UNIVERSITÄT KOBLENZ · LANDAU

Fachbereich 4: Informatik

Real-time sensor-based motion capture system for virtual reality

Masterarbeit

zur Erlangung des Grades Master of Science (M.Sc.)
im Studiengang Informatik

vorgelegt von

Vera Christ

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	M.Sc. Bastian Krayer (Institut für Computervisualistik, AG Computergrafik)

Koblenz, im Juni 2017

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐

. . . .

(Ort, Datum) (Unterschrift)

Zusammenfassung

Motion Capture bezeichnet das Aufnehmen, Weiterverarbeiten und auf ein 3D Modell Übertragen von reellen Bewegungen. Nicht nur in der Film- und Spieleindustrie schafft Motion Capture heute einen nicht mehr wegzudenkende Realismus in der Bewegung von Mensch und Tier. Im Kontext der Robotik, der medizinischen Bewegunsthearpie, sowie in AR und VR wird Motion Capture extensiv genutzt. Neben den etablierten optischen Verfahren kommen aber gerade in den letzen drei Bereichen auch vermehrt alternative Systeme, die auf Intertialsystemen (IMUs) basieren zum Einsatz, da sie nicht auf externe Kameras angewiesen sind und somit den Bewegungsraum deutlich weniger beschränken.

Schnell vorranschreitender technischer Fortschritt in der Herstellung solcher IMUs, erlaubt den Bau kleiner Sensoren die am Körper getragen werden können und die Bewegung an einen Computer übertragen. Die Entwicklung in der Anwendung von Inertialsystemen auf den Bereich des Motion Capture, steckt allerdings noch in den Kinderschuhen. Probleme wie Drift können bisher nur durch zusätzliche Hardware, zur Korrektur der Daten, minimiert werden.

In der folgenden Masterarbeit wird ein IMU basiertes Motion Capture System aufgebaut. Dies umfasst den Bau der Hardware sowie die softwareseitige Verarbeitung der erhaltenen Bewegungsinformationen und deren Übertragung auf ein 3D Modell.

Abstract

Motion capture refers to the process of capturing, processing and translating real motions onto a 3D model. Not only in the movie and gaming industries, motion capture creates an indispensable realism of human and animal movement. Also in the context of robotics, medical movement therapy, as well as in AR and VR, motion capture is used extensively. In addition to the well established optical processes, especially in the last three areas, alternative systems based on inertial navigation (IMUs) are being used increasingly, because they do not rely on external cameras and thus limit the area of movement considerably less.

Fast evolving technical progress in the manufacturing of such IMUs allows building small sensors, wearable on the body which can transfer movements to a computer. The development of applying inertial systems to a motion capture context, however, is still at an early state. Problems like drift can currently only be minimized by adding additional hardware for correcting the read data.

In the following master thesis an IMU based motion capture system is designed and constructed. This contains the assembly of the hardware components as well as processing of the received movement data on the software side and their application to a 3D model.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Objective of this thesis . . . 2
- 1.3 Structure of this thesis . . . 3
- 1.4 Related Work . . . 4

2 What is Motion Capture? **6**
- 2.1 A short historical overview . . . 7
- 2.2 Optical and non-optical tracking . . . 9
- 2.3 Current state of the art . . . 11

3 Arduino hardware **13**
- 3.1 How does a microcontroller work? . . . 14
- 3.2 Used hardware . . . 15
  - 3.2.1 Arduino Pro Mini . . . 15
  - 3.2.2 IMU: MPU-6050 . . . 18
  - 3.2.3 Network: Bluetooth HC-05 . . . 21
  - 3.2.4 How to power the different components? . . . 27
  - 3.2.5 Completed Hardware . . . 29

4 IMUs **32**
- 4.1 Accelerometer and Piezoelectric Effect . . . 32
- 4.2 Physical Principle of the Gyroscope . . . 36
- 4.3 How does data fusion work? . . . 40
  - 4.3.1 Kalman and Complementary filter . . . 41
- 4.4 Quaternion math . . . 44
- 4.5 Rotational math: From Arduino to Blender . . . 45

5 Arduino code **49**
- 5.1 Code explanation . . . 49
- 5.2 The long way to calibration... . . . 52

6 Blender plug-in **55**
- 6.1 Animating a human being . . . 59

7 Results and Evaluation **61**

8 Conclusion and vision **66**

1 Introduction

Consulting a modern dictionary, we can find the following definition for motion capture: ”a process by which a device can be used to capture patterns of live movement; the data is then transmitted to a computer, where simulation software displays it applied to a virtual actor” [1].

The desire to understand human and animal locomotion and movement has been existing for centuries. It is, that only during the last decade with the development of computer science and 3D display technologies, the methods necessary for studying human motion not only in theory but also in praxis, where finally provided. However, controlling the movements of objects is a skill which has become easier, but not easy, and still requires a lot of time and effort. Especially if the animator strives for realism. Most of the time, data retrieved by the capture system still needs to undergo several fine-tuning steps to get optimal results.

Today, motion capture is an indispensable part of the film and game industry as well as for some medical faculties. The science of capture motion is a fascinating field of study due to its wide variety of applications.

1.1 Motivation

State of the art motion capture systems have one disadvantage: they are expensive. Systems used for movies require multiple high end cameras, as well as special suits for the actors to wear. In the later chapters it is explained in detail what kind of different systems for motion capture exist, how they can differ, their advantages and disadvantages and the reasons they are used for different applications. The main motivation of this work is to look into an alternative approach to motion capture, which is more cost-efficient and therefore more accessible to a wider range of people [2]. This technique is based on IMUs, inertial measurement units, which basically are small sensors which can measure the orientation of objects. These days, IMUs are used by most people on a daily basis without them even knowing. Most modern smartphones or tablets contain IMUs which determine the rotation of the device. This enables us to control characters in games by tilting the device in a certain direction (e.g. Temple Run). Becoming a mass product made the tiny chips, which are able to measure six or even nine degrees of freedom, cheap. We can now use this technology for our own means. Many builders for example use them to level their own quadcopters. Transfered to humans, these IMUs can capture the rotation of single limbs and send it to a computer to rotate the same limb of the virtual actor [3]. When combining an inertial measurement unit (IMU) with algorithms that run navigation equations, one talks about an **Inertial Navigation System (INS)**.

Another motivation for the theses is, to get a deeper understanding of character motion and to attain a level of skill in the hardware related parts of this idea, namely building the devices around the IMUs.

There exist huge databases for BVH files on the Internet, which is a file format developed by Biovision, a company specialized on motion capture. The abbreviation stands for Biovision Hierarchy and defines a bone hierarchy to which motion is applied in a standardized manner, thus sharing captured data between many 3D programs, such as Maya, Blender, Poser, Maxon Cinema 4D, 3ds Max and multiple others. Also common game engines such as Unity or Unreal Engine 4 have adopted the standard. Some of this programs also provide means to retarget the movement data to the actual skeletal mesh if, for some reason, a more complicated or perhaps also a simpler skeleton is needed. This applies for example if a normal human skeleton is not the intention, but rather a creature with, for example, wings or a tail, but where the data is still retrieved from a human actor.

Also there are some online tools (namely bvhacker) that are able to merge data from different bvh files, for example to combine a running and a jumping animation into one constant flow of movement.

Despite the fact that many motions can be realized with the use of this programs and database, not all of them can, so the need to acquire own captured data is still existed, especially in the professional area. A huge field where the real time motion capture variant is required, is virtual reality. The following chapter provides further insights into this topic.

Not only can gestures be transfered to robots for handling dangerous situations or situations to difficult to be done without a human directing the movements, motion tracking is also essential for developing autonomous robots. An agent must be able to perceive human gestures in order to interact and co-operate with them. Commercial motion capture systems produce excellent results in closed-off tracking labs, but no comparable solutions for tracking in environments not as optimal. IMU sensors provide that amount of flexibility for a cheaper prize and therefore offer an excellent field of study.

1.2 Objective of this thesis

As described in the previous section, there are many methods available to work with precaptured BVH files. Even if this would be sufficient, which it most definitely is not, at least not for high-end productions, we need to consider that this data, by definition, is precaptured, meaning it can not be used in real time.

This is where the huge advantage of capture ones own motion comes into play. Without going into detail of which system to use, controlling movement in real time is a huge advantage over the static precaptured files. The most widely known example of this probably is the Microsoft Kinect, which allows some level of motion tracking in real time. The limited range of the Kinect, might be not a problem for certain gaming scenarios, but considering that motion capture also plays a role in the virtual and augmented reality area, this may become a problem.

Therefore, the goal of this thesis is to work out a system to track motions, without the use of cameras, therefore without their restrictions concerning range, which is also cheaper than common camera based systems. With IMUs building the core part of every sensor, ten sensors, which can be attached to the human body, are build. The objective is to retrieve rotation data from the IMUs, to do some processing on the Arduino boards also contained in every sensor, and finally to send the data via Bluetooth to a nearby computer, where the values are read, brought to a form from which useful rotational information is obtained and then applied to a 3D character.

In the ideal case, the 3D character would exactly mirror the actors movement in real time. However, it is a known and at the moment unresolved issue for IMUs that they suffer from large drift over time. In this thesis, it shall be determined if this error is even relevant for shorter periods of time.

Especially for VR or AR applications this is a huge drawback of the IMU approach. Some companies are currently trying to solve this problem with the help of additional sensors, for example using optical sensors to determine the position in the environment and correcting the readings with this new information. This should not be the objective of this thesis though.

The main part here concentrates on:

- An introduction to the main software and hardware solution currently used for tracking motion, with the main focus on IMU based motion capture, and the theoretical background to back up the application side.

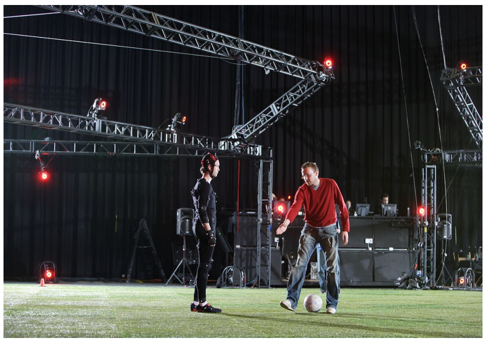
- A detailed description of how the sensors where build, why they where build this way, as well as the hardware technical background.

- The description of the software and its components which was implemented to allow the motion capture process, also containing the visualization process.

- An assessment of the received results in comparison to other techniques, some ideas for improvement, advantages and drawbacks of the IMU approach.

1.3 Structure of this thesis

The thesis is structured the following way: After some introductory words about the motivation and goals behind this thesis followed by a brief collection of current research in similar fields, the historical approach to tracking motion, the different state of the art methods and the general subdivisions of the field are explained. There are several chapters destined to explain the project itself, divided into an explanation of hardware, hardware specific


(a)

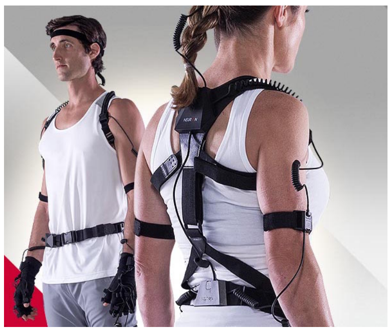

(b)

Figure 1: (a) Motion capture studio using optical methods with multiple cameras to detect the markers on the motion capture suits worn by the actors. (b) The IMU based motion capture suit from Neuronal Perception, which in contrast to (a) only uses the suit itself for tracking without any additional outer cameras [4].

code and the plug-in that has been written. To cover the theory behind the IMU measurements and the application to an object, there also is a chapter, which separately covers the mathematical and the physical concepts behind object rotation.

In the end, the final solution is evaluated and a conclusion whether the attempt is better or worse than current more frequently used approaches.

1.4 Related Work

Motion capture is a relatively young field of study, and has especially been developed for professional film projects. Since IMUs became available to the general public a few years ago, the field of IMU based tracking has started to grow.

There is a freelance like development of a system called MoCatch going on which is also based on IMUs, where the software interface comes in the form of a cinema 4D plug-in. The developer originally gave a price range of \$700 per sensor and something around \$250 for the smaller sensor versions (which can only be used in combination with larger sensors and therefore are useful to capture finger movement for example). The product was supposed to be launched in 2014, newer information is not available however [5].

There are not a lot of companies which sell IMU based motion capture systems, but it was possible to find a few startups like XSens [6], Perception Neuron [4] and Notch [7]. They all use different approaches, XSens for example uses wifi to let their sensors communicate, where Perception Neuron connects sensors with cables, see figure 1b, thus gains the advantage of being able to make them smaller. Notch uses the combination of theses two, letting the sensors communicate among each other via wifi, then transmitting the data to a smartphone using Bluetooth. The prices vary as well. The Notch system needs twelve sensors to cover the whole body, which at the moment costs \$788, Perception Neuron covers finger movements too, needing eighteen sensors for a total of \$999.

Talking about prices, one needs to see the whole spectrum of motion capture technologies to asses the prices in relation to those, even if they do not really fit inside this section and are discussed in a later chapter more in detail, therefore here are some brief numbers. Using Microsoft Kinect one can capture basic motions at home for under \$100. On the other end of the spectrum, big motion capture studios are equipped with, for example, the Optitrac system, which features eight cameras and a suit for roughly \$20.000. Only to show how high the cost can get, the price for a 96 camera system and twelve suits, allowing to capture twelve actors simultaneously, lies at \$490.000, a similar commercial studio is shown in figure 1a.

2 What is Motion Capture?

This chapter explains motion capture throughout time in all its different technologies and references what motion capture means today and which parts of it are relevant for this thesis. First, differentiate between the two terms **motion capture** and **motion tracking**, as shown in figure 2. Motion tracking includes the most difficult parts of sensing and processing the motion data, capture also includes storing the data, for example in bvh files, for later use. In principle, the data sensors or cameras provide can be stored directly, but they are seldom useful in their raw state. Resistance, currents or color pixels, whatever the sensors or cameras output, needs to be processed to movement information to be useful. All of this is included in the process of motion capture.

A short list of examples motion capture was or is still used for shows the significance of the technique:

- Gaming and interactive games: Animating walking cycles of a human figure frame by frame can be a tedious and time inefficient work. Even with the means of interpolating between some keyframes, the gait of a human person would never be imitated without people noticing the difference. Furthermore, for more complex movements designers would spend weeks animating a movement, which in the end would still not look perfect. Almost all recent games use the technique for their character movements and also other dynamic objects, e.g. the flight of a bird or helicopter can be animated by tracking a real life drone.

  Newer interactive games up to this point also feature limited forms of motion tracking.

- Movies: The arguably two most famous examples of figures which were entirely motion captured are Gollum and Smaug, both from Hollywood movies. That is what made them look so convincing in the end. While nearly all movies use motion capture of some kind, here only some recent examples are mentioned: Avatar, Thor, The Avengers, Guardians of the Galaxy.

- Robot control: Especially for humanoid robots, more complex motions are taught via motion capture, e.g. jumping, running, climbing. Even simpler task can as well be realized using captured data, for example grabbing objects. That way it is also possible for a human to steer a robot in real time.

- Sports medicine, training and medical rehabilitation: Analysis of the human gait, to help people relearn walking after accidents or illnesses has been an active field of research for a long time. The chapter 2.3 will take a closer look into this field.

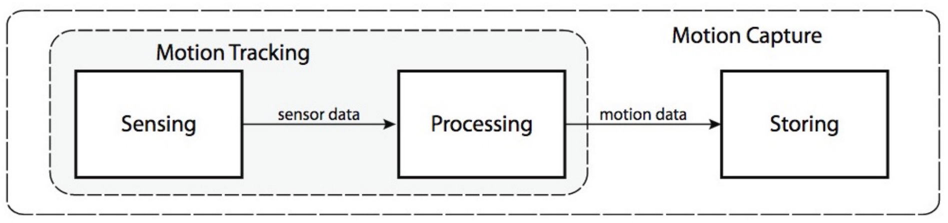


Figure 2: Motion capture generally consist of three steps: sensing, processing and storing. The first two are also called tracking [8].

In the following chapter a short review of the roots of motion capture and its development during the years is given.

2.1 A short historical overview

Seeing videos of highly futuristic robots, that developer groups all over the word currently work on, sometimes one can not tell them apart from human beings at first; an illusion dissolved as soon as the robot starts moving. Why is this the case?

Human beings perceive visual information in the following order: movement, form and color [9]. This means, we are really good at recognizing movement, because it is imperative to human survival. This makes even small children able to tell the difference if someone is moving ”right” or not, which may be an advantage in life but causes problems when humans interact with machines. The same problem occurs for every moving virtual character.

This of course was not the initial reason people started to want to understand human movement. The first approaches were made in the late nineteenth century mainly with the help of photography. Their studies, however, were for medical and military reasons.

One of the first movies which used the idea of transferring real movements to animated characters was Disney’s ”Snow White” [10]. Scenes played by real actors were filmed, and animators traced every frame to draw the animation and make their character more convincing. This technique was called **rotoscoping**, which is a procedure of drawing image sequences. It works by projecting the film image by image onto a pane of glass such that they can be copied by the drawer. As one can imagine, this is, by no means, an easy or fast process.

Around 1980 it became possible to animate characters via the computer and techniques like **rotoscoping** were started to be adapted. At the same time biomechanical labs began to become interested in the field. A professor at the Simon Fraser University attached potentiometers to actors. Potentiometers are electrical resistors, where the level of their resistance can be mechanically adjusted (equally to a volume controller for a radio). To track leg movement, he attached a sort of exoskeleton to their legs and the resistance of the potentiometer located at the knee, was turned up and down through the legs movement.

As time passed, a technique called **photogrammetry**, already known from photography, became more prominent. **Photogrammetry** is a method to recover the exact positions of surface points in 3D from video or image material, which basically means that it allows to track the path of certain reference points. That was obtained by having multiple (at least two) cameras film the same actor in a scene, having visible markers attached to him and then using the **photogrammety** algorithms to gain a 3D animation from this, which basically allowed to gather 3D information from 2D images. This was in fact the basis of optical motion capture as we know it nowadays.

The development after that was in huge parts focused on the, until then missing algorithms, that applied the captured motion to a 3D object and all sorts of post processing algorithms as well as the solution to many of the problems which occurred, when using this technique.

In the following ten years some of the first commercial optical tracking systems were introduced. For example the Selspot 3D Measurement System was one of the first commercial optical motion capture systems, using active markers (meaning the markers on the suits used LEDs to emit light) and between one and sixteen cameras, also providing its own API and programing language.

In 1988 the foundation stone to **performance capture** was laid. Performance captures nowadays meaning, is extending the basic motion capture process with additional capture methods for facial motion capture, meaning that not pure body motion, but also facial expressions can be captured. "Mike the talking head" was the first candidate for this. It was a 3D character whose facial expressions were controlled in real time by a specially build controller. The actor controls some facial parameters with this, for example the head position as well as single muscles.

In the same year, the step to the first combination of motion capture and facial motion capture in real time was archived by the company Pacific Data Images. As all optical motion capture solutions at this point were not nearly able to provide useful data in real time, they developed one of the earlier exoskeletal approaches further and used them together with the controller of "Mikte the talking head" and therefore obtained real time performance capture for the first time. This had still to be done by at least two actors, one controlling the facial expressions the other one doing body movements.

1992, taking the facial expression interface of "Mike" one step further, SimGraphics developed a new, more natural interface. Instead of the control stick for face muscular movements, they again used sensors and attached them to the face directly, covering the main areas like eyebrows, chin, lips and cheek as well as some additional sensors on a helmet. Due to the use of sensors, this was also possible in real time, such that an actor could now mimic expressions which were then roughly translated to the characters facial expressions.

Only one year later, Acclaim presented the first optical full body motion capture system, which allowed to track up to 100 markers simultaneously in real time. Up to that point of time optical motion capture had been far from real time, but with this solution one of the most important steps in the direction of modern motion capture was done. For more detailed information about the history of motion capture see [10] and [9].

2.2 Optical and non-optical tracking

Before going over the currently most intriguing research in the field, we should make a short excuse to the different kinds of motion capture, as some terms have already been used in the previous passages without much explanation.

Motion capture procedures can be divided into two categories: **optical** and **non-optical**, whereas the optical approaches can be further separated into **marker based** and **marker less** procedures. Optical solutions always belong to the **outside-in** category, which includes all kinds of systems were sensors, e.g. cameras are placed in the surrounding, whereas **inside-out** systems present systems with sensors attached only to the human actor. Many **non-optical** approaches belong to the second category, while others, like for example acoustic systems, belong to the **outside-in** category. Inertial motion capture, which will be the center of this thesis is an **inside-out** system.

An optical, markerless approach would for example be the Kinect, on the marker based side of the optical approaches the state of the art method for motion capture used in current movies and games, where actors wear markers on their bodies which are recorded by cameras. Here it is possible to either use passive or active markers, wherein active markers have the property to emit a signal, usually light.

A major problem of optical tracking systems is occlusion of markers in case they are hidden from the cameras direct view by other body parts or objects. Another huge problem is the resolution of the cameras and with that the ability to distinguish between markers that are close together. In early days, also the speed at which cameras could identify the different markers was a problem but with the technological development also in the area of camera technology this is only a minor consideration nowadays. Thirty years ago tracking twelve markers simultaneously was the maximum.

By using multiple cameras, these two main problems can be solved to some extend. Still, even today captured data is to be considered raw data and needs to undergo post processing. The study [11] compares the different kinds of motion capture.

Optical approaches with passive markers, triangulate marker position with at least two cameras. Marker paths are restored in post processing, before that there is no way of distinguishing markers from each other. A distinct advantage is that additional markers can be placed easily, which increases accuracy but leads to longer processing times. Also the space of the capture is limited to the cameras fields of view, while light conditions also play a major part in the quality of the tracking. Active markers (often LEDs) reduce post processing workload because here single markers can be identified. Otherwise the technique suffers from the same disadvantages as passive markers, while at the same time being more robust to light conditions and requiring the actor to wear batteries for the LEDs. The third optical version, markerless motion capture is an ongoing field of research. More details about algorithms for silhouette recognition and background subtraction can be found in [12].

Non-optical approaches include inertial, acoustic, mechanic and magnetic technologies. Mechanical motion capture has already been explained in the previous chapter, because it is the oldest motion capture technique. The exoskeleton has the disadvantage of restricting users movement, however.

Acoustic motion capture features ultrasonic transmitters attached to the actor and receivers for each transmitter. The distance between the receiver and transmitter gives the position of the transmitter, the system is based on the time of flight principle. The speed of sound in air is about 343 $\frac{m}{s}$, and depends on the temperature in the room, which clearly makes the system inadequate for outdoor use as it requires much manual fine tuning of variables. With one receiver only, the position of the transmitter in a sphere around the transmitter is given [8]. Adding more receivers solves the problem. Problematic is the self occlusion and the fact that self occlusion of a transmitter prolongs the way the sound has to travel to the receiver which can not be told apart from the direct way and is therefore wrongfully assumed to be the position. Contrary to what one might assume, background sound is not an issue due to the fact that this normally works in the ultrasonic range.

In magnetic motion capture, a relatively small area for movement is given. Antennas on the outside of the area generate a magnetic field over the whole area (which explains the space limitation), the devices placed on the body are a combination of three perpendicular coils, making use of induction to detect their movement inside the outer magnetic field. While occlusion is not an issue for that method, because sensors do not need a clear line-of-sight the presence of metallic objects is.

Finally, inertial motion captue: Inertial measurement is what this thesis focuses on. It presents a way of tracking motion without using cameras and thus not limiting the room for movement as for example the Kinect does. For this approach, a user has to wear IMU based sensors, which will record the movement of the body part they are attached to and send them to the computer to visualize in a 3D program. The main drawback of the sensor based motion capture however is, that the sensors tend to drift, which is explained in more detail in chapters 3 and 4.

2.3 Current state of the art

After the main distinguished areas for motion capture are clear now, approaching the most recent research in the field is subject of this chapter. Beforehand, consider the most important requirements a useful motion capture system should fulfill. Later we compare the archived results with that list to evaluate it.

- The system should not restrict the movement of the actor (e.g. tangled cables, straps too tight)
- The calibration and general handling should be relatively easy (meaning it should not take an hour to calibrate)
- The measurements should be accurate, meaning the bones of the model should match the actual actors movements. Really good systems should capture detailed and big movements equally good. Some small errors are unavoidable, however.
- The pipeline from recording the data until having a complete animation should be very tight.

Typically, augmented reality systems uses optical means like cameras to track user motion. This technique is a well proven technology which nevertheless comes with a few disadvantages such as a low range and that the area of capture needs to be equipped with markers. In the paper [13] tracking for AR with the Xsens motion capture suit has been evaluated, a means to solve those problems. There, a 3D model is integrated directly in the augmented environment.

Recently, car GPS systems are about to be upgraded by a set of different IMUs in addition to the consisting system, with IMUs having the advantage of also working in areas without GPS connection [14].

W.W. Wang and L. C. Fu designed a rehabilitation therapy for upper limbs using an exoskeleton with attached IMU sensors to the joints [15]. Other approaches use similar technologies for stroke rehabilitation and posture correction [16]. Here the IMU is attached to the patients affected limb sending its responses to a robot which mirrors the movement, either for the patients to view and reflect on his motion or for his doctors to analyze.

Various papers [17],[18], [19] present IMU applications for a sports learning context, for example measuring the acceleration of a golf club, placing a sensor inside a bowling ball to draw conclusions regarding the spinning behavior. In [20] probabilistic filters for dealing with sensor errors due to integration are proposed. Without other external correction input the position can not be tracked reliably while the posture stays accurate [11]. Various approaches like [21] therefore only track rotation with IMUs, and leave the position tracking to third party devices, in this example an ultrasonic device. This takes motion capture out of the lab and into a real life environment. Other approaches measure force detection on the ground to estimate the actors position.

Loper, Mahmood and Black, with their Motion and Shape technology [22], present a solution to a shared problem of most motion capture technologies. Whether it is sensors of markers attached to an actor, the position or orientation of those is used to animate the bones, while the sensors/markers are not really attached to the bones, but rather to the skin of an actor. These difference may seem small but in high-end contexts, they can be of relevance, especially considering that soft-tissue movement affects the sensors or markers. While those movement is treated as noise, the real movement data of the soft-tissue, which plays a not to be disdained part of a character perception, is lost. The algorithm that paper presents teaches different body forms to a model, where movement data is directly applied to the outer skin of the model without the detour through an internal bone structure.

3 Arduino hardware

Arduino is a soft- and hardware company, which started 2003 as a student project in Italy. The project was called Wiring at the beginning with the objective of making it easy for artists and designers to work with electronics, without having to understand the underlying electronic concepts.

Wiring had been based on the object oriented programming language **processing**. Processing is a language, also coming with its own IDE, that was optimized for graphics and animation by the Massachusetts Institute of Technology in Boston [23].

With this goal, an IDE with a simple editor based on the Processing IDE, was build which provided the functionality of a bootloader to easily upload sketches (which is just another word for programs) to an Atmel microcontroller, and had a serial monitor to view the output received from it. The first Wiring hardware board featured an ATmega128 microcontroller, was able to connect to the computer via USB (via a certain serial connection technology which is explained later) and had a few LEDs. By the end of 2005 Wiring was used by Universities all over the world, because it was the first viable solution which let non-engineers work with electronics in such an easy way. Back then, these boards did cost $50 which was cheaper than any other solution at that time, but still not exactly a low cost product.

In 2005 another research group took the Wire code and added support for the cheaper ATmega8 microcontroller and Arduino was born. Nowadays, the standard Arduino board is the Arduino Uno, which costs about $21 and utilizes the ATmega328. As the picture shows, there is a serial port to connect the board to a computer. The Arduino Uno, shown in 3a, is only one of the many Arduino boards available on the market, but because of its simplicity it is the board mostly used for educational purposes and in projects where the size is not relevant. There are other boards, some like the Arduino Lilypad even flexible enough to be integrated into clothing.

One may ask oneself, why Arduino was chosen for this project in the first place. It would also have been a valid approach to choose a Raspery Pi, but there are some disadvantages of the Pi with respect to this concrete project. While the standard Rasperi Pi is in the price range of $35, it is in fact more powerful than the most common Arduino Uno for $21. The advantage, however, is the wide variety of the Arduino boards, where there are smaller boards, with roughly the same functionality as the bigger Arduino Uno, for a price of only $6. So, the Raspery Pi has the much higher computing power and when juggling higher amount of data on the device, one would definitely choose it. In our case however, there were tree main advantages of the Arduino over the Rasperi Pi, which all three were crucial for the project, those being the size, the power consumption and the price.

Nowadays Arduino boards are mostly manufactured by companies like Sparkfun Electronics and Adafruit Industries, the first Arduino boards how-

(a)

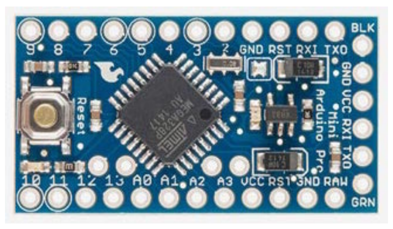

(b)

Figure 3: (a) The Arduino Uno board is the most commonly used board. Due to its simplicity and versatility it is most often used in teaching environments [24].
(b) The main difference between the Arduino Uno and the here shown Arduino Pro Mini is the seize, which may be misleading in the picture, but the Pro Mini is only a quarter of the size of the Uno [24].

ever came from a European company called Smart Projects. To name only some ideas Arduino boards can be used for: drones, data gloves, any kind of actuator controlling, home automation systems, revival of old game platforms, e-mail clients and mp3 players. For archiving this additional functionality often shields are used, which attach to all Arduino pins and provide a set of further hardware components.

Before we can continue a short introduction to the way microcontrollers work is required to understand some of the following concepts.

3.1 How does a microcontroller work?

Every microcontroller is basically a small computer on a single chip. It consists of a 8 or 16 bit microprocessor (also called CPU) as the ”brain”, a small amount of RAM storage, a programmable ROM or flash memory, parallel or serial I/O, timers, AD(analog-digital) and DA(digital-analgo) converters. Figure 4 show this.

The whole left block basically covers all I/O services. This block is used for feeding the data into the CPU. Inside the memory block, the program (here the set of instructions to be executed), and the needed data are stored. It consists of both RAM (random-acces-memory) and ROM (read-only-memory). The CPU is able to perform a limited amount of arithmetic and logic operations. The microprocessor takes the data either from the I/O devices or the memory block, and also take the instructions from memory, processes them by performing mentioned arithmetic and logic operations as required on them and hands the results back to either memory or the I/O devices [26].

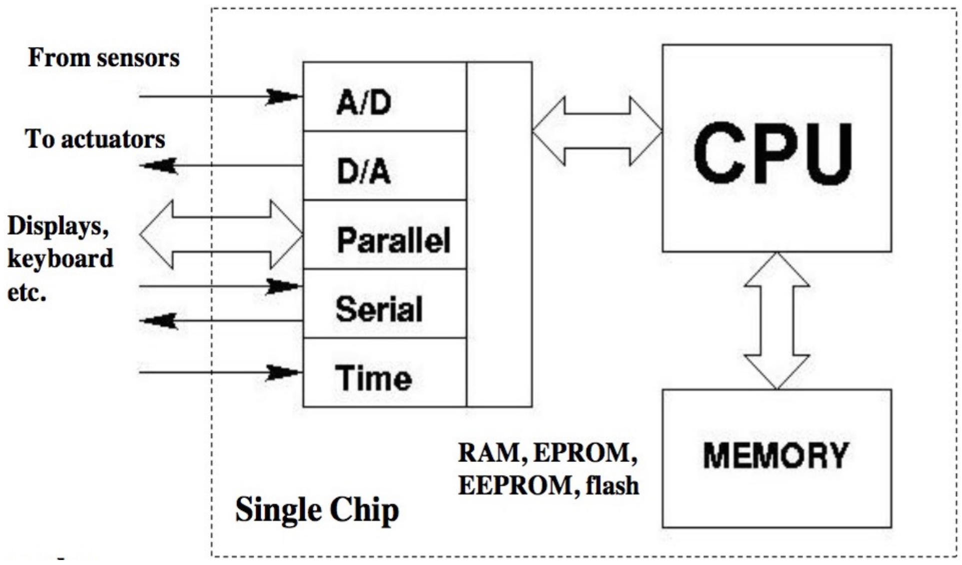


Figure 4: The structure of a general single chip microcontroller, with the three main components CPU, Memory and I/O modules [25].

This is also what happens on the Arduino microcontrollers, which mostly use the ATmega328 as their CPU. The ATmega328 is a powerful microcontroller based on the RISC (reduced instruction set computer) architecture, which means that it is optimized to decode and process instructions really quick, being able to execute powerful instructions in a single clock cycle. Compared to common CISC (complex instruction set computer) microcontrollers, through the ability to access two different registers by a single instruction in one clock cycle, it is up to ten times faster [27].

3.2 Used hardware

In the upcoming parts of chapter three the actual hardware, that was used in this project is explained in more detail.

3.2.1 Arduino Pro Mini

As already established, the Arduino board is the core of the hardware, and as the project requires a small and cheap solution, it is hardly surprising that the choice fell on the Arduino Pro Mini, shown in Figure 3b, which is about a quarter of the size of an Arduino Uno and less than a quarter of its price, while providing the same computation speed. Some of the amenities of the bigger Arduino like an easy USB host connection to the computer, had to give way to the smaller seize, however.

There are two other Arduino boards in the same price and seize category, the Nano and the Micro. The Pro Mini was chosen over them, because it is a few dollars cheaper and a little bit smaller than the Nano. The Micro is even more expensive than the Nano, and also it is as small as the Pro Mini, it is based on the Arduino Leonardo instead of the Arduino Uno and with the Uno being the most commonly used board, the availability of libraries as well as support is simply higher. The most obvious difference is, that Nano and Micro have a USB port for programming them, thus they can be programmed without any additional hardware, whereas the Pro Mini needs an FDI adapter (it can also be done with an Arduino Uno, but was not because of price considerations). One adapter can be used for all the boards to upload the sketch once.

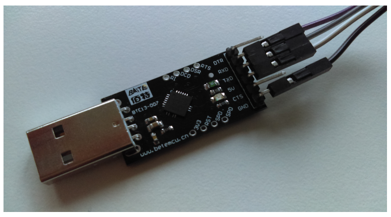
As mentioned before, the heart of this Arduino is the ATmega328 microcontroller. The six pins on the right side of the board are called the programming header, this pins allow uploading sketches to the board. In contrast to the larger boards for that an additional adapter is needed. For this the option to use a USB to TTL adapter is given and shown in figure 5a. For general clarification, here is a short list of the Arduino specific abbreviations used for labeling their pins:

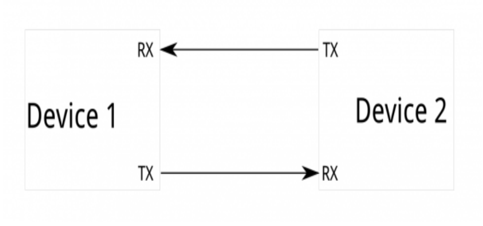
- GND: translates to ground pin which is the pin connected to the negative terminal of a battery or any other power source.
- VCC or 5V or 3.3V: pin connected to the positive terminal of a regulated 3.3V or 5V power source
- RAW: the board has a voltage divider, so it can also be operated on higher power levels up to 12V by connecting this pin instead of the VCC pin.
- RX: means a pin for receiving data through a serial line
- TX: is the corresponding pin for transmitting data through a serial line

In order to load sketches only the GND, VCC, RX and TX pins need to be connected. Available boards are either constructed for 3.3 or 5 volt usage, here 5V boards are used. Having only one serial connection interface with the pair of RX and TX pins, is one of the drawbacks of using this smaller board, there is, however, a solution to that by using a software library to change normal pins to serial pins.

Two also important pins for connecting with the IMU later are pins A4 (SDA) and A5 (SCL) which support I2C communication. I2C, Inter-integrated Circuit Protocol, allows communication of the master with multiple slave chips.

What about simply using normal serial communication (e.g. UART, RS-232) as illustrated in figure 5b? First of all, serial ports are inherently designed for communication between exactly two devices, and while that may not prevent one from connecting more chips to the line, timing this


(a)



(b)

Figure 5: (a) A USB to TTL adapter is needed to communicate between the Arduino Pro Mini and a computer, for example to upload a sketch. For the exchange of data, the VCC, GND, RX and TX pins of the adapter are connected to the corresponding pins on the programming header of the Arduino board.
(b) Serial communication between two devices. Every device sends via its TX pin and receives data on the RX pin (sometimes these pins may be labeled with RXI and TXO instead to indicate in- and output) [28].

correctly is always an issue. Apart from that, serial connection is asynchronous, making it work, therefore requires two chips agreeing on a baud rate beforehand.

So, why not use SPI (Serial Peripheral Interface)? As well as SPI, I2C is only constructed for short distance communication. However, SPI requires four communication lines, and for each additional slave there is also an additional pin on the master, needed to determine the active communication. In contrast serial and I2C only require two lines, while at the same time I2C is a stable protocol to communicate between multiple masters and slaves, thus combining the advantages of both SPI and serial communication with relatively simple hardware and a fairly simple software implementation, as shown in figure 6.

Special chips like for example IMUs (inertial measurement units), are designed to communicate using this protocol, which makes it possible to attach several IMUs for different purposes to one Arduino using only this two I2C pins. Without preempting too much, in some of the following chapters connections between sensors will be discussed further, while one of the reasons for not using I2C there is that I2C is and stays a short range protocol [28].

The Arduino Pro Mini is a board with 14 digital input/output pins, 6 analog input pins, 32kB of flash memory, an on-board resonator, a reset button, and one or two LEDs depending on the manufacturer [24]. Through a pre-installed boot loader, uploading sketches is simplified. One might also choose to use more sophisticated boot loaders like Optiboot, which are smaller, thus freeing more space for ones own sketches and also uploads those sketches faster. To burn a new boot loader onto an Arduino board, in-system-programmers (ISP), in this context for example an Arduino Uno board are used.

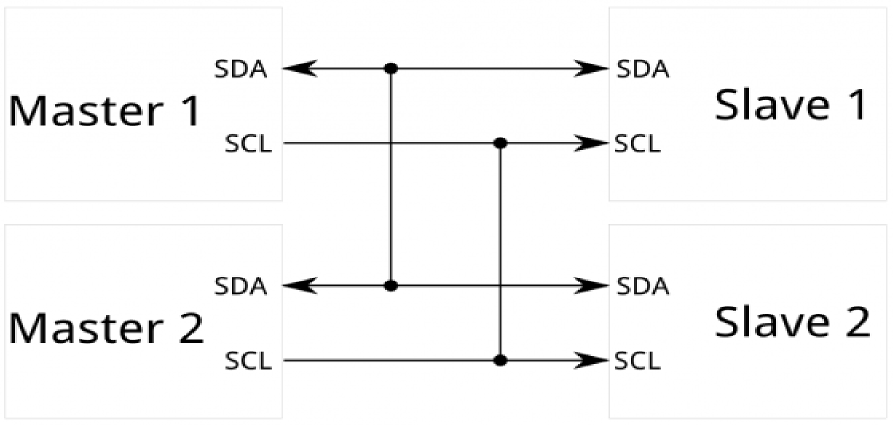


Figure 6: I2C communication between multiple masters and slaves, with I2C providing the possibility to connect multiple masters/slaves to one and the same line [28].

3.2.2 IMU: MPU-6050

An inertial measurement unit (IMU), describes a combination of inertial sensors, which can measure velocities, orientation and gravitational force. Those inertial sensors are a gyroscope and an accelerometer and sometimes even a magnetometer. Common applications of IMUs are for example the leveling of quadrocopters, determining the orientation of a smartphone or tablet as needed in many games, or in robotics, for example to level bipedal robots, or to determine a robots orientation. In smartphones some of the usages of IMUs are gesture recognition, panoramic pictures, as well as navigation and augmented reality [29]. In general, IMUs are employed whenever the orientation of a device is needed for further calculations. In older IMU designs, all the sensors were attached to a single stable platform detached from outside rotation, which had the disadvantage of being mechanically complex and also sensitive [30].

An IMU with six degrees of freedom has, at its disposal tree gyroscopes for measuring rotation along the x-, y- and z- axis, returning three angular velocities and tree accelerometers measuring acceleration for all three axis, which return linear acceleration values. Figure 7a visualizes the associated coordinate system. Sensors of the same kind are placed orthogonal towards each other. By processing the sensor data, orientation and position of a device can be received. IMUs with nine degrees of freedom have an additional triple pack of magnetometer sensors, others have an additional altimeter. Magnetometers measure magnetic fields and due to the strong earth magnetic field can also be used as a compass. In the context of IMUs, magnetometers are used to improve the signals of the accelerometer, especially to put the relative values from the accelerometer in an absolute context. For this thesis, it was decided against using an additional magnetometer in order to make the sensors as cheap as possible. Altimeters on the other hand only obtain hight information and are therefore not relevant for motion capture applications [31].

Only due to the recently fast development of MEMS devices it became possible to manufacture IMUs at the size of less than a centimeter, which allowed the usage in space critical devices like smartphones. MEMS, standing for micro-electro-mechanical-system, is a technology that allows for mechanical systems (such as gyroscopes and accelerometers) and electronic systems (such as general circuits) to be miniaturized, using microfabrication techniques. The most notable parts of the MEMS technology are microsensors and microactuators, both of those are characterized as devices, which convert energy to another form, in this case, mechanical energy into an electrical signal. Additional to microsensors and microactuators, the family of MEMS is supplemented by microelectronics and microstructures. Some of the most common MEMS technology sensors are sensors for pressure, temperature, radiation, magnetic fields and inertial force. Not only are MEMS sensors cheap in production, something else is remarkable and surprising about them. Some of those sensors, for example for temperature measurement, outperform their larger counterparts, making them an extremely useful, cheap and accurate technology. The real benefit of the MEMS technology shows itself when the sensors and actuator can be merged into the process of nanotechnical fabrication of integrated circuits, making their output available too other microelectronic components [32].

For this project the six degrees of freedom MPU-6050 sensor from InvenSense shown in figure 7b, which contains a MEMS gyroscope and a MEMS accelerometer in a single chip, is used. Furthermore, the chip comes with a digital motion processor (DMP). However, the real MPU-6050 sensor is only the 4x4x1mm big black box. The internal structure of this small sensor is depicted by figure 8. In total there are six 16-bit analog-to-digital converters on this board to convert the three gyroscope and the three accelerometer values. After combination of this six values plus a temperature (only for monitoring temperature fluctuations, which may result in disturbing the inertial sensor readings), they are feed to the sensor registration unit, which always holds the latest of these values and may be accessed any time via the serial interface [29]. These readings are also stored in the FIFO, which is accessible over the serial interface. From there data blocks can be read in burst, thus enabling lower power consumption between burst reads. It is therefore highly recommended to read data only from FIFO and not directly from the sensor data register. Furthermore, the Bias&LOD unit provides the right current for each sensor, whereby the gyroscope needs a higher voltage and therefore is supplied by separate unit called charge pump which generates high enough voltages. The DMP is a processor on its own with the

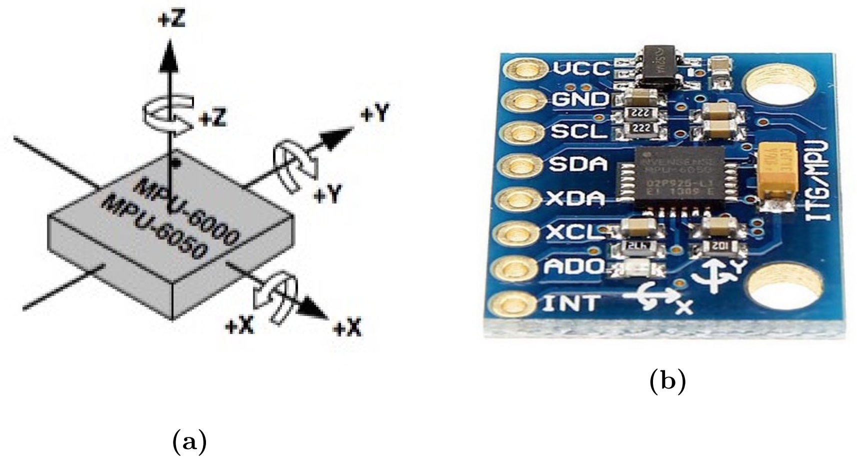


Figure 7: (a) The MPU-6050 has its three gyroscopes and three accelerometers arranged orthogonal to each other on the three axis. This chart shows how the axes are positioned from the perspective of the chip, and will be important later to fit the axis to the ones of the application [29].
(b) MPU-6050 chip integrated in the GY-521 module produced by Sparkfun [24].

single purpose to offload workload from the host processor (here the Arduino micorprocessor). To run motion tracking algorithms with high enough accuracy a data rate of at least 200Hz is necessary even if the application can not provide such a high rate. Somewhat problematic is the fact, that InvenSense does not provide any detailed information about the exact computations the DMP performs [33]. What is available, is that the DMP combines the gyroscope and accelerator measurements to gain an orientation. Alone each of the sensors is prone to errors, but combined together they balance each other. The exact methods these sensors work with, as well as the causes for errors that occur, are discussed in chapters 4.1 and 4.2. The result of the DMPs calculations is stored in quaternions. Comparisons show, that the values obtained from the DMP are extremely accurate and surpass manually read data from the sensor data registration unit [34]. That is why, for this project the data was read from the DMP, but more about that will follow in the chapters about software.

As the picture of the sensor shows, it is embedded in a larger board. There are some different breakout boards using the MPU-6050 chip, for example the popular Sparkfun boards or as used here, the GY-521 [24] which is the most commonly used IMU board. In addition to the MPU-6050 sensor, the complete breadboard comes with resistors, which guarantee a maximum voltage of 3.3 V and LEDs, which each consume 1.5 mA to show the activity state of the module. For professional, high-end application those are useless components, in such a case one would only use the MPU-6050 chip (which

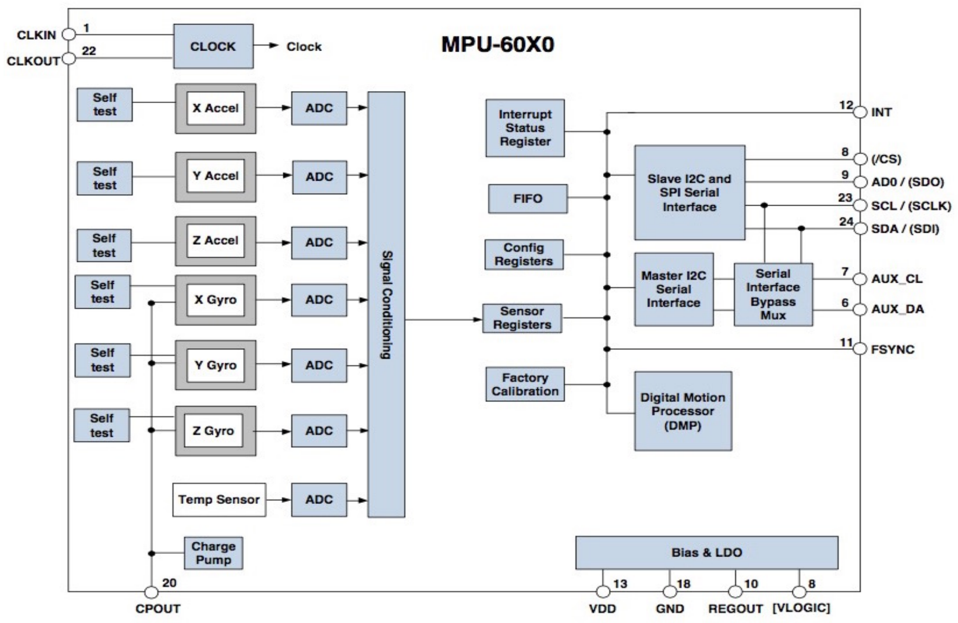


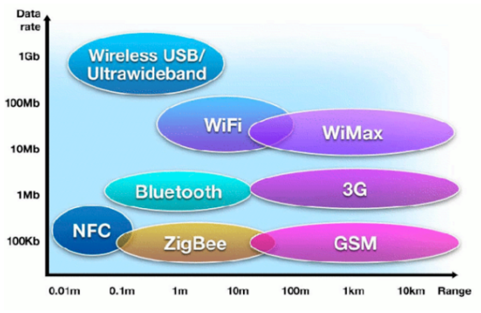
Figure 8: This block diagram shows the internal structure of the MEMS IMU MPU-6050 [29].

requires specialized labs though). Another possibility to save some energy is to short circuit the regulators and desolder the LEDs, but with that, one single mistake might destroy the board.

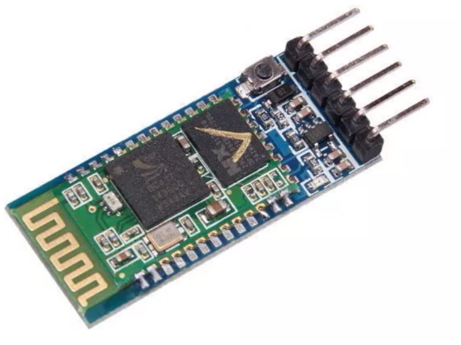
The integrated I2C data bus allows to directly connect to another 3-axis sensor, making it possible to capture more information. I2C is a multiple-slave, multiple-master, single-ended serial bus which here uses two wires, the SCL for clock values and the SDA for data transmission. Later the i2cdevlib and the Wire library will be used to model the connection via the I2C interface to the Arduino in software. To be conform with the libraries requirements, the SCL pin is connected to the A5 pin on the Arduino and the SDA pin to the A4 pin. What is more, the MPU always acts as a slave to the Arduino over the I2C bus. I2C communication is performed at 400 kHz [29].

3.2.3 Network: Bluetooth HC-05

Previously, the versatility and almost unrestricted freedom IMU system provides were explored. As a means to really provide this kind of freedom, the usage of cables, which could and would restrict the movements of an actor were attempted to be held to a minimum. Regardless of that, during this chapter it becomes clear, that with the technical and financial means of this thesis there was no way to completely do without cables. Still, at the be-



(a)



(b)

Figure 9: (a) The diagram shows the different network and communication protocols available, at the x-axis the possible transmission range in kilometers, at the y-axis the data rate [35].
(b) The HC-05 Bluetooth breakout board, next to the pins, there is an LED for showing the status as well as the 3.3 to 5V regulator and key button which forces programming mode if high before power is applied. The golden stripe on the far end of the board is the RF antenna, the two black chips are the CSR Bluetooth radio chip and a 5MB flash memory.

ginning a network protocol was sought to replace cable connections, if not between all sensors, then at least between the sensors and the computer to provide the freedom to move unrestricted inside a room.

Getting all the captured data from the Arduino board to the computer wirelessly requires the use of one or another wireless protocol like shown in figure 9a. Only three of those were shortlisted for this project: Wifi, Bluetooth and ZigBee.

All three protocols use the same frequency between 2.4 and 2.5 GHz.

ZigBee (IEEE 802.15.4) with 250KB/s has a relatively low data rate, while on the other side having the advantage of a low energy consumption and a low probability for collisions. This protocol was designed for the transfer of small data volumes. With regards to architecture it is the easiest of this three protocols, designed to work with small battery powered sensors over years. A technology, known as Low Rate Wireless Personal Area Network, which was developed especially for sensor-actor models, makes it possible to work extremely energy efficient, what is more in this networks, devices tend to be inactive most of the time which in return again saves energy. As well as with Bluetooth, participants are connected in a star or a tree topology. ZigBee presents its strength when the participants in a network need to talk to many other participant like it is the case in sensor networks. This shows, ZigBee with its focus on sporadic data transmissions and a relatively low data rate is out of the race because that is exactly the opposite of what we want to achieve. We have sensors sending a constant data stream over a relatively short time (compared to sensors which can run for years). Also, ZigBee is an expensive variant [36].

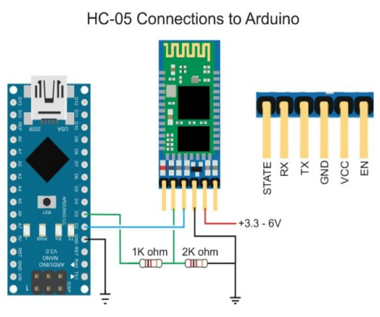
Wifi alias IEEE 802.11, has a way higher data rate (up to 54000KB/s), but also a high power consumption as well as collision probability. In general, wifi chips are used, when the need to directly connect to the Internet is arises. The data sheets of those chips state a typical power consumption of 170 mA. Going one step ahead, such a power consumption would mean a runtime of only five hours with the chosen battery and that only for powering the wifi module, not considering that also the Arduino microcontroller as well as the IMU need to be powered. This, and the relatively high prices for those chips clearly kick wifi out of the list too, at least for now. This approach is revisited later on [37].

The last option, Bluetooth is a short-range radio technology to connect electric devices wirelessly. Bluetooth is the IEEE standard 802.15.1, with a data rate of 720KB/s, a relatively low data consumption and a very low collision probability. This means, that the transmission rate does not even come close to wifi, but keep in mind that when streaming music via Bluetooth is a common practice, this will also suffice for sensing our data stream. Bluetooth also is a one-to-one ad-hoc communication protocol, meaning it forms a star topology where in one master connects all slaves and the slaves are not connected among each other. It therefore only features communication between two devices. The typically most used board is the HC-05 board. With a power consumption of 20 mA when paired (40 mA while pairing, 2 mA in sleep mode), the theoretical time to power this module with the same battery as before, goes up to 44 hours (again not counting the micro controller and IMU). Bluetooth was designed to be a robust protocol, thus being fully functional in very noisy environments. All of this makes Bluetooth the best candidate for this purpose. One might ask oneself why not to use the new Bluetooth Low Energy standard, which came with Bluetooth 4.0. The answer is, that not all computers support it yet, so do none of the cheaper smartphones. Also the common (and cheap) HC-05 and HC-06 boards do not support it and for the hand full of boards which do, not much information or trouble shooting data is available online.

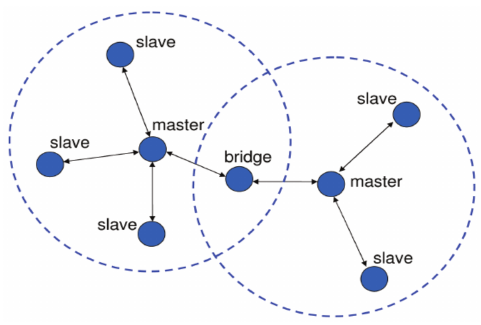
After this decision being made, let us take a look a the HC-05 module like the one shown in figure 9b.

The board uses the relatively old Bluetooh 2.0 standard, can function as either master or slave and uses the 2.4 GHz ISM band. As Bluetooth is based on Frequency Hopping Spread Spectrum techniques, which basically means that the operating frequency is changed on a regular basis, thus becoming less error-prone to interrupts from the outside, this module uses the Gaussian based frequency shifting GFSK (Gaussian Frequency Shift Keying).

The STATE pin is an interrupt pin. Some modules have an additional pin named KEY. On other modules the same pin is named EN. It is used as a reset, which is needed to get the module into programming mode. Program-



(a)



(b)

Figure 10: (a) An Arduino connected to a HC-05 Bluetooth modules. Two resistors act as a voltage divider to drop the voltage down low enough that the HC-05 does not get damaged when provided with more than 3.3V over the RX pin.
(b) A scatternet, formed by two piconets which each have a master and up to seven slaves [38].

ming it needs to be done separately, here the TTL to USB module comes in handy again. By connecting the corresponding pins one is able to connect the module to the computer, by holding the reset button on the KEY/EN pin down, while plunging in the USB. Indicated by an LED blinking every two seconds, the programing mode is active. For that the serial console of the Arduino IDE has to be set to a Baud rate of 38400, as stated in the data sheet. Using the specified commands, one can customize the name the module should appear under, a safer password, and very importantly the baud rate the module should be sending with. Supported baud rates range from 9600 to 460800. For us a baud rate of 115200, as a stable, yet fast transmission was chosen. Stop and parity bits were both set to zero, because this brought the best results while testing. Other Baud rates were tested too, but the common consent is that anything above 115200 is not reliable and baud rates below 115200, for example 9600, 19200 or 38400 proved to be too slow to transmit the data stream correctly. Missing or twisted bits were the consequence.

Being done with setting the module up, one can connect it to the Arduino breakout board via the RX and TX. The right way to connect the two modules is shown in figure 10a. Also the lines VCC and GND are connected to the power source. Reading the specification of the HC-05 leads to the conclusion, that there is no direct way of connecting a 5 V Arduino board to the Bluetooth module, because even though the module allows voltages from 3.3 V up to 5 V applied to its VCC pin, the same does not hold for the receiving RX pin. How do we drop the voltage from 5 V to 3.3 V?

A voltage divider is the solution to this problem. It is a circuit which opts to turn a larger voltage into a smaller one. The corresponding equation is as follows:

$$V_{out} = V_{in} \cdot \frac{R_2}{R_1 + R_2}$$

This means, that the output voltage is proportional to the input voltage and the ratio of R_1 and R_2. This equation can be directly derived from Ohms' Law. Figure 11a and figure 11b both show the same voltage divider with a slightly different form of notation. The left one seems needlessly complicated, but in fact it is the more accurate depiction, the one we can use to explain the derivation of the above formula, the right one, however, is closer to the real case scenario, the way this is operated in the project.

For starters, let us take Ohms' Law and for once simplify the two resistors to one and ignore V_{out}:

$$\begin{aligned} V_{in} &= R \cdot I \\ V_{in} &= (R_1 + R_2) \cdot I \\ V_{in} &= (R_1 + R_2) \cdot \frac{V_{out}}{R_2} \\ V_{out} &= V_{in} \cdot \frac{R_2}{R_1 + R_2} \end{aligned}$$

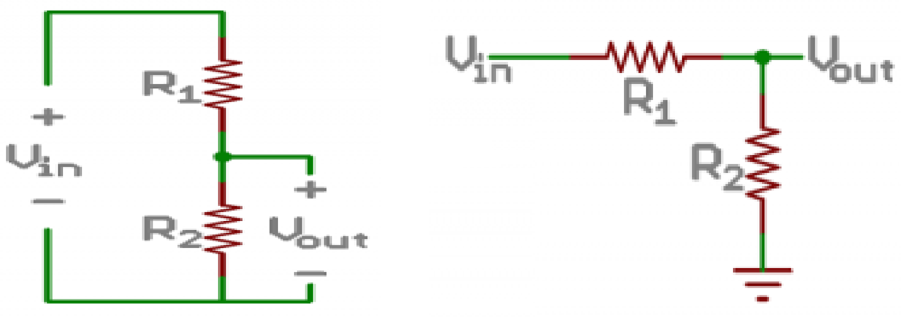
where I is the current for both circuits. From step one to step two the previous simplification $R = R_1 + R_2$ is abolished. The next step plugs in Ohms' Law for the second circuit with $V_{out} = R_2 \cdot I$, which leads to the voltage divider equation after one transformation step [39].

So, having explained this, a voltage divider is clearly a solution here. Therefore we use two resistors connected to the RX line ($1k\Omega$ and $2k\Omega$). Keep in mind that the goal was to drop the voltage form 5V to 3.3V so we have $R_1 = 1k\Omega$ and $R_2 = 2k\Omega$ and not the other way round. Inserting this values in the above equation we get $V_{out} = 3.3V$ as expected. When switching R_1 and R_2 by mistake, the equation results in $V_{in} = 1.67$, which is not enough to run the module reliably.

However good and reliable Bluetooth proved to be until here, there are some major disadvantages to the technology: Bluetooth forms out so-called piconets, which are groups of eight active participating devices. Also there may be up to 248 additional inactive devices paired too, but only eight can actively communicate at a time. From this eight devices one device has to be the master, which is in contact with up to seven slaves. Since the random frequency changes, which take place 1600 times a second are controlled by the master, many piconets can be established in close proximity to one another, because their frequencies are highly unlikely to overlap. Interconnected piconets form a scatternet.

The apple support website however literally states:

> The official Bluetooth specifications say seven is the maximum number of Bluetooth devices that can be connected to your Mac at once. However, three to four devices is a practical limit, depending on the types of devices used [40].



(a) (b)

Figure 11: (a) This circuit show a voltage divider to turn a larger voltage into a smaller one with the help of two resistors.
(b) A depiction of the same voltage divider as in (a), but in a more context oriented notation closer to the real circuit but not as intuitive for calculations as the circuit in (a) [39].

This quote is important, because at the beginning the idea was to have the ten sensors connected to the computer via Bluetooth. As that is not possible given this Bluetooth specification, because we need more than seven sensors, another solution had to be found, which will be explained in chapter 5. This solution connects each two sensors together to reduce the number of Bluetooth connections from ten to five. The test also confirmed apples statement that three or four connections are realistic. So, the connections had to be decimated drastically.

When this problem occurred another solution was discussed but later discarded: There is a protocol which builds up on some ZigBee standards and at the same time provides cheap breakout boards and higher data rates. RF24 networks. The use of the RF24L01 board, which uses that technology was considered extensively. It is also accessible via the SPI interface and has its benefits in a very low power consumption. The real difference in regards to Bluetooth is that it allows communication between multiple devices, hence it lets six Arduinos talk to a primary Arduino which then might forward the received data to another device. On the one hand this would make a huge improvement, because in software there would not be the need to deal with multiple serial connections, while on the other hand data from a few sensors would be read correctly, but the 250Khz up to 1Mhz of the RF24L01 is not high enough to read high amounts of data from six incoming channels [41].

In the end, the fact that for using this module, one would not only need to build the transmitters but also an additional receiver to attach to the computer, because a standard computer is not equipped with such a receiver, was vital for deciding against this solution. What is more the technology also holds no true advantage over Bluetooth due to as well limiting the amount of nodes that can be active in the same network, to six transmitters. For a working solution two receivers (one in case two transmitters are connected together) in addition to the ten transmitters would have been needed to be build.

While only three Bluetooth connections worked properly as predicted by apples statement but at least five were needed (if every two sensors would send over one Bluetooth module), some other solutions were considered. In this process connecting more than two sensors together (even though sacrificing some freedom of movement) was a viable option but was discarded for software technical reasons. This is explained in chapter 5. Among further ideas there also was a rather natural approach, that if the problem was having enough sensors working in the same piconet, why not create a second piconet? Yet, the need for a second computer or other receiver lead to dismissing the idea.

Further approaches included a relatively new board. The ESP8266, belonging to the wifi family gained some popularity in the last two years. Due to its focus on connecting small microcontroller devices to the Internet, for example to manipulate mail servers, this board was a mere try at first. It is the first affordable wifi board for only around $9 and also has a lower power consumption than any previous board. There even exist some approaches to lower the, in relation, still high power consumption, see [42]. Nevertheless, it is to be mentioned that when using this module it will be at the expense of the battery and will reduce the operating period significantly. For the sake of completeness, one of those modules was tested with a spare sensor, the results however showed that the module is, at least until now, not reliable enough to transmit a correct data stream in real time.

3.2.4 How to power the different components?

Every sensor we build needs its own power supply. Let us first add up how much energy is needed by the single modules:

The Arduino Pro Mini uses 5V and runs on 16MHz, the specification states that power level between 3.3V and 12V are suitable.

Unregulated voltages higher than 5V need to be applied to the RAW pin because it has an attached regulator to drop the voltage. However, a common way to power this module is to use a 9V AA battery. When using the RAW pin, considering that it always drops the voltage, it needs to receive at least 5.4V to successfully use it for a 5V board. Furthermore, the energy lost when dropping the voltage needs to go somewhere, making the voltage difference transform into heat. A high voltage is therefore not recommended in order not to overhead the board.

When applying powers between 3.3V and 5V, the VCC pin is the right choice. The power applied here does not undergo any transformation and is the same power which will be available at all pins.

The microcontroller has an approximate consumption of 15mA when transmitting.

The MPU-6050 only needs a current of in total 3.9mA (3.6mA for the gyroscope, and 300μA for the accelerometer). To power this module it is directly connected to the Arduino. Because it is a 3.3V module, connecting it to the 5V Arduino would be a problem which was solved by choosing a battery with 3.7V, eliminating the need to add an additional voltage divider to drop the power here.

For the HC-05 Bluetooth transmitter, voltages between 3.3V and 5V can be applied to the VCC pin. The usage is described to be between 20mA when connected and 40mA when connecting or sending.

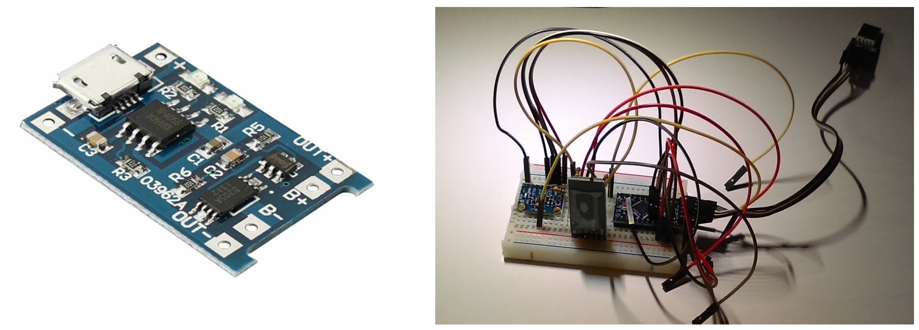
This leaves us with a summation of approximately 60mA.

As a fist idea, because of the goal to make the sensors as small as possible, button batteries were considered. Either cheap alkali-manganese or more sustainable lithium-ion batteries. Common lithium button batteries have an output of 3V and a capacity of 230mAh. 3V is not enough to power all our components, so at least two batteries are necessary which would then provide a capacity of 260mAh. On the one hand this is not really much capacity to work with and on the other hand, it is a small solution. The problem is, those are normal batteries and not rechargeable ones. Rechargeable button batteries tend to be really expensive and also have even less capacity.

So, the alternative of course is a standard rechargeable Lithium-Ion battery. This kind of battery has the advantage of being very power dense. The selected battery is a Li-Polymer type battery with 3.7V and a capacity of 1250mAh, having the disadvantage of being larger than a button battery, while on the other hand smaller than a standard AA battery with the same capacity, so it provides enough capacity to run all the modules for several hours. Theoretically at 60mA usage this leaves us with an operating period of 20 hours, which is even more than really necessary considering that batteries could be charged over night.

Lithium-Ion (also all other Lithium-Polymer) batteries are charged with a procedure called **constant current constant voltage (CCCV)**. At the beginning a battery is charged as one would expect with a set, constant current, up to 70 or 80 % which equals a predefined voltage. The higher the current, the faster the charging. Once this is reached, the charging changes to use a constant voltage. Why? If charging would continue at a constant current, the chances of overcharging would be high, because the predefined voltage would be exceeded, which would destroy the battery. A way to bypass this problem and at the same time continue charging and not to stop at 70 or 80% of the possible capacity, is to switch to a constant voltage the moment this point is reached, so it can't exceed the predefined voltage. While charging with that voltage, the higher the charge status of the battery, the lower the current drops all by itself, up to the point were it falls below a defined termination value [43].

On the one hand modern lithium batteries have a couple of advantages,



(a) (b)

Figure 12: (a) A lithium battery charger and protection module to protect the battery from being over- or undercharged. It also provides an easy interface to charge the battery over a common USB to MicoUSB cable [43].
(b) This photo shows the first draft of a sensor. Still with a lot of cables and using the TTL adapter as a power supply.

the most important ones being that storing a lot of energy at a relatively small size and reloading efficiently without negative effects makes them the first choice for portable electronics such as smartphones, notebooks and cameras. On the other hand handling this batteries requires some sensitivity as they are likely to explode, when not handled correctly. This can happen if overcharged or if the current drops too low, so the charging rate needs to be limited. It is therefore important to have separate protection modules, which may look like 12a. So, even the special CCCV charging method is not enough to protect the battery cells sufficiently. The protection circuit will primarily make sure that the battery is not charged higher than its maximum safe voltage, or discharged lower than the minimum safety voltage. It also ensures that it is not charged with a too high current and that not too much current is drawn from the battery [44].

3.2.5 Completed Hardware

As a first try, the components were pinned to a breadboard and connected with jumper wire. The picture 12b shows the first working module, still powered from the computer by using the TTL adapter only for power, not for transmission.

The final version, with all wires soldered and all modules glued inside a box the size of a palm looks like picture 13. On the left hand side a switch to turn it of and on manually, the battery and the battery protection charger are located. On the left hand side from top to bottom are the HC-05 Bluetooth module, the IMU MPU-6050, the Arduino Pro Mini and the necessary resistors. The four cables leading from VCC, GND, 10 and 11 pins

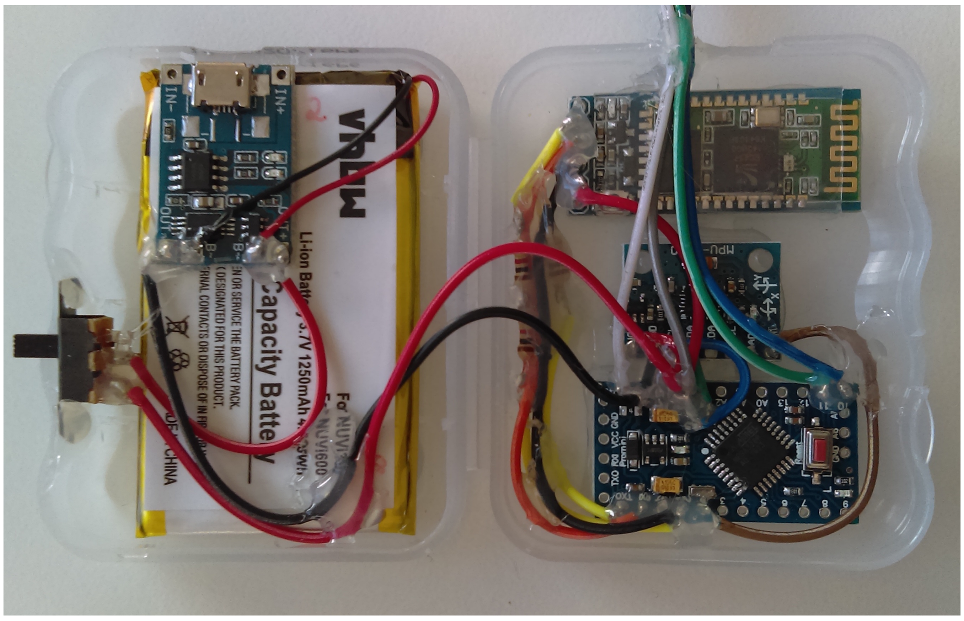


Figure 13: This is the final version of the sensor. Ten of those were build for the project.

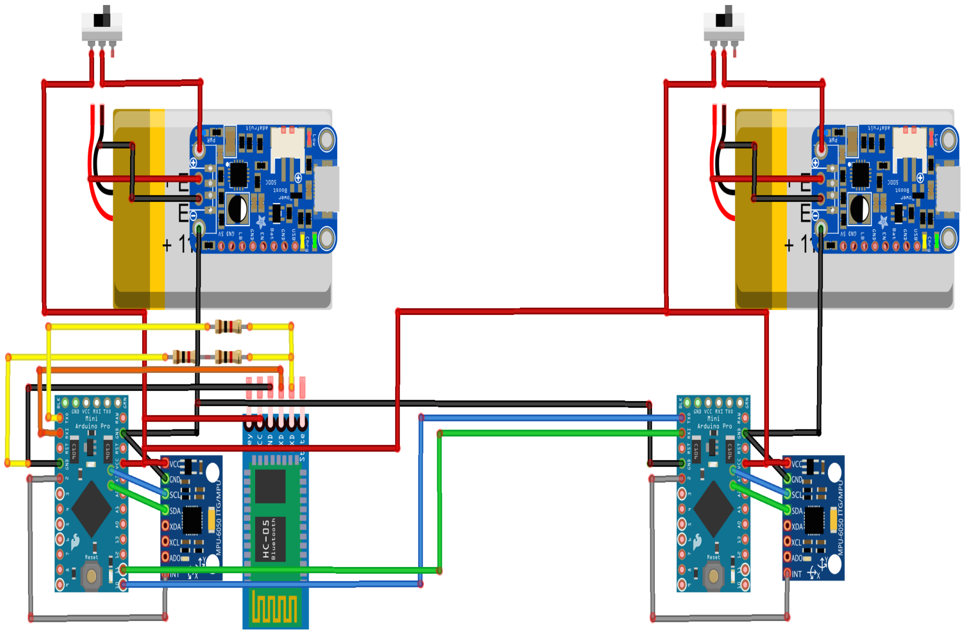


Figure 14: A sketch of the circuit, done with the electronics program Fritzing.

outside are the connection to the second sensor which is essentially build up the same way, but without the Bluetooth module and the green and blue cable from this sensor connected to its RX and TX pins. The corresponding circuit diagram is shown in figure 14.

Finally, having completed the hardware part of the project, let us move on to the the math behind all of this.

4 IMUs

Before starting to write actual code, we start with some theory. As already stated, the goal is to get rotation data from the IMU and transfer those to a 3D Object.

Therefore in this chapter, the whole process of attaining these data is covered, starting from the physical principle of accelerometers and gyroscopes, ending with the math necessary to calculate rotation as quaternions.

IMU sensors are used to get the altitude of the object they are attached to, for example self balancing robots, game controllers, or smartphones. These devices normally use gyroscopes and accelerometers to detect the orientation in a three dimensional space. Separately however, both are prone to errors as they are not if the results are combined into one angle. So, let us have a look at the way both of these sensors work and at the underlying physical principles and after that at how the data is combined.

4.1 Accelerometer and Piezoelectric Effect

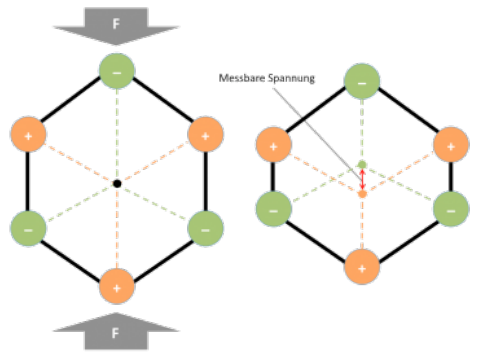
When an isolator is brought inside an electric field $\vec{E}$ the charges inside the material starts shifting and a mechanic deformation is the consequence. This effect is called **electrostriction**. The reverse effect of this is that a mechanic deformation can make the charges shift in a way that an electric field is generated. That phenomena is called **piezoelectric effect** and can only happen with special materials such as quartz, barium titanate and tourmaline. Quartz and tourmaline have, due to their internal structure, only a relatively small piezoelectric effect. The effect is higher for barium titanate. For the common MEMS sensors, ferroelectric ceramics are used, which provide an even higher piezoelectric effect [45]. Figure 15a shows a section of a crystalline grid, which gets deformed when under pressure. The effect will only happen when the crystal has at least one not mirror symmetric axis. The shift of charges leads to a measurable voltage.

Easily spoken, putting pressure on the material generates charges on the surface, thus converting mechanical to electrical energy. This electric field can be tapped of as a voltage.

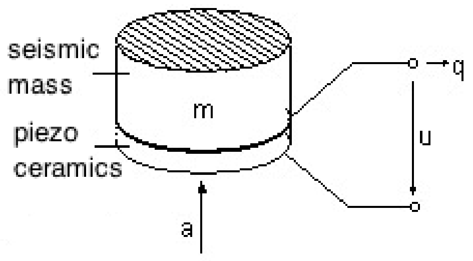
A piezoelectric acceleration sensor consists of a seismic mass on top of the piezo ceramic plate. Figure 15b shows the setup. When undergoing an acceleration the heavier seismic mass is pressure onto the piezo ceramics, induces a shift of charges and leads to a measurable voltage difference. This sensor works only for the axis it is oriented at, for each additional axis to measure acceleration on, another sensor of that kind is needed.

A more schematic way of imagining an accelerometer is to think of a ball in a box. This is not what a MEMS accelerometer looks like in reality, but it helps with understanding the sensor.

In a vacuum the ball in the box will simply float in the middle of the box

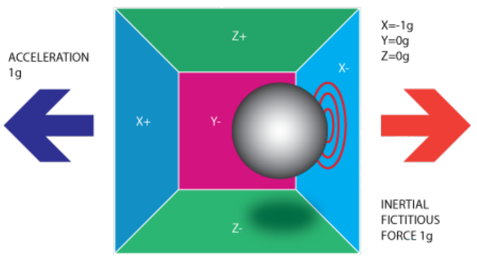


(a)

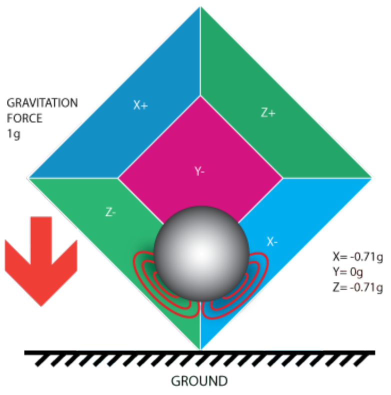


(b)

Figure 15: (a) The atomic structure of a crystal which gets deformed by outer pressure shows what happens in the crystal. A voltage becomes measurable [46].
(b) A schemata of the piezoelectric effect, with a seismic mass sitting on top of the peizo ceramic element, which is connected to a circuit to tap of the voltage [47].



(a)



(b)

Figure 16: (a) and (b) are meant to visualize the way an accelerometer works by imagining a ball inside a box which moves in accordance with the force applied to the box from the outside. In (a) the box moves to the left, so the ball puts pressure on the right wall, in (b) the box is tilted by 45° and the pressure due to gravity is the same on two walls. [48].

as long as no force is applied. When applying force by moving the box to one side, the ball touches one of the walls with a certain pressure. The walls of the box are pressure sensitive and the pressure force can be measured. In case of figure 16a, the box is moved to the left, so the ball touches the right wall which equals the x direction. By moving the box with an acceleration of 1g, which equals the earth acceleration, the reading is x=-1g, y=0, z=0. This also shows that acceleration is only measured indirectly through pressure (or force) in an accelerometer. Transporting this box image from outer space to a place on earth, the ball will lie on the bottom side of the box applying pressure to it while the box is in rest and there should be no pressure on either side of it. For that reason when calibrating the IMU in a level position, the reading is still 1g on the z axis, while 0g on the others, which needs to be taken care of in further calculation. Returning to the box example, a box can also be tilted in a certain direction like shown in figure 16b. Now a pressure of $\sqrt{\frac{1}{2}}$ is applied to two of the walls. The figure 17a shows how the axes are oriented according to the box (same colors), F is the initial force vector which in this case is the gravity vector. This value results from the use of the Pythagorean theorem in 3D which is:

$$|\vec{F}|^2 = |\vec{Fx}|^2 + |\vec{Fy}|^2 + |\vec{Fz}|^2$$

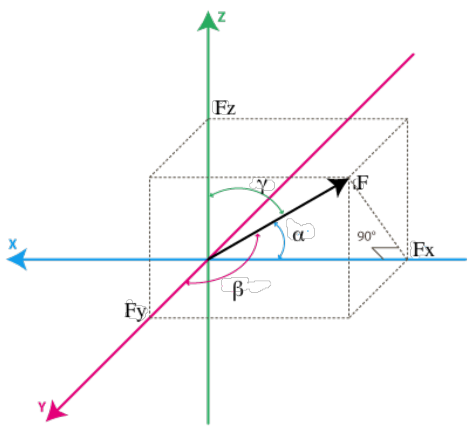
In this example, $\vec{F}$ is the vector pulling down, parallel to the earth gravity vector. Due to earth gravitation, a force of 1g should be the sum of forces applied to the two axis, in the picture the rotation is 45°, so half of it acts in the direction on the x and half in the direction of the z plane. This means the length of $\vec{F}$ is 1g and the length of $\vec{Fx}$ and $\vec{Fz}$ need to be equal, as well as $\vec{Fy}$ is zero because there is no part of the force in that direction.

$$\begin{aligned} 1^2 &= |s \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \vec{|}^2 + 0 + |s \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} |^2 \\ 1 &= s^2 + s^2 \\ \sqrt{\frac{1}{2}} &= s \end{aligned}$$

This result shows that $x = z = -\sqrt{\frac{1}{2}}g$, where the minus only indicated the direction of the force.

To calculate the inclination of the device, the angles between the force vector F and the coordinate system axes are needed as shown in figure 17a and can be obtained by:

$$cos(\alpha) = \frac{\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}}{\vec{F}}, \qquad cos(\beta) = \frac{\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}}{\vec{F}}, \qquad cos(\gamma) = \frac{\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}}{\vec{F}}$$



(a)

(b)

Figure 17: (a) Axes of the accelerometer, where F shows the initial force vector, which may be, for example the earth gravitation or an acceleration [48]. The angle from this vector to the coordinate axis is the rotation about that axis.
(b) A classic gyroscope produced in 1917 [49].

After having determined some of the different states that occur, we take a look at how to get from the internal hard- and software of the MPU to usable values and from there to rotation information, when the only things an accelerometer measures are forces. For example, let us say the accelerometer force measurements for the force vector F are: $x = 0.51g, y = 0.43, z = 0.26g$. For a sensitivity value of the sensor we assume $sens = 300mV/g$, the voltage is $U = 3.3V$ and the zero G voltage is $U_{zero} =$ 1.4V, which is the voltage which corresponds to $0g$.

Beginning with the conversion to voltages we use the formula (accordingly for y and z values):

$$\Delta_{voltX} = x[g] * sens[mV/g] = 115mV = 0.115V$$

The result represents the voltage difference between the zero voltage and the value we need, so we add the zero voltage to get the real voltage for the axes:

$$U_x = \Delta_{voltX} + U_{zero} = 1.515V$$

These voltages are the accelerometer output. From there data undergoes an analog digital transformation. The MPU-6050 contains a 16bit AD changer, so we multiply with $2^{16} - 1$:

$$A_x = \frac{U_x}{U} * (2^{16} - 1) = 30087$$

The same formula applies for the y and z values. This is the result after AD changing, the vector $\vec{A} = (A_x A_y A_z)^T$ is also the input for the DMP [50].

4.2 Physical Principle of the Gyroscope

The gyroscope was discovered 1817 by Johann Bohnenberger, and was primarily used on ships to determine the angle, relative to the suns position. Only a few year later Foucault used it to demonstrate earth's rotation. In 1916 other industries, like aviation started to use it too. Nowadays MEMS gyroscopes are used in laster pointer devices, motor bikes, compasses, autopilots, segways and of course also in robotics and virtual reality devices to determine the orientation. Mechanical gyroscopes normally consist of a wheel, where all of the mass needs to be as far away from the center as possible, which results in a heavy outer rim around the disk. The wheel is mounted in two, sometimes three gimbals to allow rotation around each axis. In the old gyroscope shown in figure 18 on the left, the disk with the golden rim represents the spinning inner wheel, the star form on the inside is not relevant for the physics of this gyroscope. Usually to get the gyroscope in motion a string is wrapped around the stick mounted to the wheel and pulled away fast to get it spinning. In this version the gimbals are mounted solid so they will not allow rotation. This is the kind of gyroscope that will not fall from the little pedestal when the wheel is spinning, even when put on top in a 90° angle, it will only rotate around the pedestal as long as the wheel is spinning. The unusual thing about gyroscopes is, that they behave like any other object, but when spinning their behavior changes. In such a spinning condition, a gyroscope resists certain movements. Take the inner orange disk at the bottom of figure 18 for example. Although the outer ring is rotated and therefore a force is applied to the object, the inner wheel is still aligned with the gravitation axis. This kind of free gyroscopes, the ones where the inner wheels stays horizontal and is not fixed to the outer gimbals, is used for measuring yaw, pitch and roll [49].

As a mind experiment, imagine a ring with four point masses spinning in space like shown in the top line of figure 18. The ring will keep spinning in the same direction as long as there is no outer force applied to it. When applying a force to one of the point masses, each of the masses will want to keep its initial direction of movement, which results an the ring begin slightly tilted but still rotating. The force is applied to the masses as a vector pointing in the direction of the rotation axis (middle picture). The faster the gyroscope rotates the larger its angular momentum and the smaller the effect of the force vector. Simply put, this means that the gyroscope disk will stay upright even if force is applied to tip it over, given that it is rotating fast enough. The sort of gyroscope discussed up to this point is called a mechanical gyroscope and relies on the maintenance of the angular momentum.

Another type of gyroscopes uses the Sagnac effect, using the character-

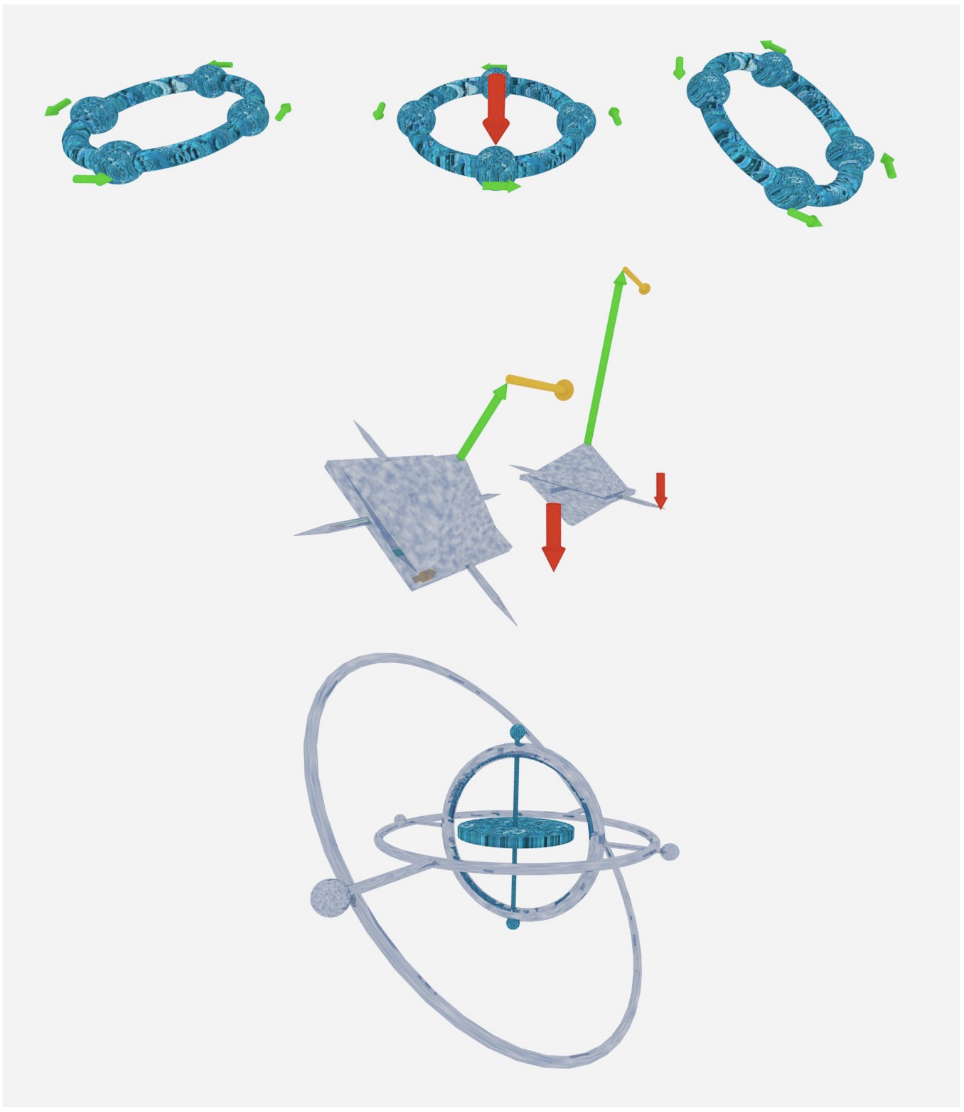


Figure 18: For simplicity, in the upper row the gyroscope wheel is depicted as a ring with four point masses. In the first picture the wheel is spinning, the motion arrows in the second picture show this. When a force is applied as shown, each point mass wants to keep moving in its original direction, resulting in a tilting of the whole ring like in picture three. The lower pictures show the difference the spinning speed makes. In the left picture, the object is rotating slowly, as a result the torque has a much higher impact than in the next picture, where the impact is lower due to a large angular momentum vector and the object stays more stable. The last picture shows the sort of gyroscopes that are used for angular measurements. Forces in the form of rotation applied to this gyroscope will not change the inner disks orientation as it will stay aligned to its original rotation axis.

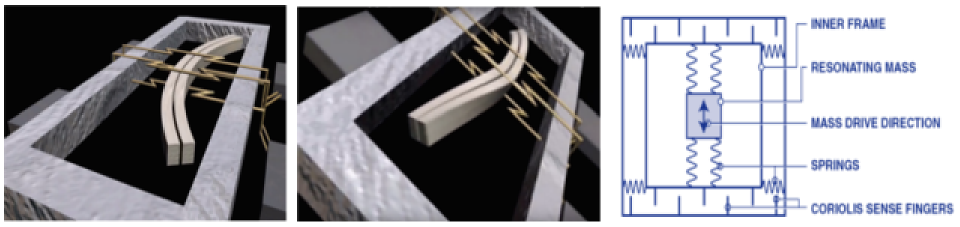


Figure 19: The first picture shows a MEMS gyroscope, with a vibrating object in the middle, in the second picture the device is rotated and the vibrating object stays in its original rotation plane, causing a force on the outer holding and inducing an electic field due to the piezoelectric effect. The third picture shows such a sensor in more detail [52].

istic of light, to get to the same results. This is done by for example a ring laser gyroscope, where two light beams travel in a circle, one clockwise, one counterclockwise. In the ring laser gyroscope this is archived with the use of mirrors to bounce the light. When the setup is rotated, one beam will have to cover a larger distance. From the difference in the return time of the two beams, conclusions about the angular velocity can be drawn.

The third kind of gyroscope uses the Coriolis effect for its measurement. This is the kind used for MEMS sensors and is called vibratory gyroscope. The MEMS version of a gyroscope works a little different than the traditional ones with the spinning wheel inside gimbals or the light bouncing versions. In contrast to their mechanical counterparts, which still have high part counts and are therefore expensive, MEMS gyroscopes can consist of only three parts. The main advantages of a MEMS gyroscope are its small size, low weight, high reliability, low maintenance need, inexpensive production, low power consumption and its robustness to work even in hostile environments [30]. Unlike for mechanical gyroscopes the basis for MEMS gyroscopes is rather a vibrating than a spinning mass. While vibratory gyroscopes were already know to Foucault in the middle of the 19th century, spinning gyroscopes where dominantly used until the emergence of the MEMS technology [51]. While it is possible to use spinning gyroscopes for MEMS, they never became successful due to their inherent instability.

The idea of the vibrating gyroscope goes back to the popular Foucault pendulum experiment, which was the first proof of earth rotation. The pendulum consists of a high mass suspended on a long rope. With gravity being the only external force to it, the pattern drawn on the floor by the pendulum is not a line, but a star kind of pattern. The only explanation is that not the pendulum but the ground experiences a rotation. The force affecting the pendulum in its own local system was called Coriolis force, which is not a force limited to earth rotation but applies to all rotations. The Coriolis force induces an acceleration if an object moves relative to a point and rotates around that point at the same time as it is the case with the pendulum (watched from the perspective of the pendulum).

In a MEMS gyroscope, the swinging motion of the pendulum is replaced by a vibrating material inside a surrounding frame. Like in the mechanical gyroscope where the wheel tends stay spinning in its original plane even when the outer frame is rotated, the same holds for the vibrating material. It keeps on vibrating in its original plane, given that the vibrating speed is high enough. Due to the pressure the Coriolis force puts on the vibrating material while rotating, a force is injected from the material to the outer frame. Like before with the accelerometer, the pressure on the piezoelectric components can be read of as an electric current. This current represents the change in the angular velocity of the object. Figure 19 shows how such a vibratory gyroscope is build. The acceleration induced by the Coriolis force acts perpendicular to the direction of movement (here vibration):

$$\vec{a_{cor}} = 2 \cdot (\vec{\omega} \times \vec{v})$$

Where $\vec{v}$ is the velocity of the vibration (in the picture that vector would be in the plane of the frame and $\vec{a_{cor}}$ standing perpendicular on it), and $\vec{\omega}$ the angular velocity at which the outer rotation of the device takes place. The resulting force is:

$$\vec{F_{cor}} = 2m \cdot (\vec{\omega} \times \vec{v})$$

As is clear from the picture, such a gyroscope only measures the rotation velocity (current induced as force by the Coriolis force) on one axis. To cover all three axes, three gyroscopes are needed.

After having determined the general way vibratory gyroscopes for MEMS work, let us do the same as before with the accelerometer values and have a look how to process that output angular velocity into useful signals.

The final results of the AD changer, starting from the three measured raw values of ω_{yz}, ω_{xz} and ω_{xy} for the rotation velocity around the x-, y- and the z-axis, did undergo the same transformation as the accelerometer values:

$$\omega_{yz} = \frac{((Gy_{yz} \cdot sens) + U_{zero}) \cdot (2^{16} - 1)}{U}$$
$$\omega_{xz} = \frac{((Gy_{xz} \cdot sens) + U_{zero}) \cdot (2^{16} - 1)}{U}$$
$$\omega_{xy} = \frac{((Gy_{xy} \cdot sens) + U_{zero}) \cdot (2^{16} - 1)}{U}$$

Gy_{yz} here is the change of the rotation in the yz plane, respectively around the y axis, measured by the gyroscope. The unit of that value is degrees per second. U_{zero} and $sens$ are the corresponding values for the gyroscope [50].

To get from an angular velocity [degree/second] to an angle [degree], one integrates over time, as it is always done to get from a velocity to a distance

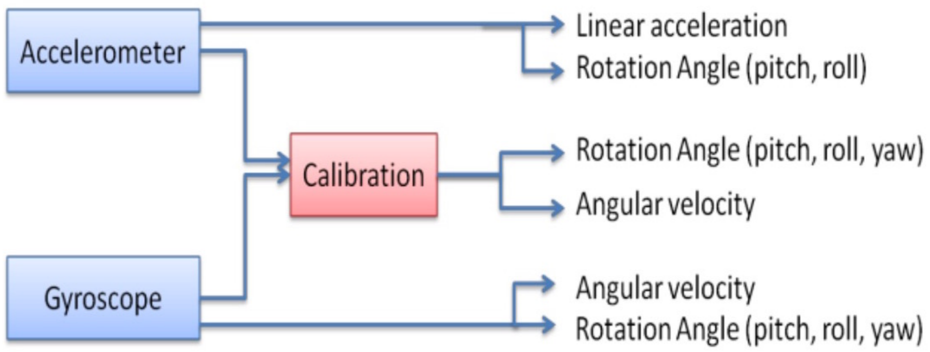

Accelerometer
Calibration
Gyroscope
Linear acceleration
Rotation Angle (pitch, roll)
Rotation Angle (pitch, roll, yaw)
Angular velocity
Angular velocity
Rotation Angle (pitch, roll, yaw)

Figure 20: Schema of sensor output and fusion, here calibration [53].

(here only the example for one of the axis):

$$\alpha_{yz}(t) = \int_0^t \omega_{yz}(t)dt$$

The continuous signal (the integral) is then transformed into a discrete digital signal. This is necessary but introduces some further errors to the system.

Furthermore, gyroscopes drift. This is unavoidable. Drift occurs as a function over time. Intrinsic errors of the sensors such as some minor noise also are accumulated with time due to integration. This leads to errors like not returning to zero if the sensor is placed in a level position. Zeroing out whenever possible (that is when the gyroscope is in a stationary position) helps reducing the problem, so does continually correcting it. Concrete correction methods are described in the following part.

4.3 How does data fusion work?

In this chapter, the data fusion of the gyroscope and accelerometer values is examined more closely. An object in space has 6DOF, three for the position and three for orientation. So one might be lead into thinking, that with a 6DOF IMU it is possible to get a position and orientation of an object. That is not the case, because both, the three accelerometer and the three gyroscope values are used to get the same thing: the orientation. How to combine this data is explained in the following.

Gyroscope readings have the problem that they tend to drift, in contrast to accelerometers. On the other hand, accelerometers are prone to errors because of vibration (in general linear mechanical movement), because they simply measure all the forces applied to it, whereas this is not a problem gyroscopes suffer from, because they measure angles and not linear movement and are therefore not sensitive for those. Also accelerometers are more reliable in the long run, whereas gyroscopes have their best results shortly after power on. So both sensors have their different sources of errors; combining them is therefore a valuable step to balance this and get a better estimate for resulting angles as a means to combine the best of both sensors. Figure 20 shows what accelerometer and gyroscope return in detail.

Algorithm which archives this usually work the following way: The accelerometer readings from the AD changer give an orientation, which we will then correct with the integrated gyroscope data from the AD changer, as well as taking previous already calculated results into account. Examples for such algorithms are the Complementary filter and the Kalman filter.

4.3.1 Kalman and Complementary filter

For combining accelerometer and gyroscope output into one final object orientation filters like the Kalman filter or the Complementary filter can be used. Being fairly easier to understand, the Complementary filter offers a good starting point. The theory is, that at the beginning the more stable gyroscope measurements are weighted heavier, while later on the better results come from the accelerometer so then they are valued higher.

The drift of the gyroscope is a low frequency signal and thus can be filtered out by applying a high pass filter. The formula is:

$$\begin{pmatrix} \alpha_{gyro,n} \\ \beta_{gyro,n} \\ \gamma_{gyro,n} \end{pmatrix} = \tau \cdot \begin{pmatrix} \alpha_{gyro,n-1} \\ \beta_{gyro,n-1} \\ \gamma_{gyro,n-1} \end{pmatrix} + \tau \cdot (\begin{pmatrix} \alpha_{RawGyro,n} \\ \beta_{RawGyro,n} \\ \gamma_{RawGyro,n} \end{pmatrix} - \begin{pmatrix} \alpha_{RawGyro,n-1} \\ \beta_{RawGyro,n-1} \\ \gamma_{RawGyro,n-1} \end{pmatrix})$$

This uses the raw data as well as the previous raw data, calculates the difference (which may or may not be drift) and adds some amount of the previously filtered value to it. τ determines how much drift correction is desired. *alpha*, *beta* and *gamma* stand for yaw, pitch and roll here.

The corresponding theory holds for the accelerometer. Its data needs to be smoothed out, for example by using a low pass filter, which damps the higher frequencies which represent noise. The equation for a low pass filter is:

$$\begin{pmatrix} \alpha_{acc,n} \\ \beta_{acc,n} \\ \gamma_{acc,n} \end{pmatrix} = \tau \cdot \begin{pmatrix} \alpha_{RawAcc,n} \\ \beta_{RawAcc,n} \\ \gamma_{RawAcc,n} \end{pmatrix} + (1-\tau) \cdot \begin{pmatrix} \alpha_{acc,n-1} \\ \beta_{acc,n-1} \\ \gamma_{acc,n-1} \end{pmatrix}$$

Where the result is the n'th filtered data coming from the n'th raw data and the result from the previous step.

Figure 21 shows how the data is filtered and combined, the corresponding equation is (only for α here but equally for β and γ):

$$\alpha_n = (1-\tau) \cdot (\alpha_{n-1} + (\omega_{gyro,n} \cdot dt)) + \tau \cdot \alpha_{acc,n}$$

Tweaking τ for optimization leads to either reduced drift in gyroscope readings or reduced noise in accelerometer readings. τ should be a function of

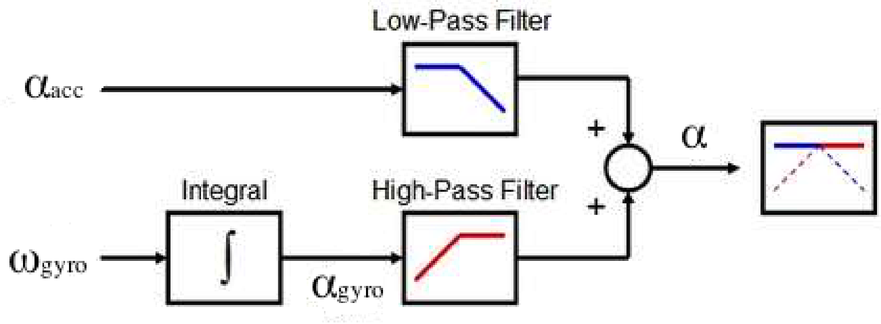


Figure 21: A low pass filter is applied to the accelerometer readings and a high pass filter to the integrated gyroscope readings. The combination of both determines the final rotation angle.

the drift and noise functions. Drift reduction is also improved by merging previous data with actual gyroscope data.

More advanced than the Complementary filter is the well known Kalman filter. As well as its easier counterpart, this filter serves to correct errors from erroneous instruments. Sources for these errors and the mathematical and physical structure are presumed to be known beforehand. The filter is a stochastic estimation procedure. Kalman filtering is an iterative filter technique. The exact theory behind this filter is for example explained in this paper [54]. Here, we will only look at the application needed to combine accelerometer and gyroscope data.

As an input to the filter we need a linear model of our problem:

$$x_{k+1} = A \cdot x_k + B \cdot u_k$$

Here x_k is the state of the system at time k, x_{k+1} then the state at the next time step. A system state is always described as a vector: $(\alpha \quad \alpha_{bias})^T$. α of course is the rotation angle, and in α_{bias} the bias, which represents the drift of the gyroscope is stored. So in the end, when wanting to find the final angle, one can easily subtract α_{bias} from α. u_k is the gyroscope reading, which is the angular velocity ω. For the matrices A and B we use their definitions:

$$A = \begin{pmatrix} 1 & -dt \\ 0 & 1 \end{pmatrix}$$

$$B = \begin{pmatrix} dt \\ 0 \end{pmatrix}$$

Putting this all together shows why:

$$\begin{pmatrix} \alpha \\ \alpha_{bias} \end{pmatrix}_{k+1} = \begin{pmatrix} 1 & -dt \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \alpha \\ \alpha_{bias} \end{pmatrix}_k + \begin{pmatrix} dt \\ 0 \end{pmatrix} \cdot w_k$$

Resolves to:

$$\alpha_{k+1} = \alpha_k - \alpha_{bias,k}dt + \omega_k dt \qquad and$$
$$\alpha_{bias,k+1} = \alpha_{bias,k}$$

Now, the angular speed ω is integrated to get an angle. B is zero in its second value, because there is no bias measurement coming from the gyroscope. This formula predicts the state and is updated in each step (after every reading) to get the new state from all the old ones. To show the correct notation:

$$\hat{x}_{k+1|k} = Ax_{k|k} + B\omega_k,$$

where $\hat{x}_{k+1|k}$ means the estimate of the state at time k+1 considering all previous states up to time k. The gyroscope values are used to build the state as the equations show. For the prediction in each step until here only the gyroscope values are considered in relation to previous states. How to get the acceleration values into play? Additional to the prediction step an update step is performed for each time step. In the prediction step we had calculated $\hat{x}_{k+1|k}$, then, the only thing left to do is, to also take the current step into account, thus to calculate $\hat{x}_{k+1|k+1}$:

$$\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + K_k \cdot I_k, \tag{1}$$

where I_k is called the innovation and is the step where the accelerometer values are used. When we let y represent the accelerometer reading of the angel, in each step we calculate the difference between the accelerometer reading and the predicted value to get the innovation:

$$I_{k+1} = y_{k+1} - C \cdot \hat{x}_{k+1|k}$$

C is simply defined as $C = (1 \quad 0)^T$ to only take the calculated angle into account. The innovation also has a covariance, defined as:

$$s_{k+1} = C \cdot P_{k+1|k} \cdot C^T + R$$

Both P and R are covariance matrices themselves. While $P_{k+1|k}$ is the error covariance matrix considering all previous error covariance matrices, R is the measurement covariance matrix. Therefore, R reflects the amount of jittering expected in the accelerometer readings.

This leaves us with K, called the Kalman gain. It is meant to indicate how trustworthy the innovation is. That means, it determines how correct we assume the accelerometer reading to be in exactly this step. The formula for the Kalman gain is the following:

$$K_{k+1} = A \cdot P_{k+1|k} \cdot C^T \cdot s_{k+1}^{-1}$$

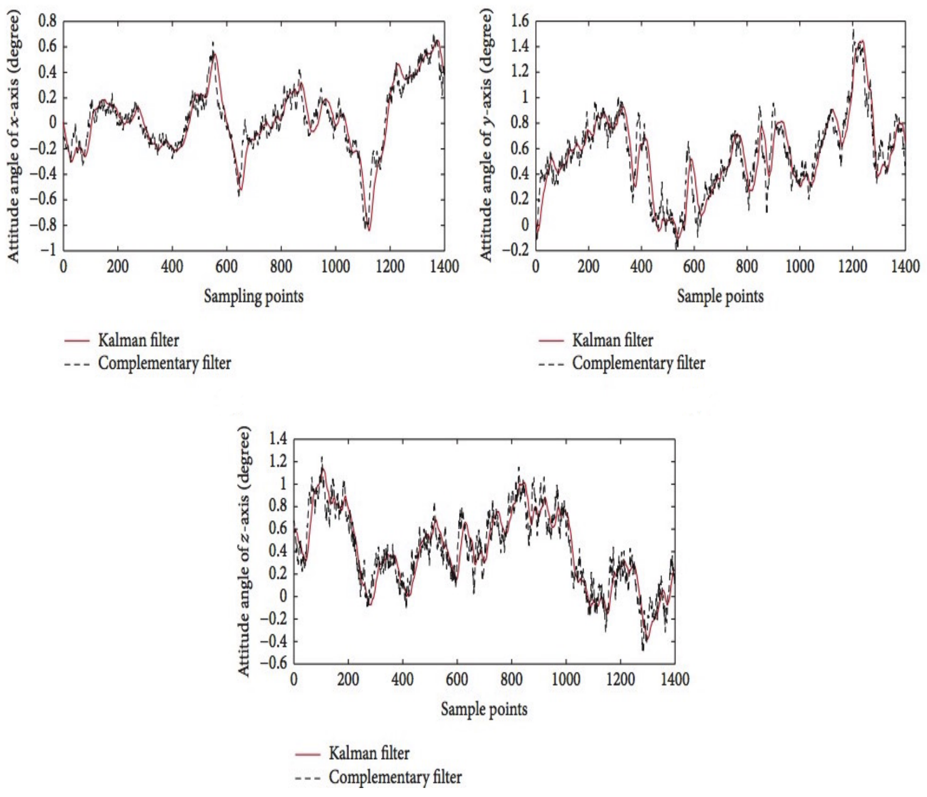


Figure 22: Comparison of sensor data once filtered with the complementary filter and with the Kalman filter for the red curve [55].

As a final step, the error covariance matrix $P_{k+1|k+1}$ is updated:

$$P_{k+1|k+1} = A \cdot P_{k+1|k} \cdot A^T - K \cdot C \cdot P_{k+1|k} \cdot A^T + sn,$$

with sn as the noise covariance matrix [54].

After having gone trough the significantly more complex Kalman filter, what are the differences in the results these two filters provide? The paper [56] shows, that the Kalman filter is indeed the best filter one can use, since it minimizes the error from the input signals. Figure 22 shows how similar the results on all three axis are, with the results of the Kalman filter being a bit smoother than the results from the complementary filter. Also it is visible that the complementary filtered curve lags a little behind the Kalman filtered curve.

4.4 Quaternion math

Rotations can be described mathematically, either with rotation matrices, Euler angles, angular axis or quaternions. To rotate a point one can simply multiply it with the corresponding rotation matrix. This is the easiest approach, but nonetheless also the slowest, because one needs nine values for the matrix in a case where three would suffice. Euler angles can be imagined as rotating an object around three axis in a sequence, here called yaw, pitch and roll. The problem of that easy and intuitive solution is that Euler angles suffer from gimbal lock, which leads to an unwanted rotation around the x axis.

A solution to this problem are quaternions, which basically present an extension of complex numbers into three dimensional space. A quaternion

$$\begin{aligned} q &= (w, \vec{n}) && \text{with} \quad w \in \mathbb{R}, \vec{n} \in \mathbb{R}^3 \\ &= (w, (x, y, z)) && \text{with} \quad w, x, y, z \in \mathbb{R} \quad = w + ix + jy + kz \end{aligned}$$

consists of an angle w and a vector $n = (w, y, z)$. To rotate a point in space, it is rotated about the degree of w around the vector $\vec{n}$. Similar to complex numbers, w is the real part and $\vec{n}$ the imaginary part with:

$$i^2 = j^2 = k^2 = -1$$

Here, only the operations necessary for the code explained in the next chapter, are covered. For a more detailed description see [57].

For rotating a point with a quaternion, the following formula can be used:

$$p' = q \cdot p \cdot q^*$$

with q^* being the complex conjugate of q: $q^* = (w, -\vec{n})$. However, the sensors do not provide rotations relative to a previous position, they supply us with absolute orientation data. Therefore, instead of using this formula the orientations can simply be assigned to the orientation of the 3D object. Additional to just assigning the orientation, some correction procedures need to take place, therefore adding up two rotations becomes necessary as well as subtracting rotations:

$$\begin{aligned} q_3 &= q_1 \cdot q_2 \\ q_4 &= q_1 \cdot q_2^{-1} \end{aligned}$$

The first equation shows adding two rotations together by multiplying their quaternions. To subtract simply use the inverse of the second quaternion as shown in the second equation. This is all that is needed for quaternions, the application of these formulas are shown in the next part.

4.5 Rotational math: From Arduino to Blender

In this chapter the transformation from the MPU-6050 coordinate system, to the Blender coordinate system are covered. This goal is shown in figure

23. Distinctive for Blender is how it orients the local coordinate systems for bones. The y axis is always oriented along the bone, meaning yaw rotations are rotations along the y axis.

In the following, coordinate systems are distinguished like these:

- I: IMU system
- B: Blender system
- O: Blender system without offset
- F: Blender system, free of influences from other bones and without offset

Let us look at one example, e.g. the head. The set of rotations is different for every sensor, as sensors are oriented differently along the body as well as the bones have their distinct locale systems. For the head bone for example, the coordinate system, shown in figure 23, is the correct one, for the legs this system is turned upside down, to the right or the left respectively for the arms. Imagining the sensor shown on the right side of the diagram standing upright on its lower edge, the y axes match one another, but the x and z axes need to be inverted.

Since, we are in the right coordinate system now, rotations are now performed around the right axis and in the right directions.

For all but the four arm sensors, an additional rotation is required in order for them to be physically held upright and not lie flat on a table. The bone is rotated such that it recognizes the upright position of the sensor as its starting point. In this case, a rotation of 90° about the x axis is performed:

$$R_{90x} = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0, 0)$$

Furthermore it has to be considered, that, if an actor tilts his body to the right, without moving his head, the sensor mounted to the head will obviously recognize the same tilting and as a result will tilt the head as shown on the left side of figure 24, which is not desired. What is, is what is displayed at the right side, that the tilting of the head sensor is detected as wrong and corrected accordingly. This is archived by multiplying the inverse of the parents bone rotation (e.g. body bone) to the heads rotation from the right to neglect the effect.

In addition to the issue discussed above, calibration of the sensors is not perfect, but more from that in the next chapter. For now, it is enough to know, that we have to deal with an initial wrong offset in the rotation about the z axis (which corresponds in the z axis of the IMU and therefore is yaw). This however can simply be solved by subtracting or adding the wrong offset from or to the rotation quaternion q_{offset}. For most of the sensors the offset is about 0.05. This will be fine tuned for each sensor inside the Blender code.

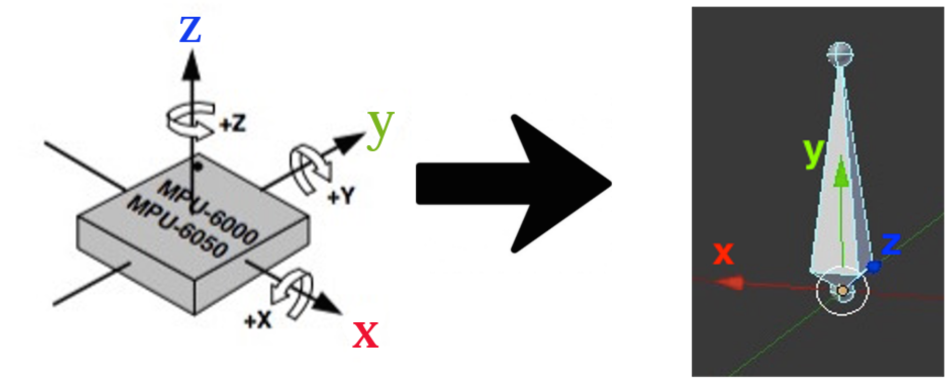


Figure 23: Transformation: MPU-6050 coordinate system to Blender local bone coordinate system.

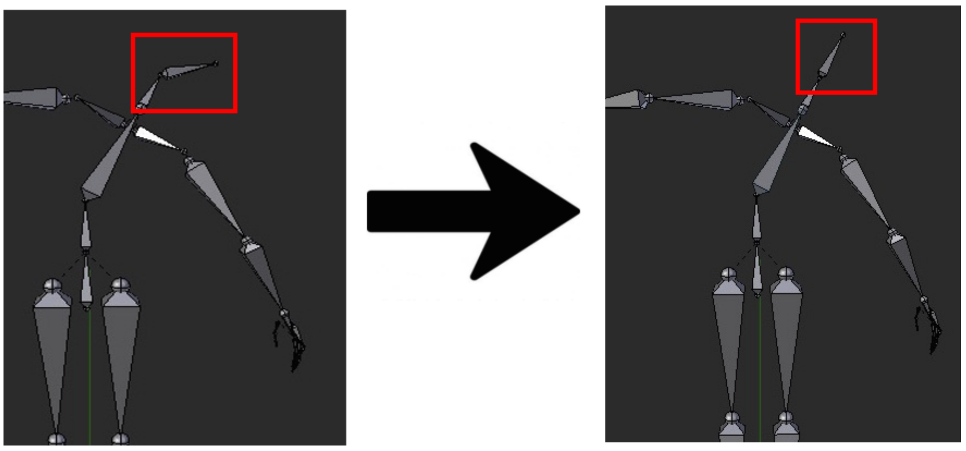


Figure 24: Wrong tilting of the head due to movement of the body. The sensor on the head is unable to tell which rotation comes from the body alone and which is from the head.

The goal is to go from the IMU system I to the final system F. Hence:

$$T_I^O = \underbrace{T_I^B}_{(w,-x,y,-z)\cdot(\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}},0,0)} \cdot \underbrace{T_B^O}_{q_{off}} \cdot \underbrace{T_O^F}_{q_{parent}^{-1}}$$

5 Arduino code

As explained previously, the Arduino boards are essentially microcontrollers with a programming interface. This enables programmers to upload their code to the boards. After the first upload the microcontroller will execute the uploaded code on every reset (which can be manually done by pressing a button) until a new one is uploaded.

5.1 Code explanation

For the general setup a few steps are necessary to get the computer to receive and transmit data from the Arduino, and the Arduino to receive this data from the IMU in the first place. For the latter part the I2C interface is used, by initializing the Wire and setting the clock rate for the I2C data transfer. This is done with the help of the i2dev library by Jeff Rowberg, which provides a general interface to I2C devices. For the former part a serial connection between the Arduino and the computer has to be opened and a baud rate set. Here 115200 was chosen, which means, that as many symbols are transmitted per second. After checking if the connection was successfully established, `mpu.initialize()`, where mpu is of type MPU-6050, starts the IMU.

Also before we can turn to look at the DMP now, it is a good idea to make the actual start of the reading user input dependent, which later allows us to start the sensors without starting the actual measurement and therefore start the measurement of all sensors together when starting the Python program, which will be explained in the next chapter. `Mpu.dmpInitialize()` initializes the DMP, then the corresponding offsets need to be set, which are different for every MPU but this will be explained directly in the following subchapter. After some error detection code we move on to the `loop()` method.

For starters, it is necessary to wait for an interrupt from the IMU, which informs the Arduino, that there is some data it wants to transmit. The DMP is a FIFO buffer performing all the math from chapter 4.3. At this point, the data lies there for us to use inside the buffer and it is as easy as `mpu.dmpGetQuaternion(&q, fifoBuffer)` to read it from there. Q is a Quaternion here and fifoBuffer represents the actual buffer itself. With a simple serial print order, the received quaternions are send to the computer. Everything that is transfered out is simply applied to the TX (transmit) pin. Because there is our Bluetooth modules RX (receive) pin connected to it, this module takes over handling the data from here.

However, this covers only the necessary steps for a sensor to transmit its own collected data, but for reasons stated earlier, it is inevitable to connect at least two sensors together and have them send their data together. That is why only half of the sensors are equipped with Bluetooth modules. Sensor, which purely send their own data are called slaves here, sensors, which control when their slave is allowed to send and forward the data to the computer, are masters. Therefore, the steps explained above are totally sufficient for the slave. For the master however, some additional lines of code are required.

First, the Arduino Pro Mini, which was used provides, in comparison to the (much bigger) Arduino Uno, only one interface to connect over a serial line through its RX and TX pins. As this connection is already taken to connect to the computer via Bluetooth, another solution had to be found. With the help of the SoftwareSerial library, it is possible to transform two out of the eight analog output (PWM) pins into serial out- and input pins. This is done by defining `SoftwareSerial slaveSerial(10, 11)`, which makes 10 the second RX pin and 11 the second input pin. As one would expect, this connection needs to be set to a specific baud rate on startup as well. This is where the first problem occurs. Having the same baud rate as the master is not possible, because the software solution lacks the underlying performance optimization of the hardware serial solution and is simply not able to send on speeds that high.

After some testing, the highest speed possible, where the received data is still sufficiently accurate (which means without any transposed digits) was found to be 57600. It is convenient to have the sensors be able to start their readings all together at the program startup and not individually on power up. Therefore, at the moment the master receives his own "start" signal though its serial connection to the computer, it forwards the information to the slave to make him start as well.

Again, inside the loop, for every Quaternion read from the DMP of the master, the slave also sends one, which the master has to forward. The following code does the trick.

```
// Now print slave values
while(slaveSerial.available()>0
       && lineEndeReached==false)
{
 slaveRecieved = slaveSerial.read();
 slaveInData += slaveRecieved;

 if (slaveRecieved == '\n')
 {
  Serial.print(slaveInData);
  slaveInData = ""; // Clear recieve buffer
  lineEndeReached = true;
 }
}
lineEndeReached = false;
```

This is necessary, because the read() method, only provides us with the functionality to read the first byte of the incoming data. To read a whole line (which equals one quaternion), all the bytes until an endline symbol is reached, have to be read and put together as one line for the master to forward.

As mentioned in chapter 3.2.3, relying on so many Bluetooth connection did not result in a satisfying solution. One of the (later discarded) solution approaches featured connecting multiple sensors together with cables. The idea was that the interconnection of two sensors was already working just fine, so an extension to more sensors should not be a problem. Unfortunately, this is not the case. The first idea, of having three master sensors and connecting two or three sensors as slaves via the software serial interface did not work out, because of the limitations of the Arduino Pro Mini and all its related Arduino boards of the same size. Unlike the much bigger Arduino board, which features two hardware serial and multiple software serial lines, the software serial library can extend the smaller breakout boards only by one serial connection. Thus, a maximum of two Arduinos can be connected this way.

Another attempt to bypass that problem, is to connect the sensors in series, which basically means that one sensor only sends the data to the next one, which combines his own data with the received data and forwards the collection to a next sensor and so on. The final sensor in the line would then use its Bluetooth module to send the combined data to the computer. This did not work out, because as it turned out after some testing, the reliability of the final data stream was just not sufficient. Most of the data was corrupted when arriving at a two hops distant Arduino. This was tested with all the boards fixed in their zero position, so all lines should have looked like that:

```
m1.00;0.00;0.00;0.00
```

where the first letter represents the sensor and the rest a quaternion with w, x, y, and z readings. Some of the values can also have a minus in front, which does not change the result. Even though testing with all supported baud rates, the results did not even come close to being correct. In a test set of hundred readings (with baud rate 57600), 87 reading where corrupted, either completely or at least having one character wrong or missing completely. An attempted explanation is, that the timing between the sensors was off, or that the amount of data transmitted was simply too high in a too short frame of time. After all we are talking about two IMU readings being send on a serial line and after that, three IMU readings transmitted through only one Bluetooth module. Since this number of failure is clearly unacceptable, the idea was dismissed.

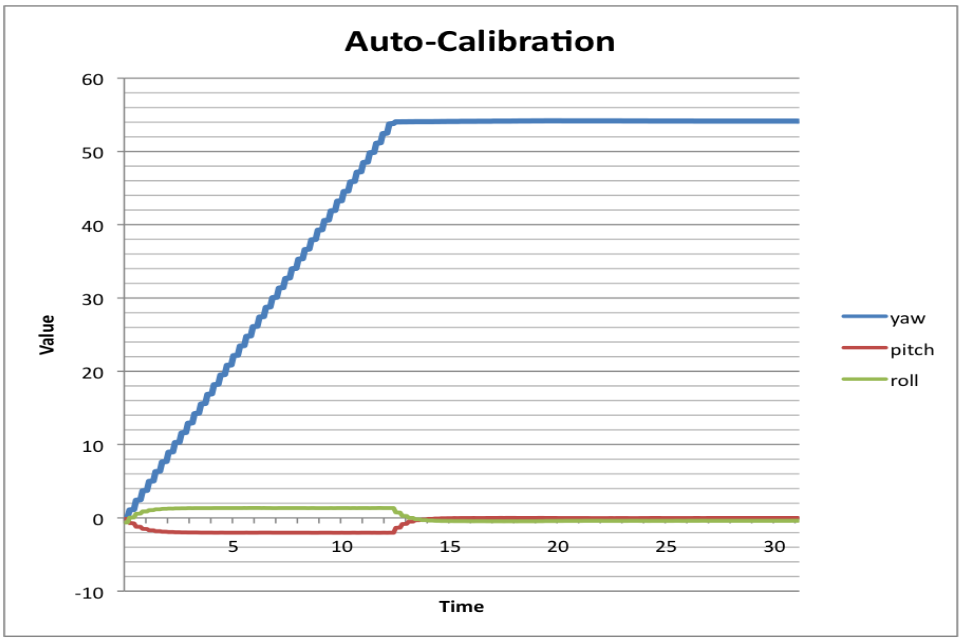


Figure 25: Output when reading the DMP combined yaw, pitch and roll values. After some seconds the values stabilize [58].

5.2 The long way to calibration...

If a device is positioned perfectly level and not moving, which would be the ideal case, then:

- x, y and z gyroscope readings are 0, because there is no rotation currently taking place
- x and y acceleration readings are 0
- z acceleration reading is 1g due to gravitational acceleration (at the default sensitivity of 2g defined in the data sheet, this equals 16384)

Truth be told, this is not going to happen, because there will always be some noise and it is difficult to place a sensor perfectly level because of its high sensitivity.

So, when the sensor is positioned, the readings should be 1g for whatever axis is parallel to the earth gravitation vector. The other axis lying horizontally towards the gravitation vector should read zero. However, the raw values obtained by the sensor tend to be highly unstable. There may be strong changes between readings, by cause of the high sensitivity. The sensitivity of each sensor may be chosen (2, 4, 8, or 16 g for the accelerometer and 250, 500, 1000, or 2000 degrees per second for the gyroscope), here we took the default of 2 g and 250 degrees per second. To know the sensitivity is essential to determine the scale of the read values.

The conditions under which a calibration should take place are the following:

- The sensor is placed on a, as horizontal as possible, surface, preferably a concrete instead of a wooden table.
- The sensor should be running for at least some minutes before calibration begins to minimize influences caused by temperature fluctuations.
- At best, there should be no traffic at nearby streets.
- As power supply, a high enough battery instead of the unstable voltage coming from a USB port should be used.

Due to shipping times and temperature changes in the surrounding of the sensors, the initial calibration (which is not performed by all manufacturers) becomes unreliable. Now the task is to find the values for the gyroscope and accelerometer to get the data to match these ideal case values when level and motionless and add them as offset to each of the sensors. The function applied to the MPU are **setXGyroOffset()** and **setXAccelOffset()** as well as the same functions for the y- and z-axis. This offsets are different for each device, so each of the devices needs to be calibrated individually and the found offsets inserted as arguments into this function.

To find the necessary values a calibration sketch has to be uploaded to the Arduino first. Of course, as before, the I2C and serial connection to the computer have to be initialized before any further calculations can take place. For this, the basic idea is to make an initial guess and try to fit the average of a number of readings to it, until it converges [58]. Inside the loop function, the means to read the raw data (which had still undergone some DMP calculation) are provided by this function:

```
mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
```

where mpu represents the MPU-6050 object and the method reads three accelerations and three angels. It is recommended to wait at least ten to twenty seconds after power on to get the values to even out and get stable. Discarding the first few hundred readings is therefore essential, in order not to let the calculation be destroyed by those wrong start data. As the digram in figure 25 shows, the DMP does some auto calibration to get the values stable for itself during the first seconds after startup. Showing minor errors in pitch and roll during the first seconds, and huge drifts in the yaw value, the figure also shows that they get fairly stable after that (when the sensor is placed in a level position and not moved). The removal of the zero error, which is, that the sensor is situated in a level position but thinks it is angled, is the goal of our own calibration sketch.

Beginning calibration, the initial guess sets all six values to zero, a fixed amount (the higher it is the more accurate the calibration) of raw data is read with the above command and in each step the reading gets added to this guess. After dividing by the buffer size (number of readings), six values, the means, are returned. The following lines take this means and calculate the needed offset from them, by scaling them in accordance to the information provided by the data sheet:

```
gx_off=-gx_mean/4;
gy_off=-gy_mean/4;
gz_off=-gz_mean/4;
ax_off=-ax_mean/8;
ay_off=-ax_mean/8;
az_off=(16384-az_mean)/8;
```

Setting this values as our new guess, the whole process starts all over again. The program will converge if the calculated mean drops below a predefined value. The closer to zero (respectively 16384 for the z acceleration) this value, the more accurate the calibration. After having figured out the offsets, they are added to the main sketch and uploaded to the Arduino. Now, for better usage in further calculations, the values are output as quaternions. The w, reading is one, the x and y readings are zero as expected, but a problem is presented in the z value. Across tests with all ten sensors, the z value (after data fusion for gyroscope and accelerometer z readings) is not zero, but varies between 0.08 and 0.20. This error is owed to the inaccuracy of the yaw value readings due to the fact, that is has to be parallel to the z-axis. Not shown in figure 25, because it does not display a long enough time frame, the yaw value suffers some a certain drift over time. This is inherent to the way the sensors work and can not be changed with this sensors alone. A solution often used is the addition of a magnetometer, which can correct this error.

Normally in a six degree of freedom sensor, the accelerometer readings are used to minimize drift in the roll and pitch gyroscope readings as previously described in detail. Therefore it is rather astonishing that the diagram also shows a stabilization of the yaw value. The following try to an explanation can only be a guess, because InvenSense keeps its silence about the intern DMP methods at work. The DMP seams to realize that the constant changing it gets from the yaw is a drift error and corrects it accordingly. This is confirmed by rotating the sensor during initialization phase, then no such correction takes place during this time, but when letting the sensor rest again after some seconds the value is corrected, because again the absolutely linear movement is detected as an error. The conclusion from that, however, is the recommendation to let the sensors adjust, without moving them for ten seconds after power up.

6 Blender plug-in

This chapter examines the main software aspect of the thesis, thus, covering the python code for Blender, which was written for this project. Blender is an open source 3D software, maintained by the non profit Blender Foundation, and hugely supported by a vast online community. Due to this, developing add-ons and all kind of extensions to Blender is relatively straightforward.

In the course of the development the list of the requirements for this software became clear:

- open a serial connection to receive sensor data,
- read data from the stream and translate it to a format which can be interpreted as rotation values,
- have a character rig ready to apply rotation to, for every sensor, there needs to be a corresponding bone on which the rotation can be applied to,
- align the bone coordinate systems with the sensors coordinate systems
- set key frames, to make an animation,
- enable the exporting of the animation (preferably as a BVH file) for use in future projects and
- allow the user to view his motion applied to the character in real time

The demand for real time feedback makes the usage of the Blender game engine a natural choice. The engine is an independent additional component, which is based on a system of **logic bricks**, namely sensor, controller, and actuator bricks. The "always sensor" is used, when something needs to be executed each tick and is connected to a python controller. Through this combination the script attached to the controller is executed every tick. This obviously leads to problems in maintaining permanent values over the course of execution. The use of the function `globalDict()` from the `GameLogic Module` allows to bypass this problem by storing variables in a global way, so they can be interchanged between scenes. In this context that would be the variable, which holds the serial connection to the Arduino, which of course should be opened only once, but used in every frame. The main difference from the Blender standard is, that a script is always associated with an object, and while it can be attached to multiple objects, it is executed for each object separately. In theory, this should allow us to attach different scripts, opening different serial connections towards to different bones, without the need to use a separate threading system. In praxis however, this proved to simply be too slow. When connected to two sensors at a time, the scripts were not executed truly in parallel but rather alternating, which lead to one bone moving and then stopping, while the other one moved and stopped again and so on.

As an approach to avoid the problem of Blender being too slow, a C++ script was written, with the intention to send the data to Blender as a single combined stream, such that Blender only has to deal with picking that stream apart again, transforming it into rotation data and applying those to the bones, while the multiple serial connections are handled in C++. In contrast to Python, handling serial connections in C++ is a little bit more difficult, but faster on the other hand. The following code establishes the connection in the first place:

```
int open_port(const char* port){
  int fd = 0;
  struct termios options;
  fd = open(port, O_RDWR | O_NOCTTY | O_NDELAY);

  //ERROR CASE
  if(fd == -1) {
   return fd;
  }

  fcntl(fd, F_SETFL, 0);
  tcgetattr(fd, &options);
  cfsetispeed(&options, B115200);
  cfsetospeed(&options, B115200);
  options.c_cflag |= (CLOCAL | CREAD |CS8);
  options.c_cflag &= ~(PARENB | CSTOPB);
  tcsetattr(fd, TCSANOW, &options);
  return fd;
}
```

At the beginning, the serial connection is opened with the specified port of the Bluetooth connection, normally ”/dev/cu.name” or ”/dev/tty.name”. The flags are set to allow both reading and writing on the port without a delay. `fcntl` clears the flags of the file descriptor, `tcgetattr` reads out the exiting attributes, to allow changing them in the next step, in order to use the required baud rate of 115200.

The Arduino is programmed in a way, that requires the user to send a byte, as a permission for it to start sending. This can be done by one single command:

```
int n = write(fd, &c, 1);
```

where c can be any string, here, it is only a space symbol.

The function to read from the port afterwards requires a few more lines, as it has to read the data byte by byte:

```
std::string read_from_port (int fd) {
 char c = '␣';
 int i = 0;
 std::string line;

 read(fd, &c, 1);
  while (c!= '\n') {
   line.push_back(c);
   read(fd, &c, 1);
  }
 return line;
}
```

One byte is read and stored in c at the beginning. Until the end of a line, all the bytes are pushed into a list.

`Read_from_port` is called in a loop, which, unfortunately does not provide the necessary speed to get all the sensor readings in time. Threads are the solution to the problem. By opening a new thread for every sensor, the ports are read in parallel and their line readings stored in a global list, or when wanting to do it in real time, in a pipe each iteration of the inner loop. A pipe is a mechanism for interprocess communication, allowing data written by one process to be read by another one. This is realized by a FIFO buffer in a temporary file.

```
myfifo = "/tmp/myfifo";
mkfifo(myfifo, 0666);
pipe_fd = open(myfifo, O_WRONLY);
```

This three lines provide the means to share information between a C++ process and a Python process. The fifo allows writing and reading from all processes. With calling

```
write(pipe_fd, line.c_str(), strlen(line.c_str())+1);
```

in each iteration in every thread, the pipe should provide a real-time feedback of all sensor readings.

All that remains is reading and processing in Blender. Reading proved to be fairly easy in Python, with only two main lines of code:

```
os.mkfifo('/tmp/myfifo')
fifo = open('/tmp/myfifo', "r")
```

The data is send and received correctly, however, the visualization remains an obstacle.

It became clear, that in order for Blender to visualize a constant stream of data, a special operator is necessary. A **time dependent modal operator**.

Modal operators in Blender are useful for all constantly running functions. To name an easy example: For rotating an object manually, one presses R, from then on, all mouse movement is interpreted as rotation to the object. The operator finishes, when pressing either the left or right mouse button, to apply or to cancel. It works the same way for the data stream coming through the serial port. Once the data transfer starts, all of the data is applied to the object until the finish command is given. Prior to using this structure, the data transfer constantly stopped, because after a few readings the control was always given back to the main program for user input, interfering with the data stream.

Modal operators rely on events. This special operator uses a timer event to execute code and visualize the output every frame. Modal operators come in different kinds, some for example are useful for popup functionality, or for leaving file selector windows open until another event happens. The method `modal` provides the needed functionality, as it is called every frame by the `execute` method, which is basically a wrapper to call the function from the outside. It reports back that it is still running with RUNNING_MODAL, `modal` returns PASS_THROUGH which gives the handle back to Blender.

When not wanting to rely on C++ and interprocess communication, one can also use the internal methods of Python combined with the modal operator. C++ was considered because of its high execution speed and also because of the fact, that pure Python did not prove to be a workable solution, as modal operators were essential for the visualization, and threads where essential for providing the necessary speed. The combination of both, however, is not supported.

Nevertheless, a way for combining the two was found. So, let us start from the beginning. As briefly mentioned before, serial connections in Python are by no means complicated. The pyserial module provides excellent and easy access in only one line:

```
self.ser = serial.Serial(
                connectionName,115200, timeout=1)
```

Together with setting the rotation modes of the bones via:

```
self.obj.rotation_mode = 'QUATERNION'
self.obj.rotation_quaternion
            = mathutils.Quaternion( ( 1, 0, 0, 0 ) )
```

This is done in the `_init_(self)` function, which is by default called at a first step in the modal operator before entering the RUNNING_MODAL loop. Inside the loop it is still possible to listen to events, here only used for aborting the calculations.

The key to running threads inside modal operators is to use the `map` function of the, unfortunately undocumented, class `Pool`. By calling

```
pool.map(threadName, iter),
```

on the Thread pool variable `pool` every frame, new threads are being opened by the `modal` function of the timed modal operator. Also we need to keep track of the number of threads running in the pool, close and join them after finishing their tasks in order to maintain thread safety.

The whole functionality of the thread function is enclosed in a mutex for ensuring that the data is read correctly. Data is read by checking if there are new values available, reading, decoding and splitting every line in a loop:

```
while (ser[i].in_waiting !=0):
    data = ser[i].readline().decode().strip('\r\n')
    w, x, y, z = data.split(';')
    firstChar = w[0]
```

The only thing left to do is to apply the rotations as presented in the previous chapter. It has to be done differently for different bones due to discrepancies in IMU orientations and bone local coordinate systems. This is the example for the head bone:

```
q = mathutils.Quaternion( ( w, -x, y, -z )
q = rot_x_p * q
q.y += 0.05
g = obj[9].rotation_quaternion.copy()
obj[4].rotation_quaternion =   q * g.inverted()
```

Basically, this is the exact application of the equations chapter 4.5 demonstrated.

6.1 Animating a human being

Up to this moment, rotation values coming from the MPU-6050 were read, processed and matched to the corresponding bones on a Blender skeleton. Since human beings do not only consist of bones, a process called **skinning**, which fits a humanoid 3D model to the bone model, is applied. In Blender, this is done by assigning weights to each vertex, which determines how much its deformation is influenced by a certain bone. Blender provides means for automatic weight assignment, although this should be adjusted manually afterwards. Important to note is, that this can be done for arbitrary 3D characters (given a roughly humanoid shape). The Blender plugin written for this section, which essentially only needs the bone structure, is therefore applicable for all the different character one might want to animate without adjustments on a code level.

Character movements linked to the bone structure are caught in an animation by the plug-in. With the help of the BVH exporter coming natively with the Blender application, animations can be exported. This also allows us to take the captured motion and easily transfer it to other 3D Engines such as Unreal Engine 4 or Unity, to be applied to a final character there.

7 Results and Evaluation

IMUs suffer from two error categories: deterministic and random errors. Deterministic errors are generally easier to compensate for. A bias, for instance, is a deterministic error. IMUs can have a bias, that means a factor added to the real value, offsetting it. The inertial bias for each power up is different, when running the bias changes over time. This is often related to a change of temperature of the sensor. Another deterministic error is the scale factor error. If that error is for example two degrees, the results may vary about that amount from the real value.

On the other hand, there are the random errors, like for example **random walk**. This is the result of integrating a signal with the additive random noise signal added on top of the sensor readings. Sensor non-orthogonality, which basically means that the six sensors are not aligned right due to imperfect manufacturing, is a source of error too. For example, if a sensor is not aligned exactly at 90° to the z-Axis, than it will measure gravity too, which will result in wrong readings. This error however, can be reduced by factory calibration and testing.

Furthermore, timing errors are not uncommon when interacting with other devices. The following figures show an extraction of the results.

The prototypes orientation, transfered to a 3D model, is shown in figure 26. Powering and data transfer are done via the USB connection for this first try. The results are accurate and in real time. The same holds for the final sensor, which uses the Bluetooth connection. The sensor shown here is already calibrated manually, as most calibration routines proved not to be sufficiently accurate. Below, a comparison of the pre-installed DMP and a self written Complementary filter is visualized. As discussed in chapter 4, the Complementary filter does not provide accurate results for the yaw value, but otherwise equals the DMP results fairly well. Not visible on the picture however is the slight lagging of the Complementary Filter, as the red bar always follows the blue DMP bar with a delay of some milliseconds. Surprisingly, the DMP values for yaw turned out to be very stable as well, which has to be due to InvenSenses' internal algorithms. Validated by this readings, the usage of the DMP instead of the Complementary filter was decided to be the better starting point for further calculations, also given the aspect of real time tracking. Hence, all the code shown in the previous section is based on the DMP results.

Figures 27 and 28 show the final results of the motion capture application. In figure 27 the movement of four sensors (two for each leg) is tracked, while for figure 28, six sensors (two for each arm, one head and one body sensor) are tracked.

For the conclusion of this chapter the initial list of requirements and preferable properties is reviewed:

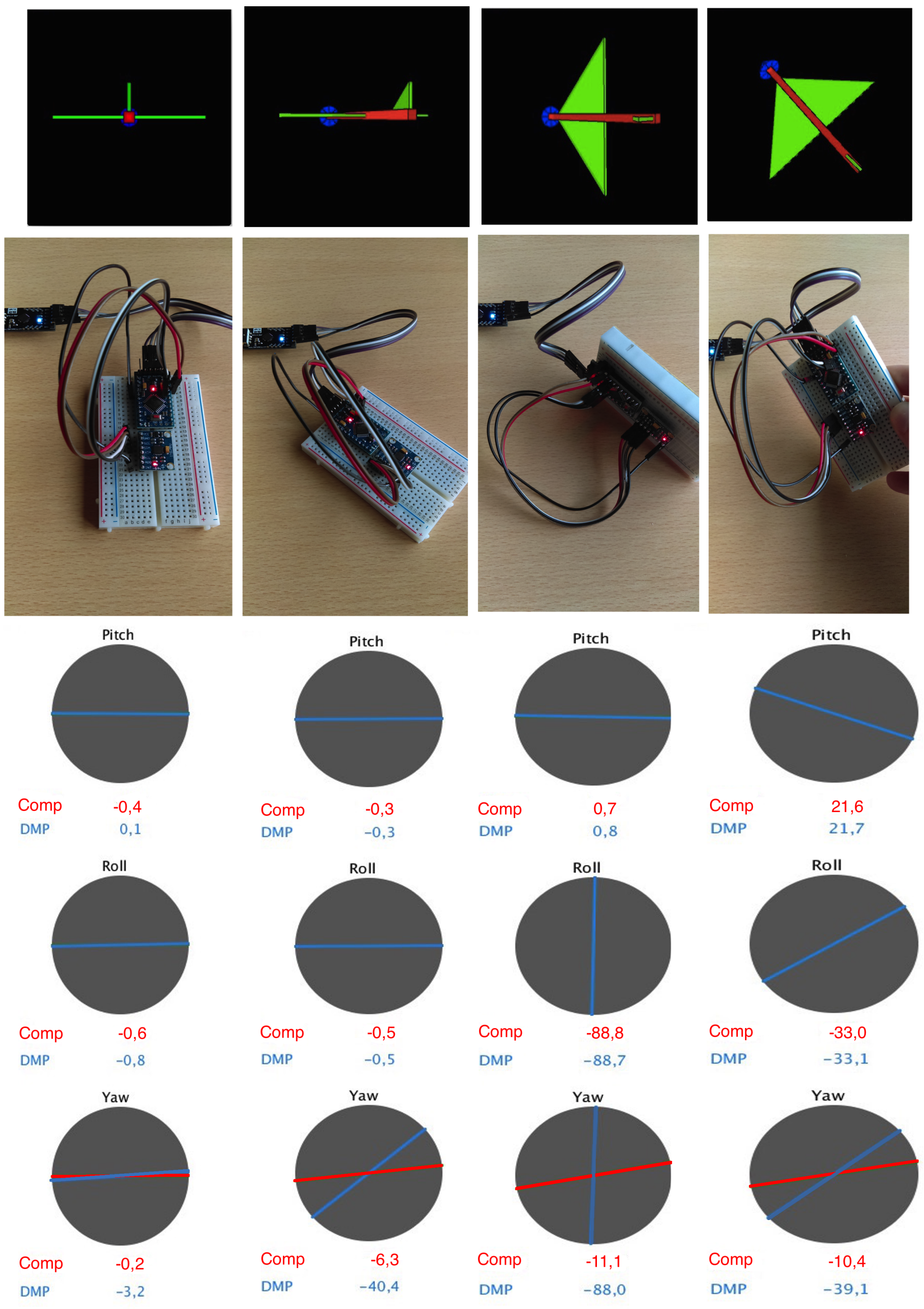


Figure 26: Here, the orientation of the prototype is transfered to a 3D model using the Arduino related Processing IDE. Also the figure shows a comparison of the DMP and the Complementary Filter.

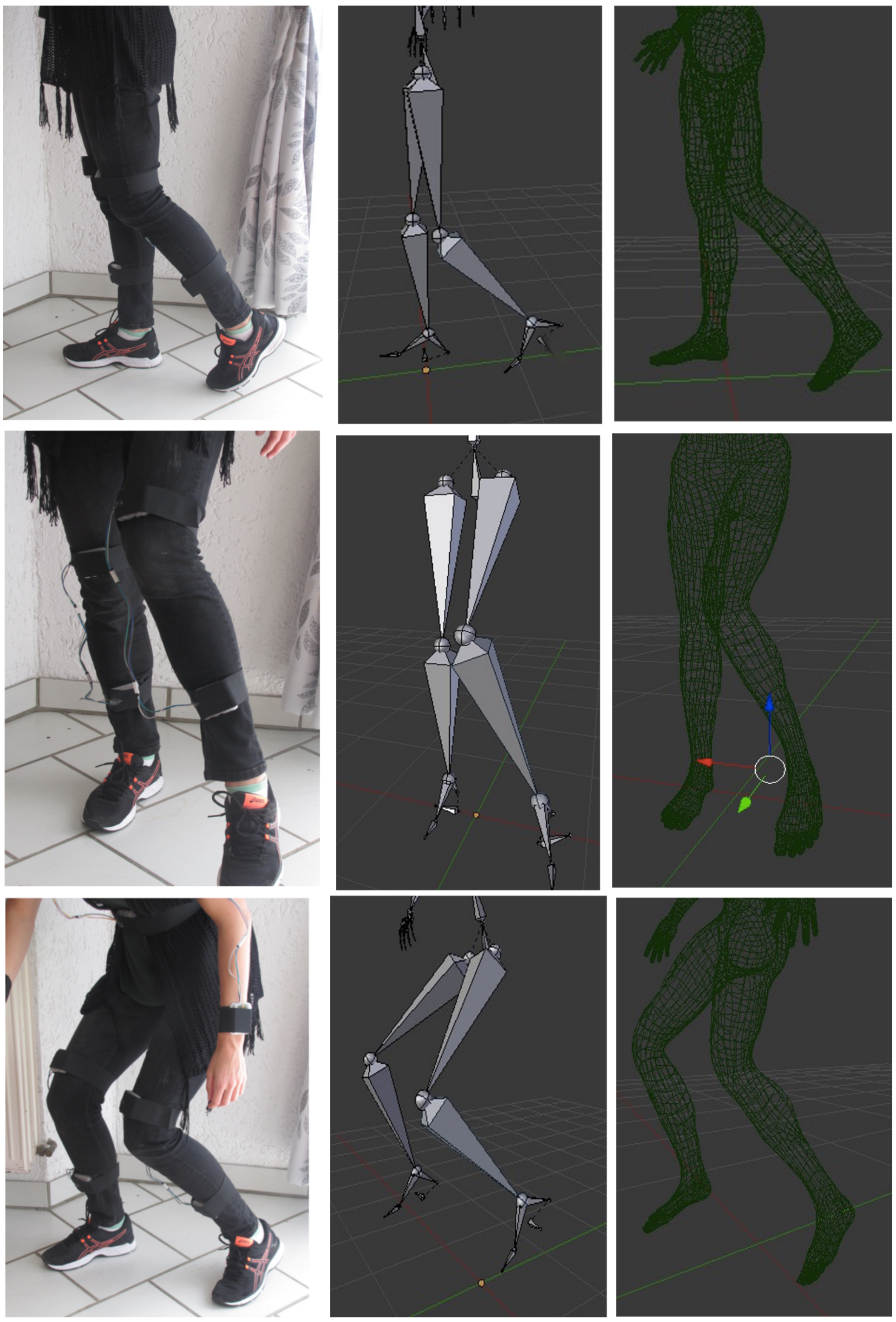


Figure 27: Results for different leg positions

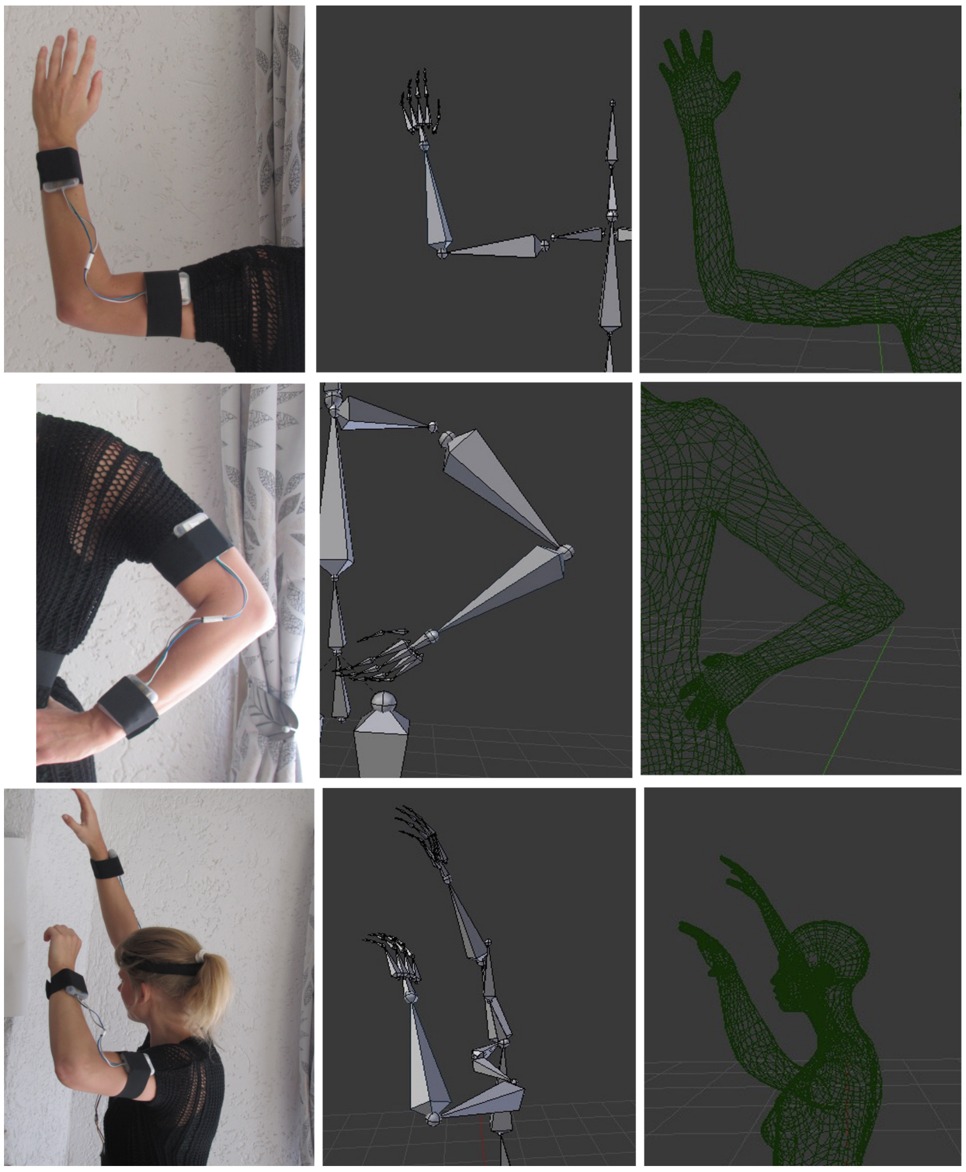


Figure 28: Results for different arm, body and head positions

- The system should not restrict he movement of the actor: Due to cables between every pair of sensors there is some kind of restriction, but on the other hand the system allows an actor to move freely in any given area of about ten meters around the receiving node (the computer here). In comparison to optical motion capture suits, the sensors are equally limiting, the free choice of a capture ground is an advantage, however.

- The calibration and general handling should be relatively easy: While putting on the build sensors requires some caution, this can not be generalized for professional systems, which use IMU technology. Calibration is done in a previous step and is not handled by the user. The user is only required to hold a T-pose while powering the sensors on, which takes up to thirty seconds.

- The measurements should be accurate (smaller and bigger movements are captured equally good): With the sources of errors described earlier in mind, the IMU readings are still fairly accurate and even recognize tiny movements. Wrong offsets due to differences in manufacturing were corrected manually. There are lags when handling too many sensors at the same time, however. Compared to optical motion capture, we get correct trackings, which do not suffer from occluding errors. Thinking about long term use of an IMU based system, the drift error can certainly become a problem, which optical systems do not suffer from.

- The pipeline for recording the data until having a complete animation should be very tight: As there is no manual preprocessing necessary when having precise calibrated sensors, reading sensor values and displaying the results in real time inside a 3D Engine is possible and was done in this project, thus after the recording is done one only needs to export the animation.

8 Conclusion and vision

To recap, the goal of this thesis was to answer the following questions: First, what differentiates the various motion capture systems, that are on the market or still in development, what are their individual strength and drawbacks? Second, what drives the IMU based approach to motion capture? Are the goals even the same as for classical motion capture? Third, is it possible to build ones own IMU based motion capture system that provides all the advantages, theory states it should? Is it affordable? Are the results comparable to professional motion capture, or at least on the same level as optical systems in the same price range?

All of these questions were answered throughout this thesis, the first two ones in the introductory chapters, the third one by building such a system as described in chapters 3, 4, 5 and 6. The advantages and disadvantages were discussed throughout these chapters too, as well as in chapter 7 while also the comparison to optical motion capture was drawn. In short:

Inertial measurement is a valid alternative to expensive optical systems for motion capture and was explored in this thesis. With IMUs, disadvantages of optical systems like occlusion, ghost markers and complex post processing, do not play a role any longer. On the one hand it provides an effective technique for real time motion capture, lacking the need for post processing after initial calibration entirely, while at the same time providing a higher range and flexibility of movement and usage in outdoor environments. On the other hand, IMUs come with their own set of difficulties, namely noise and especially drift. Determining a pose additional to an orientation is only sensible with additional sensors. Throughout this thesis it could be seen, that IMU based motion capture is generally more sensitive to errors, and not as accurate as optical motion capture. Furthermore tracking finger movement is a difficult task, and the tacking of facial expressions not yet possible given the size of todays technology.

For future versions of this motion capture systems, a better way of communication between the sensors and the host computer should be found, preferably something independent from outer conditions. Bluetooth has not proven to be a reliable technologies for this purpose. Wifi did also not prove reliable with the given hardware, but should be given more thought, especially with the new Rasperi Pi Zero W on the market for a few weeks now, which could be a valid replacement system.

Also, one has to be aware that boards like the Arduino Pro Mini, the MPU-6050 and the HC-05 used for this project are made for a wide range of usages and by no means specialized for the motion capture task. In a professional context, an approach for making the sensors way smaller, like a third of their current size is to use the very expensive version of button batteries. This would also require a specially manufactured hardware to recharge the batteries. Saving a lot of space would also include, to only use the core elements like the ATmega328 chip or the actual inertial sensor instead of using the whole boards. The pin-outs of this parts are well documented and it is even possible to use the Arduino boards to upload the sketches and later detach the microcontroller from the board. It is possible to solder them to custom circuit board, which are made by laser printers. This is what allows professional manufacturers to build sensors that small.

With this possible improvements, motion capture with inertial navigation techniques could become important for some industries in the future. When one thinks of the future of gaming, then virtual reality in a sense that players really become their characters, comes to mind. This includes moving a character around as one moves in reality as well as having a more realistic experience by ”feeling” things happening. To accomplish that purpose one would have to wear a special suit most definitely as well as a head mounted display. Tracking the players motion in that kind of scenario is predefined to use sensor based motion capture, also in regard to marketing, because in that case, if a player has to buy a suit anyway, the costs for adding additional tracking features to the suit would me marginal in comparison to buying a camera based system.

List of Figures

1 (a) Example of an optical motion capture studio.
(b) Example for IMU based motion capture [4] 4
2 Schematic representation of the process of capture motion [8] 7
3 (a) The Arduino Uno board [24].
(b) The Arduino Pro Mini board [24] 14
4 Structure of a singel chip microcontroller [25] 15
5 (a) Communication using a TTL adapter.
(b) Serial communication between two devices [28] 17
6 Diagram of I2C communication for multiple devices [28] . . . 18
7 (a) Axis orientation of the MPU-6050 chip [29]
(b) MPU-6050 chip integrated in the GY-521 module [24]. . . 20
8 Internal structure of MPU-6050 [29] 21
9 (a) Diagram of relevant network protocolls [35].
(b) HC-05 Bluetooth breakout board 22
10 (a) Circuit diagram of the Arduino to HC-05 connection.
(b) Depiction of a scatternet [38] 24
11 (a) Schematics of a voltage divider.
(b) Voltage divider set into context [39] 26
12 (a) Lithium battery charger and protecton module [43].
(b) First draft for the sensors 29
13 Final sensor 30
14 Circuit diagram of the whole sensor 30
15 (a) Atomic structure deformed under pressure [46].
(b) Piezolectic effect schematics [47] 33
16 Visualization of accelerometer operation [48] 33
17 (a) Axes orientation of an acceleromter [48].
(b) Classical gyroscope [49] 35
18 3D attempt to explain the operation of a gyroscope 37
19 A MEMS gyroscope [52] 38
20 Schema of sensor output and fusion [53] 40
21 Composition of the complemenatry filter 42
22 Comparisson of Complementary filter and Kalman filter [55] . 44
23 Transformation: MPU-6050 coordinate system to Blender local bone cooridnate system 47
24 Wrong tilting of the head due to movement of the body . . . 47
25 Raw output of the DMP [58] 52
26 Orientation transfered from prototype to 3D model, with comparisson between DMP and Calman filter 62
27 Results for different leg positions 63
28 Results for different arm, body and head positions 64

References

[1] P. zhan Chen, J. Li, M. Luo, and N. hua Zhu, “Real-Time Human Motion Capture Driven by a Wireless Sensor Network,,” *International Journal of Computer Games Technology*, vol. vol. 2015, no. Article ID 695874, 2015.

[2] M. Gleicher, “Animation from observation: Motion capture and motion editing,” *ACM SIGGRAPH Computer Graphics*, vol. 33, no. 4, pp. 51–54, 1999.

[3] L. Mündermann, S. Corazza, and T. Andriacchi, “The evolution of methods for the capture of human movement leading to markerless motion capture for biomechanical applications,” *Journal of NeuroEngineering and Rehabilitation*, vol. Volume 3, no. Number 1, 2006.

[4] P. Neuron, “Perception neuron.” Last visited: April 2017.

[5] A. von Koenigsmarck, “Full body motion capture with mocatch for maxon cinema 4d-users.” Last visited: April 2017.

[6] X. N. A. Inc., “Xsens.” Last visited: April 2017.

[7] N. I. Inc, “Wear notch.” Last visited: April 2017.

[8] M. Leman and R. I. Godøy, “Why study musical gestures,” *Musical gestures. Sound, movement, and meaning*, pp. 3–11, 2010.

[9] A. T. G. Wes Trager, “A Practical Approach to Motion Capture: Acclaim’s optical motion capture system,” *ACM SIGGRAPH ’94*, vol. Cours Notes: Character Motion Systems, 1994.

[10] D. J. Sturman, “A Brief History of Motion Capture for Computer Character Animation,” *ACM SIGGRAPH ’94*, vol. Cours Notes: Character Animation Systems, 1994.

[11] M. Field, D. Stirling, F. Naghdy, and Z. Pan, “Motion capture in robotics review,” in *Control and Automation, 2009. ICCA 2009. IEEE International Conference on*, pp. 1697–1702, IEEE, 2009.

[12] W. Li, Z. Zhang, and Z. Liu, “Expandable data-driven graphical modeling of human actions based on salient postures,” *IEEE transactions on Circuits and Systems for Video Technology*, vol. 18, no. 11, pp. 1499–1510, 2008.

[13] I. Damian, M. Obaid, F. Kistler, and E. André, “Augmented reality using a 3d motion capturing suit,” in *Proceedings of the 4th Augmented Human International Conference*, AH ’13, (New York, NY, USA), pp. 233–234, ACM, 2013.

[14] T. S. Bruggemann, D. G. Greer, and R. A. Walker, "Gps fault detection with imu and aircraft dynamics," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 47, no. 1, pp. 305–316, 2011.

[15] W.-W. Wang and L.-C. Fu, "Mirror therapy with an exoskeleton upper-limb robot based on imu measurement system," in *Medical Measurements and Applications Proceedings (MeMeA), 2011 IEEE International Workshop on*, pp. 370–375, IEEE, 2011.

[16] J.-N. Kim, M.-H. Ryu, Y.-S. Yang, and T.-K. Kim, "Upper extremity rehabilitation program using inertial sensors and virtual reality for patients with upper extremity hemiplegia due to disorders after stroke," in *Proceedings of International Conference on Computer Science and Technology (CST'12)*, pp. 71–76, 2012.

[17] N. C. Perkins, "Electronic measurement of the motion of a moving body of sports equipment," June 26 2007. US Patent 7,234,351.

[18] K. King, S. Yoon, N. Perkins, and K. Najafi, "Wireless mems inertial sensor system for golf swing dynamics," *Sensors and Actuators A: Physical*, vol. 141, no. 2, pp. 619–630, 2008.

[19] T. M. Hon, S. A. Senanayake, and N. Flyger, "Biomechanical analysis of 10-pin bowling using wireless inertial sensor," in *Advanced Intelligent Mechatronics, 2009. AIM 2009. IEEE/ASME International Conference on*, pp. 1130–1135, IEEE, 2009.

[20] S. L. L. H. Zhang M., Hol J.D., "Second order nonlinear uncertainty modeling in strapdown integration using mems imus," in *Proceedings of the 14th International Conference on Information Fusion*, Vol. 1, pp. 1679–1685, ACM, 2011.

[21] D. Vlasic, R. Adelsberger, G. Vannucci, J. Barnwell, M. Gross, W. Matusik, and J. Popović, "Practical motion capture in everyday surroundings," in *ACM transactions on graphics (TOG)*, vol. 26, p. 35, Acm, 2007.

[22] M. Loper, N. Mahmood, and M. J. Black, "Mosh: Motion and shape capture from sparse markers," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 6, p. 220, 2014.

[23] H. Barragaxn, "The untold history of arduino." Last visited: April 2017.

[24] A. Foundation, "What is arduino?." Last visited: Mai 2017.

[25] M. Shaaban, "Microcontroller basics," 2000.

[26] U. o. M. W. Durfee, "Arduino microcontroller guide," 2011.

[27] J. B. Cardell, “Overview of microprocessors,” 2014.

[28] S. Electronics, “I2c.” Last visited: Mai 2017.

[29] InvenSense, *MPU-6000 and MPU-6050 Product Specification*, 8 2013. Rev. 3.4.

[30] O. J. Woodman, “An introduction to inertial navigation,” Tech. Rep. 696, University of Cambridge, 2007.

[31] I. Skog and P. Händel, “Calibration of a mems inertial measurement unit,” in *XVII IMEKO World Congress*, pp. 1–6, 2006.

[32] Memsnet, “Micro-electro-mechanical systems.” Last visited: Mai 2017.

[33] D. Titello, “Arduino guide using mpu-6050 and nrf24l01,” tech. rep., Advanced Institutes of Convergence Technology, 2015.

[34] Debra, “Mpu-6050 redux: Dmp data fusion vs. complementary filter,” 2014. Last visited: Mai 2017.

[35] A. Malik, “Nfc vs bluetooth vs wifi direct: Comparison, advantages and disadvantages.” Last visited: Mai 2017.

[36] Netplanet, “Drahtlose lokale netzwerke.” Last visited: Mai 2017.

[37] A. R. C.S.R. Prabhu, *Bluetooth Technology and its applications with Java and J2ME.* New Dehli: Prentice-Hall of India, 2007.

[38] Researchgate, “Scatternet topology.” Last visited: Mai 2017.

[39] Sparkfun, “Voltage dividers.” Last visited: Mai 2017.

[40] Apple, “Connect multiple bluetooth devices to one computer.” Last visited: Mai 2017.

[41] P. Schnabel, *Kommunikationstechnik-Fibel, Grundlagen, Testnetz, Mobilfuktechnik, Breitbanktechnik, Netzwerktechnik.* Ludwigsburg: Books on demand GmbH Norderstedt, 2003.

[42] M. Schwartz, “How to run your esp8266 for years on a battery,” 2017.

[43] J. Groot, “State-of-health estimation of li-ion batteries: Cycle life test methods,” 2012.

[44] D.-K. N. A. Editors, “A designer’s guide to lithium (li-ion) battery charging.” Last visited: Mai 2017.

[45] D. Meschede, *Gerthsen Physik.* Springer Lehrbuch, 2001.

[46] F. Cengic, “Mmb2 kraftmessung.” Last visited: Mai 2017.

[47] M. Weber, “Piezoelektrisches prinzip.” Last visited: Mai 2017.

[48] A. Sanjeev, “Imu interfacing tutorial.” Last visited: Mai 2017.

[49] C. T. Pearson, “How a gyroscope works.” Last visited: Mai 2017.

[50] starlino, “A guide to using imu (accelerometer and gyroscope devices) in embedded applications.” Last visited: Mai 2017.

[51] A. A. Trusov, “Overview of mems gyroscopes: history, principles of operations, types of measurements,” *University of California, Irvine, USA, maj*, 2011.

[52] jadmin, “Gyroscope.” Last visited: Mai 2017.

[53] N. Ahmad, R. A. R. Ghazilla, N. M. Khairi, and V. Kasi, “Reviews on various inertial measurement unit (imu) sensor applications,” *International Journal of Signal Processing Systems*, vol. 1, no. 2, pp. 256–262, 2013.

[54] G. Welch and G. Bishop, “An introduction to the kalman filter,” 1995.

[55] P. Chen, Y. Kuang, and J. Li, “Human motion capture algorithm based on inertial sensors,” *Journal of Sensors*, vol. 2016, 2016.

[56] W. T. Higgins, “A comparison of complementary and kalman filtering,” *IEEE Transactions on Aerospace and Electronic Systems*, no. 3, pp. 321–325, 1975.

[57] S. Grzonka *et al.*, “Mapping, state estimation, and navigation for quadrotors and human-worn sensor systems,” 2011.

[58] S. B., “Calibrating and optimising the mpu6050.” Last visited: Mai 2017.