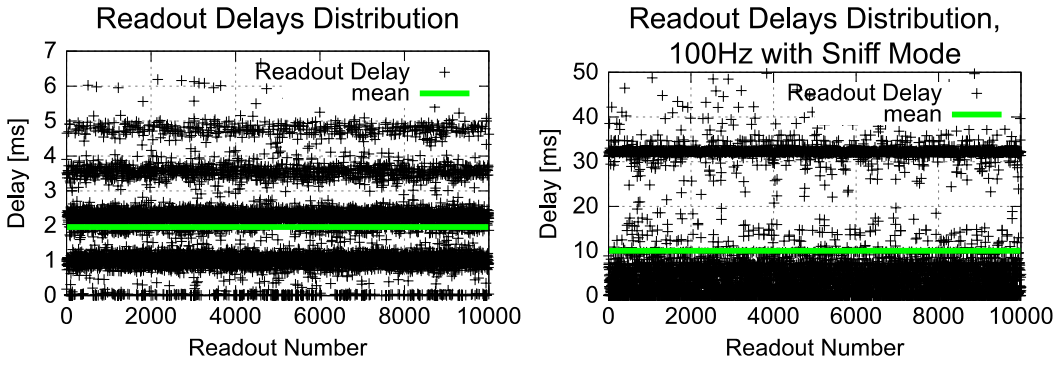


FIGURE 7.3: Delays between received readouts for 500Hz (left) and 100Hz with Sniff mode (right).

## 7.3 Static Orientation & Position Error

The AHRS algorithm comes with two parameters, $\beta$ and $\zeta$, as described in section 2.2.4. Effects of these can be seen in figures 7.4 and 7.5, presenting static hand orientation and position errors when stationary over a period of five minutes using configuration B.

The left plot in figure 7.4 demonstrates that integrating unfiltered angular velocity data provided by the gyroscope results in significant errors. Roll and pitch oscillate in a range of about 75° and 100°. Yaw is affected by bias drift and exceeds an error of 360°. The change from 180° to −180° occurs because the angles are encoded with positive and negative ranges to indicate direction. The right plot shows the effect of $\beta$. For the two measurements shown in the plot however, $\beta$ was set to the proposed optimal value of 0.041 for Madgwick's MARG, as well as to the proposed maximum of 0.5, for all three MARGs. It can be seen that errors for roll and pitch are below at least 5° for both values. Yaw is still affected by bias drift for a $\beta$ of 0.041; the error is above 50° after five minutes. This can not be seen for a $\beta$ of 0.5, which is due to the short time frame.

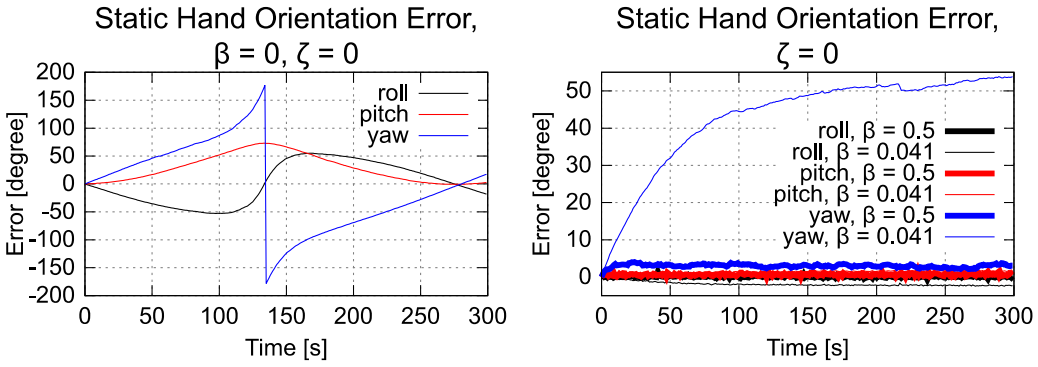


FIGURE 7.4: Static orientation error of hand MARG from integrating raw gyroscope data (left) and using $\beta$ filter gain, compensating mean zero gyroscope errors (right).

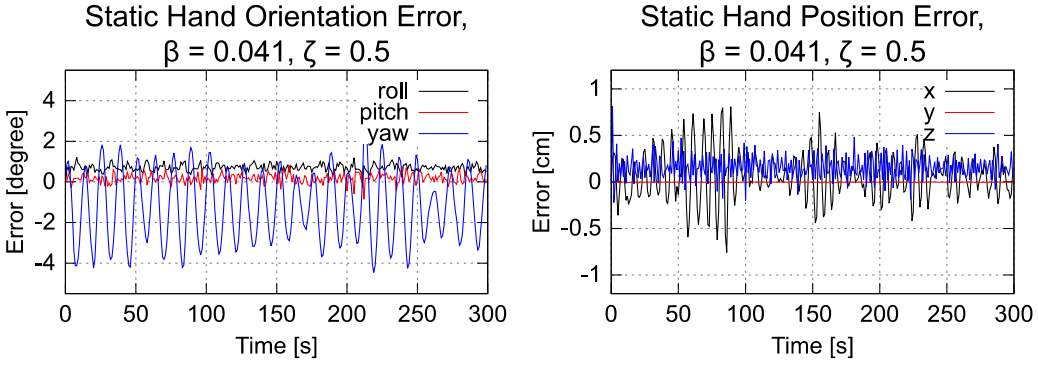


FIGURE 7.5: Static orientation error of hand MARG from compensating mean zero gyroscope errors and bias drift (left). Static hand position error resulting from arm parts' MARG orientations using the same compensation.

Figure 7.5 demonstrates the effect of $\zeta$. The left plot shows measurements for a $\zeta$ of 0.5, which has been chosen arbitrarily. The bias drift on yaw is removed. It oscillates between about $-4°$ and $2°$ due to continuous bias drift error estimation and correction. Errors for roll and pitch are below an absolute $1°$. For the same filter gain values, the right plot presents the hand's position error, resulting arm parts' orientations. The errors on all axes are below an absolute 1cm, most of the time even below an absolute 0.5cm. The error on the y axis is virtually zero.

## 7.4 Position & Orientation during Motion

Any orientation calculated by the AHRS algorithm, as presented in section 2.2.4, is potentially off during motion of the MARG and converges to the actual value once stationary. The following experiments present effects of different sources on the hand's position during motion, as well as the convergence delay afterwards. The author's arm part lengths, as utilized in these experiments, are 25cm for the upper arm and 30cm for the forearm. Initially, the arm is stretched out forward, leading to a hand position on the x axis (left to right from the author's perspective) of 0cm, y axis (back to front) of 55cm and z axis (down to up) of 0cm. Configuration B has been utilized. Two motion patterns have been used: First is the shoulder rotation, during which the stretched-out arm is moved in a circular motion around the shoulder before returning to the initial position. This takes about 3 to 5 seconds. Second is the $90°$ rotation, during which the stretched-out arm is moved in less than a second by $90°$ to the author's right. Both patterns occur at the beginning of each of the following experiments. Sensors have been calibrated indoors.

Figure 7.6 presents the hand position during shoulder rotation, indoors and with temperature drift compensation. A $\beta$ of 0.041 and a $\zeta$ of 0.5 have been used. The algorithm requires more than 70 seconds after the motion stops to converge the hand's position to the actual, initial one. Note that during the measurements, the MARGs' temperature sensors stated a difference of multiple dozens degrees Celsius compared to the time of calibration, which is not reliable.

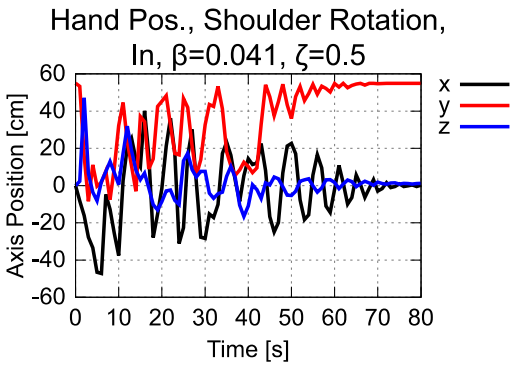


FIGURE 7.6: Hand position during shoulder rotation. Temperature drift compensated.

Figure 7.7 is based on the previous measurement. The left plot shows the results without temperature drift compensation, resulting in a convergence delay of about 30 seconds. The right plot is based on the left one, but performed outdoors. The convergence delay is about 70 seconds, and the axes show further continuous oscillations compared to figure 7.6. The x axis position is off by about -20cm, which the author assumes is due to occurring MARG misalignments in the fabric, as they are not tightly fixated.

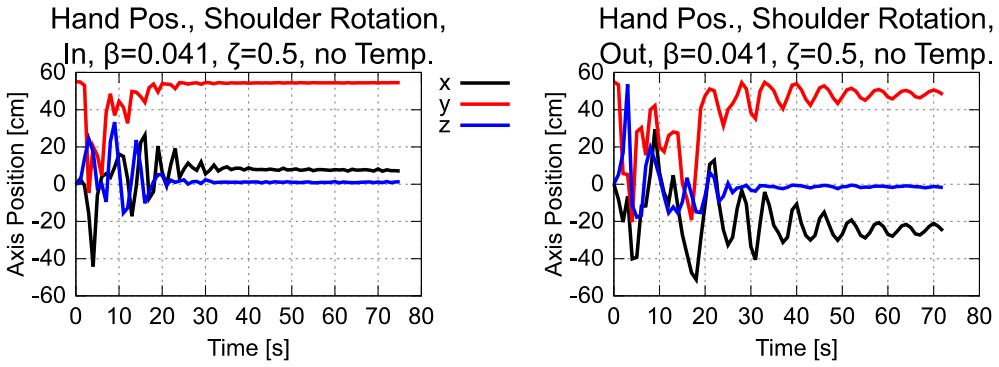


FIGURE 7.7: Hand position during shoulder rotation without temperature drift compensation: Indoors (left) and outdoors (right).

For the measurements of figure 7.8, $\beta$ has been increased to 0.5. The experiments are performed indoors and without temperature drift compensation. The left plot, showing the shoulder rotation starting at second 3, displays that the convergence delay has almost been reduced to zero. Once stationary, the axes positions are converged at about second 8. The x axis position is off again due to MARG misalignments in the fabric. The right plot shows the rapid 90° rotation. The motion is performed from about second 0.6 to 1.4. The x and z axis positions require about 1 second to converge afterwards, whereas the y axis position requires about 3 seconds. This indicates higher errors in the orientation's calculation during fast motions, as it can be seen for the hand in figure 7.9.

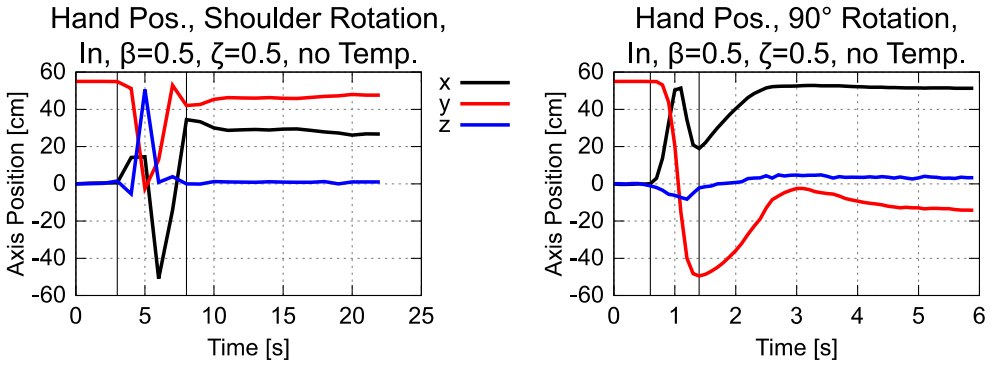


FIGURE 7.8: Hand position for increased $\beta$, with temperature drift compensation and indoors: During shoulder rotation (left) and fast 90° rotation (right).

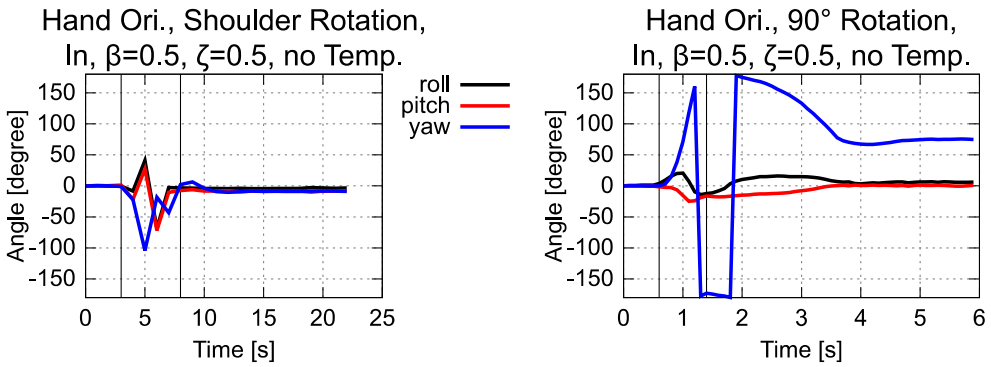


FIGURE 7.9: Hand orientation for increased $\beta$, with temperature drift compensation and indoors: During shoulder rotation (left) and fast 90° rotation (right).

## 7.5 Compression

Figure 7.10 depicts the sizes of 1,500 compressed readouts for a stationary hand and during shoulder rotation. Mean sizes are below 65 bytes of a raw readout: 49 bytes when stationary, and 51 bytes during the motion. Both cases show a significant distribution. The author assumes the reason is sensor noise, leading to changes in the values' LSBs.

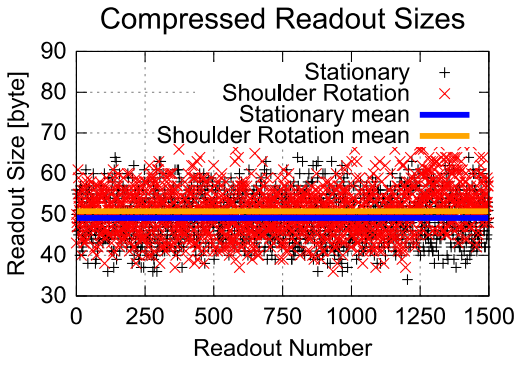


FIGURE 7.10: Sizes of compressed readouts: Stationary and during shoulder rotation.

# Chapter 8

# Discussion

With regards to the design goals, this chapter discusses design decisions and experimental results. Section 8.1 focuses on achieved accuracy for orientation and hand position calculations, as well as their error sources. Latencies between motions and the availability of the aforementioned data are criticized in section 8.2. Section 8.3 discusses power consumption and how to reduce it, while section 8.4 refers to the remaining design goals.

## 8.1 Orientation & Hand Position

The absolute 3D orientation of the hand can be determined with a static error of less than about 4°. Suboptimal calibration, as well as filter gain parameters not suited to the used MARGs, are the reasons for the static error not being equal to 1° or less, as possible according to Madgwick. By utilizing proper calibration devices and carefully choosing filter gain parameters, sub-degree static orientation accuracy is possible. This would further decrease the static hand position error, which currently is mostly below an absolute 0.5cm. The author considers this to be already acceptable for audio applications.

During motions however, calculated orientations are off and show a convergence delay to their actual values once the MARG is stationary. As this applies to the arm MARGs, too, the calculated hand position is off. Depending on the filter gain parameters and the velocity of the motion, the delay can be in the range of multiple seconds. While slow motions profit from a high $\beta$ and show a small convergence delay, fast motions require a low $\beta$ to not have off orientations after they stop. This is especially crucial during long gestures or continuous motions like dancing, where values would be constantly off. The problem however is inherent to the AHRS algorithm, since results are erroneous during motions and it is assumed that "accelerations due to motion and local magnetic distortions are present for only short periods of time." [11] This may be true for aircrafts, where such algorithms are commonly used, but not for hands and arms. To solve this, "it may be beneficial to use dynamic values of gains $\beta$ and $\zeta$. This will

the influence accelerometers and magnetometers have on the estimated orientation to be reduced during potential problematic periods; for example, when large accelerations are detected." [11] Such a dynamic adaptation mechanism however was not explored in this work.

As the experiments have been done with a readout rate of 500Hz, it can be seen that rates above 100Hz provide no advantage in accuracy or convergence delay if parameters are not chosen well and the sensors are not properly calibrated. This limits the advantages of high readouts rates to reduced delays and latency only. Furthermore, calibration requires precise placement of the MARGs on the PCBs, correct alignments during calibration and suitable equipment. Otherwise, errors, oscillations or convergence delays are noticeable. Miscalibrated MARGs may even result in calculated changes to angles that have not actually been changed, leading to confusion.

Furthermore, the used temperature sensor integrated into the MARGs is not reliable. Using its values for temperature drift compensation lead to less precise orientation results. It has also been demonstrated that using an indoors calibrated device outdoors affects calculations and introduces oscillations in orientation angles. This is because the Earth's magnetic field is distorted indoors by ferromagnetic objects and materials in the environment, and the AHRS algorithm tries to compensate this. For best results, all magnetometers must be recalibrated in the individual environment in which they are going to be used, which is impractical for the user. However, the error in orientation due to magnetic distortions might be small enough to be acceptable for the user, removing the need for recalibration, especially when utilizing suitable filter gain parameters. Still, the sensors must have at least been properly calibrated once.

Regarding the design goals, the hand's orientation and position can not be determined accurately and fast enough for every kind of motion. Future work must explore dynamic filter parameter adaptation and proper calibration techniques must be applied.

## 8.2 Latency

The latency between sensor readouts and availability of orientation and hand position data has not been determined empirically. The issue is that the transmission time can not be specified, as internals of the glove's and PC's Bluetooth modules, as well as their drivers, are not accessible. What has been determined is the minimum latency for the glove firmware and PC application software, which, for configuration B, is about 1.8ms. Furthermore, assuming that both the glove's and PC's Bluetooth modules account for half of the readout round-trip time each, it mostly requires about 0.5ms, 4ms or 7.5ms to transmit a single set of readout data.

The transmission behaviour however changes once readouts are continuously tried to be sent at 500Hz. The delay between received readouts is about a multiple of 1.25ms between 0ms and 5ms for most readouts. Values below the mean of 2ms are possible

if multiple readouts are completely or partially transmitted, which means that at least two readouts must have been buffered in the Bluetooth module. This increases latency for the readout to be sent first, as its transmission is delayed. Values above 2ms occur if data is split into packets of multiple slave to master (glove to PC application) timeslots, effectively increasing latency. By looking at the data, multiple consecutive delays of more than 2ms however are rare. Furthermore, note that all incoming readouts may be subject to a general latency offset, independent of the delay between them. Delays in the double-digit range appear for less than 0.5% of the readouts. By looking at this data, the author assumes that the latency between sensor readout and data availability seldom exceeds 10ms, which satisfies the design goal for low latency.

If the glove is used in its low-power mode with a 100Hz readout rate and utilizing Sniff mode in-between, about 20% of the readouts show a delay of more than 30ms. This is off-putting for musicians and does not satisfy the low latency goal.

## 8.3 Power Consumption

Using data compression at a 500Hz readout rate (configuration D), the sensor readout size decreased from 65 bytes to a mean of about 49 to 51 bytes, equal to about 23% less. This resulted in a 0.8% reduction in current consumption, from 91.2mA/h to 90.5mA/h. Obviously, decreasing the transmission data size does not necessarily lead to a reduction in power consumption. This is because the timeslot-based communication pattern requires the power-consuming exchange of link maintenance data to keep the connection alive. Only by using the Sniff mode, the keep-alive mechanism can be suspended for a well-defined period, allowing the Bluetooth modules to enter power-saving modes. The usage of 100Hz and Sniff mode (configuration E) reduced current consumption by 24% to 69.4mA/h.

The 100Hz readout rate can also be achieved with lower clock frequencies, like 8MHz for the system and 250kbaud for the UART (configuration F). Current consumption however increases to 78.3mA/h again. During its active mode, that is when not being in a power-saving mode, the MSP430F5438A typically draws 1.84mA at 8MHz and 8.9mA at 25MHz [22], showing significant savings for the former compared to the latter. Computation times however increase, leading to less usage of power-saving modes. Furthermore, communication times over the UART increase, too. At least these effects add up to an increased power consumption at lower frequencies.

To save power, future work should evaluate the use of Bluetooth Low Energy. While it is limited to a readout rate of about 133Hz, the systematic utilization of connection intervals might be superior to the Sniff mode in terms of power consumption, and probably even latency if the 30ms delays can be avoided. Additionally, other MARGs should be evaluated. The LSM9DS0's gyroscope draws 6.5mA, while e.g. the MPU-9250's one requires only about half [18, 19]. Power-saving modes of the LSM9DS0 have not been utilized in this work. At a 500Hz readout rate, the delay between exiting such a mode

and the availability of sensor values is too high to achieve 500Hz. If however Bluetooth Low Energy is used, utilizing these modes might be viable at the cost of an increased latency.

With its current implementation, the glove draws too much power for a LiPo small enough to fit on the back of the hand and allowing a runtime of six hours or more.

## 8.4 Remaining Design Goals

The following remaining design goals have been reached: Translation of sensor data to MIDI signals, 500Hz readout rate, 100Hz readout rate with reduced power consumption, PCB size small enough to fit into a glove on the back of the hand, wireless operation, LiPo rechargeable via USB, and capacitive touch.

Linear velocity and angular acceleration errors correlate to those of the hand's orientation and position, as the former are derived from the latter. Therefore, linear velocity and angular acceleration can not be determined accurately for every kind of motion and require future work.

Finger flexing can be determined accurately with the used sensor stripe. Multiple stripes an a single finger, e.g. one per knuckle, would allow for an even more precise finger flex representation.

A vibration motor is not reasonably usable with the current implementation, as it can not be controlled or stopped.

The effects on readout latency and delay when using two gloves have not been determined. Due to soldering issues with the Bluetooth module, the second glove is not functional.

# Chapter 9

# Conclusion

This work presents the implementation of a wireless sensor glove providing the hand's orientation and position relative to the shoulder, linear and angular velocity and acceleration, as well as finger flexing. This allows the user to control DAW software with his hand motion. Important aspects are accuracy of the provided values, latency between sensor readout and value availability, power consumption, as well as the PCB sizes.

The accuracy is sufficient for simple either slow, continuous motions, or rapid motions, depending on the chosen filter parameters. Future work must research a dynamic adaptation mechanism to provide accurate results for arbitrary motions. Furthermore, proper calibration devices and precise methods must be utilized to increase orientation quality. The author expects that for both during motion and if stationary, sub-degree accuracy is possible. Commercially available AHRS platforms using MEMS MARGs demonstrate this [51].

At a high readout rate of 500Hz, latency rarely exceeds 10ms, which is sufficient for the purpose of DAW control. Power consumption however disallows the use of small LiPos placed on the hand. To reduce power requirements, Bluetooth Low Energy has to be evaluated in future work, as it is expected to be superior to using the Sniff mode in terms of power consumption and latency. The increase in readout delay by being limited to 133Hz however might lead to noticeable lags, which should be evaluated by performing user studies. However, when using Bluetooth Low Energy, the developer must provide its own SPP implementation. The latency and readout delays of the latter have to be evaluated.

This work demonstrates that small-sized PCBs are already realizable with components available today. Future work must test whether a system frequency of 8MHz is sufficient to drive the readout rate with 133Hz and Bluetooth Low Energy. If possible, the level converter between microcontroller and Bluetooth module could be removed, as the 1.8V required by the latter are sufficient to reach 8MHz for the microcontroller. Furthermore, the FT232R BSL circuit can be removed, and the JTAG port can be replaced by a two-port Spy-Bi-Wire interface. The author expects that with these changes, together with

smaller switches and a double-sided mounting of components, the size of the hand PCB can be reduced to 3×3 $cm^2$ or less. The FFC cables however should be replaced by more flexible ones to prevent the arm PCBs from being displaced in the textile due to stiff cables. Additional future research on capacitive fabrics can be used to realize buttons and sliders on the textile itself, if connected to a capacitive pin, as shown in this work.

While small-sized PCBs suitable for textile gloves and arm wraps exist today, further research must be done on component selection to decrease power consumption. Readout latency is low enough to be suitable to control DAW software. The hand's orientation and position values however can only be utilized as DAW controls for a limited set of motions. Future work is required to provide quality values during arbitrary motions.

# Appendix A

# Code, Hardware Design & Data

The source code for the glove firmware and the PC application, the hardware schematic and board design files, as well as the experiments' data, can be accessed at:

https://github.com/hlohse/sound-glove

# Appendix B

# Bill of Materials

| Part Type | Manufacturer | Number/Value | Qty. |
|---|---|---|---|
| Micro USB port | FCI connect | 10118193-0001LF | 2 |
| Pin header | Wurth Electronics | 61303211121 | 14 |
| Switch | C&K components | JS202011SCQN-DPDT | 4 |
| Fuse | Vishay Beyschlag | MFU0603FF00500P100 | 2 |
| Schottky diode | Vishay Semiconductors | LL103A-GS08 | 2 |
| Ferrite bead | Laird-Signal Integrity Products | MI0805K400R-10 | 2 |
| Dual LED | Panasonic | LNJ167W8RRA | 2 |
| LiPo charge IC | Microchip Technology | MCP73831T-2 | 2 |
| Switching volt. regulator | Texas Instruments | TPS622316 | 2 |
| Linear volt. regulator | Texas Instruments | LP5951MF-1.8 | 2 |
| Inductor 0603 | Murata Manufacturing | LQM18FN2R2M00 | 2 |
| Microcontroller | Texas Instruments | MSP430F5438A | 2 |
| Level Converter | Texas Instruments | TXB0106 | 2 |
| Bluetooth module | Panasonic | PAN1325B | 2 |
| MARG sensor array | STmicroelectronics | LSM9DS0 | 6 |
| JTAG 14-pin header | Assmann WSW Components | AWHW14G-0202-T-R | 2 |
| RS232 converter | FTDI | FT232R | 2 |
| LED | Osram Opto Semiconductors | LG L29K-F2J1-24 | 8 |
| FFC port | FCI | HFW12R-1STE1LF | 8 |
| FFC 12” | Parlex USA | 100R12-305B | 2 |
| FFC 2” | Parlex USA | 100R12-51B | 2 |
| MOSFET transistor | International Rectifier | IRLML6402PbF | 2 |
| Flex sensor stripe | Spectra Symbol | FS-L-0095-103-ST | 1 |
| Diode | Diodes Incorporated | S1A-13-F | 2 |
| iPhone 5 vibration motor | | | 1 |
| LiPo 180mAh | ACME | AA0150-E | 2 |
| Glove | Jasmine Silk | Pure Silk Gloves | 2 |
| Crystal resonator | Multicomp | MCRJ332768F1220HOW | 2 |
| Capacitor thru-hole | | 18p | 4 |
| Capacitor 0805 | | 2.2u | 2 |
| Capacitor 0805 | | 4.7u | 8 |
| Capacitor 0805 | | 10u | 12 |
| Capacitor 0402 | | 2.2n | 2 |
| Capacitor 0402 | | 10n | 2 |
| Capacitor 0402 | | 100n | 14 |

| | | | |
|---|---|---|---|
| Capacitor 0402 | | 220n | 6 |
| Resistor 0402 | | 0 Ω | 4 |
| Resistor 0402 | | 600 Ω | 4 |
| Resistor 0402 | | 1k | 8 |
| Resistor 0402 | | 2.5k | 2 |
| Resistor 0402 | | 27k | 14 |
| Resistor 0402 | | 47k | 2 |
| Resistor 0402 | | 56k | 2 |
| Resistor 0402 | | 1M | 2 |

TABLE B.1: Bill of Materials.

# Bibliography


[1] J.-L. Tirpitz, H. Lohse. *DJ-Handschuh.* http://joanna.iwr.uni-heidelberg.de/projects/2014SS_DJHANDSCHUH/index.html (29.06.2015)

[2] RFilkov.com. *Kinect v2 -– What's New.* http://rfilkov.com/2014/05/13/kinect-v2-whats-new/ (09.06.2015)

[3] NI Mate. *NI Mate.* http://www.ni-mate.com/ (09.06.2015)

[4] Mi.Mu Ltd. *mi.mu dev-blog.* http://dev-blog.mimugloves.com/ (09.06.2015)

[5] Mi.Mu Ltd. *Mi.Mu Gloves.* http://mimugloves.com/ (29.06.2015)

[6] Blogspot. *Little Scale.* http://little-scale.blogspot.be/2014_04_01_archive.html (29.06.2015)

[7] Wikipedia. *Tait–Bryan angles.* https://en.wikipedia.org/wiki/Euler_angles#Tait.E2.80.93Bryan_angles (29.06.2015)

[8] Thalmic Labs. *Myo Blog: GUI without the G: Going Beyond the Screen with the Myo Armband.* http://developerblog.myo.com/gui-without-g-going-beyond-screen-myotm-armband/ (29.06.2015)

[9] VectorNav. *Importance of Industrial Grade Sensor Calibration.* http://www.vectornav.com/support/library/calibration (02.06.2015)

[10] Wikipedia. *Dipole*, https://en.wikipedia.org/wiki/Dipole (29.06.2015)

[11] S. Madgwick. *An efficient orientation Filter for inertial and inertial/magnetic sensor arrays.* 2010.

[12] E. Ferro, F. Potortì. *Bluetooth and Wi-Fi Wireless Protocols: A Survey and Comparison.* Institute of the National Research Council, 2005.

[13] R. Friedman, A. Kogen, Y. Krivolapov. *On Power and Throughput Tradeoffs of WiFi and Bluetooth in Smartphones.*

[14] Johnson Consulting. *Bluetooth - An Overview: How timeslots are used*, http://www.swedetrack.com/images/bluet12.htm (10.06.2015)

[15] Google Patents. *A method of compressing the header of a data packet.* https://www.google.com.na/patents/WO2007031090A1?cl=en (29.06.2015)

[16] Bluetooth SIG. *Bluetooth Master/Slave Communications and Sniff/Sniff Sub-rating Modes.* 2014.

[17] D. Camera. *Embedded Bluetooth Stack.* Undergraduate Thesis, LA Trobe University, 2011.

[18] InvenSense. *MPU-9250 Product Specification.* 2014.

[19] STMicroelectronics. *LSM9DS0 Datasheet.* 2013.

[20] Panasonic Industrial Devices Europe GmbH. *PAN13XX Core Specification.* 2014.

[21] Texas Instruments. *TI Bluetooth Stack: certified and royalty-free* http://www.ti.com/tool/tibluetoothstack-sdk relax (30.06.2015)

[22] Texas Instruments. *MSP430F5438A Mixed Signal Microcontroller Datasheet.* 2013.

[23] Future Technology Devices International Ltd. *FT232R USB UART IC Datasheet.* 2010.

[24] Texas Instruments. *MSP430 Programming Via the Bootstrap Loader (BSL).* 2010.

[25] Texas Instruments. *TXB0106: 6-Bit Bidirectional Voltage-level Translator with Auto-direciton Sensing and +/-15kV ESD Protection. Datasheet.* 2012.

[26] MindTree Ltd. *MSP430BT5190+CC2560 EtherMind Bluetooth SDK Developer's Guide.* 2010.

[27] OSRAM Opto Semiconductors GmbH. *LG L29K SMARTLED Datasheet.* 2012.

[28] Texas Instruments. *MSP430 Hardware Tools User's Guide.* 2015.

[29] Texas Instruments. *MSP430x5xx and MSP430x6xx Family User's Guide.* 2015.

[30] Future Technology Devices International Ltd. *User Guide for FTDI FT_Prog Utility.* 2011.

[31] Spectra Symbol. *Flex Sensor FS Datasheet.*

[32] International Rectifier. *IRLML6402PbF Datasheet.* 2014.

[33] ET-Tutorials. *Wozu benötigt man eine Freilaufdiode?.* http://et-tutorials.de/721/wozu-benotigt-man-eine-freilaufdiode/ (05.07.2015)

[34] Texas Instruments. *MSP430 Capacitive Single-Touch Sensor Design Guide*. 2008.

[35] Vishay Beyschlag. *MFU Series Thin Film Chip Fuses Datasheet*. 2013.

[36] Microchip Technology Inc. *MCP73831/2 Datasheet*. 2013.

[37] Texas Instruments. *TSP622XX Datasheet*. 2010.

[38] Texas Instruments Wiki. *CC256X MSP:EXP430F5438*. http://processors.wiki.ti.com/index.php/CC256X_MSP:EXP430F5438 (02.07.2015)

[39] Texas Instruments. *LP5951 Datasheet*. 2014.

[40] Texas Instruments. *MSP430 32-kHz Crystal Oscillators*. 2009.

[41] Texas Instruments. *MSPWare*. http://www.ti.com/tool/mspware (08.07.2015)

[42] Github. *hlohse: dj-glove*. http://github.com/hlohse/dj-glove (09.07.2015)

[43] Nikolaus Gebhardt. *Welcome to the Irrlicht Engine*. http://irrlicht.sourceforge.net/ (08.07.2015)

[44] Tobias Erichsen. *virtualMIDI SDK*. http://www.tobias-erichsen.de/software/virtualmidi/virtualmidi-sdk.html (08.07.2015)

[45] Texas Instruments. *MSPWare*. http://www.ti.com/tool/mspware (08.07.2015)

[46] Prof. Jeffrey Hass, Indiana University. *Introduction to Computer Music: Volume One. Chapter Three: MIDI*. http://www.indiana.edu/ emusic/etext/MIDI/chapter3_MIDI4.shtml (09.07.2015)

[47] M. Looney. *A Simple Calibration for MEMS Gyroscopes*. Analog Devices, 2010.

[48] National Oceanic and Atmospheric Administration. *Compute Earth's Magnetic Field Values*. http://www.ngdc.noaa.gov/geomag/magfield.shtml (03.06.2015)

[49] Analog Devices EngineerZone. *FAQ: Hard & Soft Iron Correction for Magnetometer Measurements*. https://ez.analog.com/docs/DOC-2544 (03.06.2015)

[50] The MathWorks MATLAB Central. *Ellipsoid fit*. https://www.mathworks.com/matlabcentral/fileexchange/24693-ellipsoid-fit (03.06.2015)

[51] YEI Technology. *3-Space Embedded Sensor*. http://www.yeitechnology.com/productdisplay/3-space-embedded-0 (19.08.2015)

# Declaration of Authorship

I warrent, that the thesis is my original work and that I have not received outside assistance. Only the sources cited have been used in this draft. Parts that are direct quotes or paraphrases are identified as such.

Signed: _____

Date: _____

I hereby grant the Heidelberg University the right to publish, reproduce and distribute my work, especially when it is to be presented to a third party for inspection.

Signed: _____

Date: _____