DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

Soccer Tactics Simulation: User Interface Using an EMG Armband

Kağan Batuker

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

Soccer Tactics Simulation: User Interface Using an EMG Armband

Fußball-Taktik Simulation: Nutzschnittstelle mithilfe eines EMG Armbands

Author:	Kağan Batuker
Supervisor:	Prof. Gudrun Klinker
Advisor:	M.Sc. Sandro Weber, Dipl.Inf. David Plecher
Submission Date:	17.07.17

I confirm that this bachelor's thesis in Informatics: Games Engineering is my own work and I have documented all sources and material used.

Munich, 17.07.17 Kağan Batuker

Acknowledgments

I would like to thank all the people who made this project possible. First of all I would like to thank Professor Gudrun Klinker and the Chair of Computer Aided Medical Procedures and Augmented Reality for offering me this thesis. Second, I thank my advisor Sandro Weber for his willingness to provide such a project and his support throughout my work. Last of all I would like to thank Thalmic Labs for providing the source code for the Myo software and an easy-to-read Unity3D API.

I also would like to thank all the people below who participated in the user study and helped with various aspects of my work wihtin four months (in alphabetical order):
Okan Agca
Larissa Akcetin
Alp Danisman
Clemens Fromm
Onur Kilimci
Özge Kilimci
Konstantin Kirilov
Waltentin Lamonos
Kivanc Mertek
Alex Müller
Felix Novoa
Ozan Pekmezci
Daniel Schroter
Berkay Soykan
Kagan Tunca
Oktay Turan
Katarina Weber
Baris Yolsal

Abstract

As the technology goes further, new methods of human-computer interaction are also developed. Today, haptic devices in this field are commonly used and show promise for intuitive and effortless interaction. In this regard, an EMG armband called Myo is studied in this project. The goal is to implement interaction features for Myo so the device can be easily integrated into possible virtual simulations and similar games, starting with the "Soccer Tactics Visualization", a big-scaled project also done in the Chair of Computer Aided Medical Procedures and Augmented Reality in the Technical University of Munich.

In this project, three features for the usage of Myo armband are implemented with the help of the Unity3D Engine and the Unity Myo API provided. Then, a user study is carried out to test the features and figure out their performance in terms of comfort, intuitiveness and reliability. The user study shows that the features are functioning up to an extent where they can definitely be integrated into future projects with room for improvement, both in terms of feature implementation and Myo armband's accuracy.

Contents

1 Introduction

An introduction to the subject of this thesis is made in this chapter. The structure of the thesis will be explained first, followed by the motivation of the thesis.

1.1 Structure

After this chapter, a research consisting of related works and a literature review is outlined. The next chapter focuses on the main catalyst of the thesis, the Myo armband and it's capabilities. The fourth chapter handles the implementation of the thesis; as some obligatory topics for the application, the used API and the realized features will be explained in detail. The fifth chapter discloses an user study, in which testers evaluate the functionality of the implementation. At last, a conclusion is made with the things that could be made better and possible future work on this area.

1.2 Motivation

As technology expands it's borders everyday, different forms of interaction also surface to replace the classic combination of keyboard and mouse. The developments in the last decade were mainly focused, but not limited to, finding new methods for users to interact with their PC, cell phones, consoles or any electronic device.

The rise of the touchscreens and haptic feedback triggered this new wave of change at first. The flexibility it brought made video games and electronics more accessible, as every user felt it was more intuitive to use their own fingers for control rather than a indirect interaction through a screen via the old keyboard and mouse. The exploding demand for the mobility led to more research to this subject, drawing out large amounts of investment and marketing.

Today concepts of virtual reality(VR) and augmented reality(AR) are spearheading this trend. Products like Kinect and Leap Motion are all developed to improve the immersion and flexibility for the consumer. Google Glass, Oculus Rift and HTC Vive try to mold the visual sensors with virtual worlds.

While it seems that tactile interaction has not improved that much, it seems to me that it is one of the most comfortable methods of interaction. Already proven to be

better than mice, devices working with touch input bring a different kind of mobility to the relation of the user and device: Reliable unlike voice control, comfortable because of the fact that the user does not need to stand in front of a camera. The possible improvements on motion capture and gesture control makes this kind of interaction a powerful tool with a promising future.

The armband Myo, used for gestural control through haptic feedback, is the center of this written work. In the Chair of Computer Aided Medical Procedures and Augmented Reality, a big scale project of visualizing soccer tactics is ongoing and this project seeks to develop an interface for the team manager to effortlessly interact with the simulation. Through the user interface developed using Myo armband, the manager can intuitively explain tactics and communicate with players. This interface could be the first step in integrating such haptic devices into bigger games or simulations by using the pattern that is proposed in this written work.

2 Related Work

This chapter involves the research done to familiarize with the theme. It consists of the definition for related publications and similar works put out in the field of haptic feedback and Myo usage in various platforms.

2.1 Related Work

The first research topic would be the accuracy of EMG (electromyography) sensors in the field, and how the data is converted into different classes of gestures, along with encountered difficulties in the process.

Since Myo armband belongs to the family of haptic feedback devices, similar devices can be counted as research interests. Other devices in the field of virtual and augmented reality will also be investigated since there are lot of works that use Myo and such devices together as an input provider. Another point of interest can be a comparison between Myo and other VR/AR devices in terms of comfort, reliability and accuracy.

To understand the potential of Myo armband and draw hints and inspiration for the implementation, other areas of usage regarding Myo should also be surveyed. These areas consist of, but are not limited to, medical surgery, cross-application usage, music production and robotics.

2.2 Literature Review

Since Myo is a fairly new released product, scientific papers that mention it or use it are scarce. Still, a number of papers provide insight about the armband's boundaries and utilization.

The development in the EMG classification research field is one of the most important aspects for the functionality of Myo. Reading EMG data and classifying it in real time is how Myo operates so effectively. In this field, relevant research goes back to 1989, where surface EMG classification was tried through neural networks to map five finger movements. The goal was to reduce the physical and mental effort of prosthetics users[12]. Other methods for classification include similar approaches like spatial filtering algorithms combined with linear discriminant analysis (LDA)[14], LDA used

with support vector machines (SVM)[3], SVM combined with signal-based wavelet optimizations[18], fuzzy classification with basic isodata algorithms[6] and at last, cascaded architecture of neural networks with feature maps (CANFM)[13]. Among these implementations, most of them work in similar fashion, trading computation power and accuracy with simplicity and speed. Overall, the differences in results are minor.

One interesting research is the investigation of the accuracy of Myo using an unique data acquisition interface to gather surface EMG data. The data is then tested with a handful of standard gestures[1]. The results show that Myo is still not perfect: Although EMG and IMU data are processed with good accuracy, classification errors still occur. Another interesting point is that two Myo's were tested against each other and the uncertainty in the signals between the two was estimated to be $\pm 2\%$.

Some of the problems faced when using surface EMG as a data accumulator are gaps between the EMG sensor and the skin, humidity, temperature, hydration, body fat and perspiration levels. In [32], the effect of the temperature when measuring EMG data is investigated. The outcome specifies that the signal amplitude is doubled and average signal level increased in cooler temperatures, meaning that temperature is a deciding factor. Another paper ([21]) on the influence of body fat states that the distance of the electrode and the muscle caused by body fat leads to a data variance up to 68%, reduced with maximal contraction of the muscle.

Figure 2.1: One of the haptic devices, Cybergrasp(1998).[4]

In the field of haptic feedback hardware, the first paper I found (maybe the most accumulative article till it's publishment) sums up the history of haptic devices until the 1990's and discusses the usability of haptic feedback in the future ([4]). Emerging an late 1960's, this field was really underdeveloped in comparison to visual and auditory modalities until the 1990's, where it became popular again. Nevertheless, devices with haptic feedback were not that comfortable and not so visually appealing in comparison to the modern ones (see figure 2.1).

A paper ([5]) directly makes a comparison of mid-air gestures between 3 platforms: Kinect, Leap Motion and Myo. Results indicate that most common gestures in these platforms' applications are cursor movement, waving and swiping, followed by air taps and rotations. It is also specified that translation of touch-based elements to mid-air interaction is very common and will continuously be used in standard sets.

Many researchers tried to combine the powers of other devices with the mobility of Myo and compensate for their weaknesses on both sides. Publishers of [25] devised a prototype to decrease time for prosthetic training by using an Oculus Rift, a Kinect and a Myo at the same time, with the end goal of making prosthetics more accessible for amputees.In another paper([30]), Myo and Leap Motion are used together for full arm tracking. According to the paper, Myo's sensor limits can be overcome through merging the sensor data with data coming from a Leap Motion device and using a Kallman filter algorithm to combine data with peak performance.

Other variations of a Myo armband being used as a robotics control device include a paper([19]) in which a robot designed for household necessities is controlled in this way. The same methodology is used in [29] to control a virtual robotic arm. At last, a rover with 5 degrees of freedom is commanded via Myo again, with the help of a Leap Motion device, in [20]. Through the use of this research, skilled operators can be replaced by this interface, which provides equal dexterity and mobility when handling the rover designed for exploration and rescue.

In the field of medical surgery, Myo's haptic capabilities are presented as a solution to improve the lack of haptic feedback in surgeries with minimal robots ([23]). The argumentation is that tactile feedback improves the surgeon's accuracy and helps him/her understand how much force is applied through the robot.

As opposed to previous fields, Myo is also explored as a NIME(New Interfaces for Musical Expressions) device for producing music in [22]. First, some tests are carried out about noise levels and data reliability. After that, an interface to create electronic music is tested out using different combinations of Myo's spatial and gestural data.

2.3 Literature Critic Overview

Through all the articles, publishers also debate the advantages and disadvantages of using Myo for their projects, reflecting this on their results. While most of them find Myo mobile, dexterous and very intuitive to use, shortcomings of Myo are also noted.

A paper is just dedicated to a survey done for Myo in context of merging it with a navigation application([28]). The survey informs that although Myo is interesting, socially acceptable and easy to use, users mainly think that it is unnecessarily complex, needs a learning process and has some inconsistencies when perceiving gestures. Users also complain about the muscle fatigue when using Myo for a significant amount of time.

Reports of misclassification of gestures happens to be on other papers as well ([22] and [30]), while in [19] signal acquisition and processing is praised.

Another issue is the limited amount of gestures that are recognized by Myo, as indicated in [22] and [19]. A solution to this in the last article was to combine positional and gestural data to have different outcomes(figure 2.2). Thalmic Labs have also stated that they will be adding new gestures to Myo in the near future, so it is only a temporary problem.

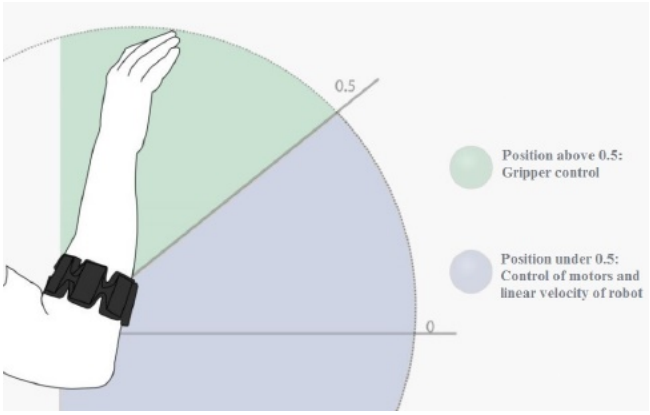

Figure 2.2: The article [19] demonstrates how to combine spatial and gestural data.

A last major critic about Myo would be users getting tired of making gestures over and over again ([28] and [19]). Apparently users need to perform a gesture so strictly that it will be recognized by Myo. Improving the machine learning classifiers would not only improve the accuracy of the device, it would also make it easier to perform the gestures, making it more comfortable for the end user. It is also suggested in [1] to show users how much muscle tension they are applying so that they have an understanding when gestures are recognized.

3 Myo

3.1 Myo in a Nutshell

The device that will be used in this thesis, is an EMG armband named "Myo"(figure 3.1). It is a wearable gesture control device designed by Thalmic Labs for allowing user interaction through the spatial and gestural signals sent from a person's forearm. These signals are acquired with 8 EMG sensors attached to one another via a flexible band that holds these parts together. When the brain commands the arm to move, it sends electric signals through the neurons to the arm and hand muscles. EMG sensors detect these signals that move through the forearm and translate it to data.The data Myo receives is then processed with machine learning to determine gestural commands.[27]

Figure 3.1: The Myo armband. Source:[33]

3.2 Myo Gesture and Motion Capabilities

As said above, Myo reads gestures and poses using eight medical grade stainless steel proprietary electromyography (EMG) sensors. The sample rate of these sensors is 200 Hz. Other than that, Myo includes a motion sensing unit using a 9-axis inertial

measurement unit, consisting of a gyroscope (3-axis)[1], an accelerometer (3-axis)[2] and a magnetometer (3-axis). Those three parts are also called the inertial measurement unit (IMU) together. Myo is capable of pulling IMU data at a sample rate of 50 Hz, measuring the motion, orientation and rotation of the forearm. This gives Myo an out-of-the-box functionality since using Myo to interact becomes simple and intuitive, because the device reacts to specific gestures and the orientation of the arm ideally[24].

The Myo armband has been made to work best at the widest part of the forearm, which is the upper forearm. This way the sensors touch the skin without any gaps. A syncing gesture is always made when wearing Myo for the first time, as it trains the armband according to the user's arm muscles.

There are five hand gestures that Myo can recognize. These poses are a fist, a wave-in gesture, a wave-out gesture, spreading fingers and double tapping. Along with Myo's ability to identify the arm's rotation and pan, the overall control Myo can give a user is definitely noticeable (see figure 3.2).

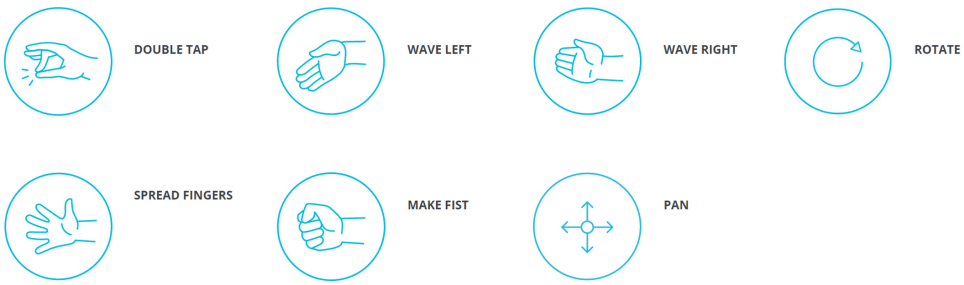

Figure 3.2: The gestures for Myo. Source: [33]

The logic behind the production of such an armband was mostly mobility. The products previously in the market or that were announced at the time were not satisfactory for the developers at Thalmic Labs. Mouse and keyboard interaction is already deemed "old-school", and other solutions were also not working for them either: Touch interaction is not that mobile and it does not work with gloves. Other wearables like glasses are difficult to use in noisy environments: The wearable space is not reliable because of voice control and it can be socially awkward at times. Lastly, motion tracking devices such as Kinect require the person to face the camera all the

[1] A gyroscope is a device with a spinning disc or wheel mechanism that harnesses the principle of conservation of angular momentum.[11]

[2] An accelerometer is a device that measures changes in gravitational acceleration in a device it may be installed in. Accelerometers are used to measure acceleration, tilt and vibration in numerous devices.[31]

time[9].

However, the Myo armband is better suited for determining the relative positioning of the arm rather than the absolute position, a consideration to be aware of when applying pattern recognition algorithms and developing programs for Myo[1].

Figure 3.3: The logo of Thalmic Labs, creators of Myo. Source:[16]

3.3 Areas of Usage

There are already several uses to Myo armband in very differing disciplines, which proves that Myo is indeed intuitive and easy to learn and use.

- **Presentation:**Through an API a person can control presentation software such as Powerpoint or Keynote with ease. Going through slides and pointing are among the usabilities Myo offers.
- **Gaming:** As Myo can be paired with a multitude of software, it can also be integrated into games on PC, iPhones and Android phones.The spatial control Myo provides opens new ways of playing a game that leads to totally unalike user experiences.
- **Entertainment:** Armin Van Buuren, one of the most famous electronic dance music DJs, uses two Myo's attached to each arm to control the music and stage shows during his live concerts for a more immersive performance (see figure 3.4).
- **Device Control:** Various drones and devices like Sphero can be controlled intuitively with Myo.
- **Prosthetics:** At John Hopkins University, two Myo armbands worn on the upper arm of patients are used to control a prosthetic arm attached directly to their skeleton with very promising results.

- **Medical Imaging:** Spanish surgeons are using Myo armbands to navigate medical slides whilst performing surgery. The software is developed by the medical imaging firm TedCas, and Myo is in the center of the innovation, supported by medical cameras and voice recognition.
- **Sign Language:** Researchers at Arizona State University are using Myo in collaboration with a highly sensitive motion sensor to translate American Sign Language on a computer screen. They believe the software can accelerate the communication for the hearing impaired and those who do not know the language[17].

The sections above are a glimpse of what Myo is capable of. There are still emerging software that use Myo as an important aspect within the project. There are also other applications built for Myo that explore other ways of utilization. These can be found on this website [15] run by Thalmic Labs.

Figure 3.4: Armin Van Buuren using Myo's at a live show. Source:[16]

4 Implementation

4.1 Proposed Work

Our main goal is, as explained in Chapter 1, is to implement the Myo Armband to the Soccer Tactics Simulation for the manager to be more comfortable and focused on his work by providing a better user interaction. However, another goal of this work is to show that this interface can easily be implemented to other projects and used efficiently. So it can be developed independently of the main simulation. Three main features are experimented with, which also could fit in with similar projects.

4.1.1 Approach Taken

With regards to the proposed work, a separate scene is used to test and implement the features and their utilization to provide the independence. The Unity3D API published by Thalmic Labs is at the foundation of the whole implementation. It is the bridge between the raw data coming from the electrical readings and the functionality in the programming language. The data is converted through the API to functions and variables, ready to use and updated in every frame. Every feature and it's components draws the necessary input data from this base.
The implemented features are as follows:

- Drawing on a 2D board with a virtual cursor.
- Moving the camera within a field.
- Selecting players with the help of a circular menu.

The user can choose between different features through an UI and can switch between them with a specific gesture that can be interpreted by Myo. For the last feature (circular menu), two different approaches are tried out: One with an relative menu with respect to the orientation of the arm, and the other with an absolute menu with fixed angular slots.

4.1.2 Requirements

Software Requirements

Two major software are used in this project. The first one is the Unity3D Engine, providing a platform to implement and run the project using the programming language C#. The second software is Myo Connect which is used as the accumulator of data from the Myo device to the computer. Connection is established via Bluetooth.

Hardware Requirements

As for hardware requirements, a decent PC which can run Unity and Myo Connect with ease is needed. A PC below the standards can lead to ansyncronized data transfer between Myo and Unity3D, which deteriorates the usability and comfort of the project.The minimum requirements to be able to run this project without problems are a GPU supporting at least DirectX9(shader model 3.0) and a SSE2 instruction set support within the CPU.

4.2 Myo Unity3D API and Fundamentals

In order to understand how the designed features work, the Unity3D API for Myo and how Myo interacts with it as well as a basic proprieties of Myo must be explained.

4.2.1 Quaternions

The orientation of the arm is stored in so-called "Quaternions": A number system that extends the complex numbers.It can be defined as the quotient of two vectors[10, p. 34].

There are several ways to define orientation in programming such as Euler angles or a rotational matrix. However, both ways have singularity problems when defining an angle (such as a gimbal lock when using Euler angles), which means that not all angles have a unique representation, leading to programming errors.A quaternion does not have this problem, however it is hard to understand it at the first glance.

A quaternion, as opposed to a normal 3D-vector, is defined as follows:

$$q = s + xi + yj + zk \qquad s, x, y, z \in \mathbb{R}$$

Where s is a constant, x, y and z are the respective axes in the three dimensional plane and i, j and k are complex numbers with

$$i^2 = j^2 = k^2 = ijk = -1 \qquad i, j, k \in \mathbb{C}$$

It can be seen that an unit of quaternion has four elements within itself. As it is common with the complex numbers, the first part ("s") is defined as the real part (also called the scalar part), and the rest of the elements are defined as the imaginary part(also called the vector part) of one unit. The quaternion can be split up to these two parts like this:

$$q = (r, \vec{v}) \qquad r \in \mathbb{R}, \vec{v} \in \mathbb{R}^3$$

Following this notation, addition and multiplication with quaternions are as follows (with "·" as the dot product and "×" as the cross product):

$$(r_1, \vec{v}_1) + (r_2, \vec{v}_2) = (r_1 + r_2,\ \vec{v}_1 + \vec{v}_2)$$
$$(r_1, \vec{v}_1)(r_2, \vec{v}_2) = (r_1 r_2 - \vec{v}_1 \cdot \vec{v}_2,\ r_1\vec{v}_2 + r_2\vec{v}_1 + \vec{v}_1 \times \vec{v}_2)$$

It should be noted that the arithmetic operation of multiplication with quaternions is not commutative, leading to more distinctive solutions.

Due to the quaternion's nature it has to be normalized after every operation, because floating-point precession errors will cause it to be not unit length. The normalization procedure is the quotient of the quaternion and it's norm (square root of the product of a quaternion's elements.) A quaternion of norm one is thus named a "unit quaternion"(q_n).

$$q_n = \frac{q}{\|q\|}$$

Due to using complex numbers, interpreting quaternion values is an extremely tiring task. Because of that, we use quaternions to define the spatial location and rotation of an object, and convert it every time so specific values that can be understood much more simply. These values are explained in the next section.

4.2.2 Yaw, Pitch and Roll

Since quaternions do not offer much usability for human understanding, they are converted to Euler angles in every updated frame and then used accordingly. The Euler angles are three angles found by Leonhard Euler to describe the orientation of a rigid body with respect to a fixed coordinate system[8]. Euler angles are defined by three elemental rotations, namely rotations about the axes of a coordinate system.

There are different ways to calculate Euler Angles, as the order of equations for finding the final rotation are different in each part. In this project, the definition of yaw, pitch, and roll are used because of the comprehensibility: Originally named the Tait-Bryan Angles in flight dynamics and used in aerospace applications, in which

zero degrees elevation means the horizontal attitude. In Tait-Bryan angles, every angle represents it's own distinctive axis whereas (proper) Euler angles use the same axis(mainly the z-axis) for the first and second angle calculation.

If we define the coordinate system so that the x-axis points forward, z-axis to the right and y-axis downward with angles defined for left-handed rotation, we get:

- Roll (ϕ) : rotation about the x-axis(longitudinal axis in aerospace)
- Pitch (θ) : rotation about the z-axis(lateral axis in aerospace)
- Yaw (ψ) : rotation about the y-axis(perpendicular axis in aerospace)

An example using an aircraft can be seen here in figure 4.1:

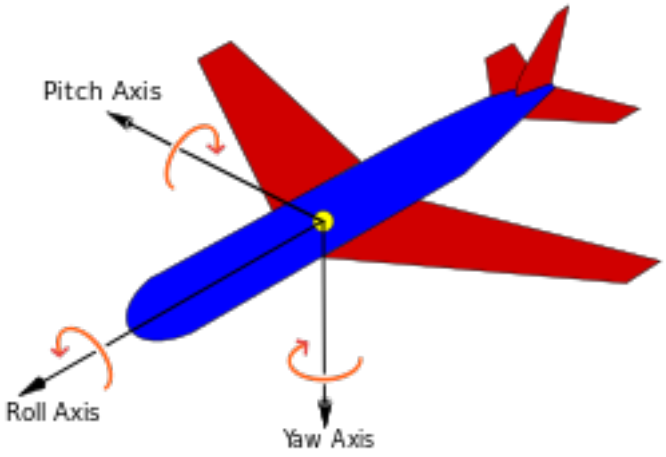

Figure 4.1: The roll, pitch and yaw axes represented on a plane. Source:[2]

Quaternions can be converted into these three angle representations through a series of equations as follows[26]:

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} atan2(2(s*x+y*z),(1-2(x^2*y^2)) \\ asin(2(s*y-x*z)) \\ atan2(2(s*z+x*y),(1-2(y^2*z^2) \end{bmatrix}$$

Note that s,x,y and z stand for the single elements of a quaternion.
It should be noted that atan2() function is used here instead of arctan() function.The base functions in C# only produce results between $\frac{\pi}{2}$ and $-\frac{\pi}{2}$, and for our three rotations it is not sufficient, because all orientations can not be acquired.

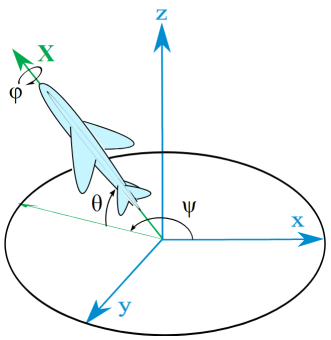

Figure 4.2: The roll, pitch and yaw angles with respect to the coordinate axes. Source:[7]

4.2.3 Gestures and Locking Policy

When using Myo, gestures play a vital part in the control mechanism. But in order to improve performance and prevent users from making unwanted function calls, locking is introduced. Simply put, the Myo unlocks itself when performing a gesture (the type of unlocking can depend on the code) for a given time so that the gesture's functionality is not interrupted by another gesture done right afterwards. It is also useful to lock Myo and avoid unnecessary commands when it is not needed, depending on the context.

How the unlocking works wihtin this project can be seen in subsections 4.2.7 (for different types of unlocking) and 4.3.1 (the function used for extending the unlock). Thalmic.Myo.Pose enumerations are explained in detail in subsection 4.2.7.

4.2.4 Unity3D Environment

Unity3D comes with many strengths and built-in classes that make building a game/simulation so much easier. The project is made up of "scenes", which can be seen as different levels within a game or a simulation. Every object that finds itself in this scene are named hierarchically as "GameObjects". These GameObjects can be anything from a cube to a point light or a placeholder for Myo, in our example. The scene is percepted through cameras, -another type of GameObject- which can be installed in numbers, but there is only one main camera. The main camera is always the one that's being used at the start of a scene. Additionally there are specific elements for the user interface(UI) stored in a Canvas GameObject. This UI is employed to navigate between different implemented features.

In the project there are 2 crucial GameObjects that make everything work.The first one is "Hub - 1 Myo" and it works as a connection between the Unity3D and Myo Connect. It must have a ThalmicHub.cs script attached to it.The second one is "Myo",

the representative of the armband. It must have a ThalmicMyo.cs script attached to it. The Myo GameObject should be nested under the hub, the so called "parenting" in the Unity3D environment (see figure 4.3). These scripts will be explained in detail below.

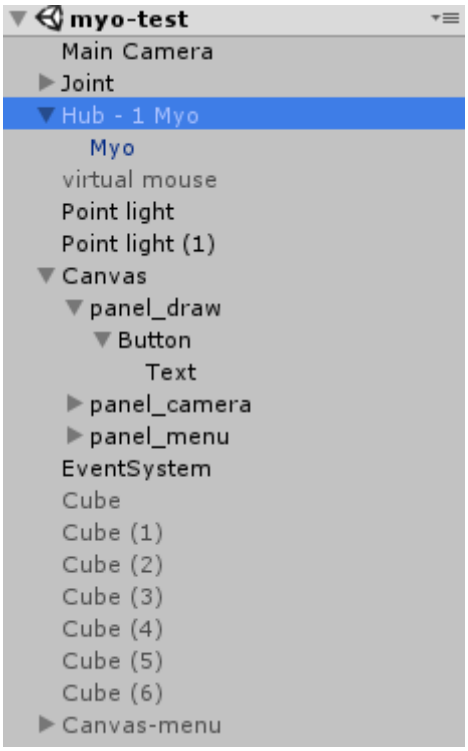

Figure 4.3: A basic object hierarchy in Unity3D.Note that "Hub - 1 Myo" is the parent of "Myo", as shown with the small arrow beside it.A child can also have other children, such as in the example of "Canvas"/"panel_draw"/"Button"/"Text".

In addition to these, another GameObject is very helpful in the overall calculation, namely the "Joint" (see figure 4.4). Every time Myo is connected, it will initialize with a random rotation. So in order to send meaningful signals through Myo, Myo's orientation should be turned into an orientation corresponding to the user's arm. This will be handled in myo_base.cs script in chapter 4.3.1.

It should also be noted that Unity3D has some basic functions within it's scripting API that are always used throughout the project.These functions and their connection to Myo readings should be elaborated fisthand.

- **Update():** This function gets called in every frame of the simulation. So calculations regarding the orientation of Myo and related work are always done in the Update() function. Spatial events of Myo also occur in sync with this function.

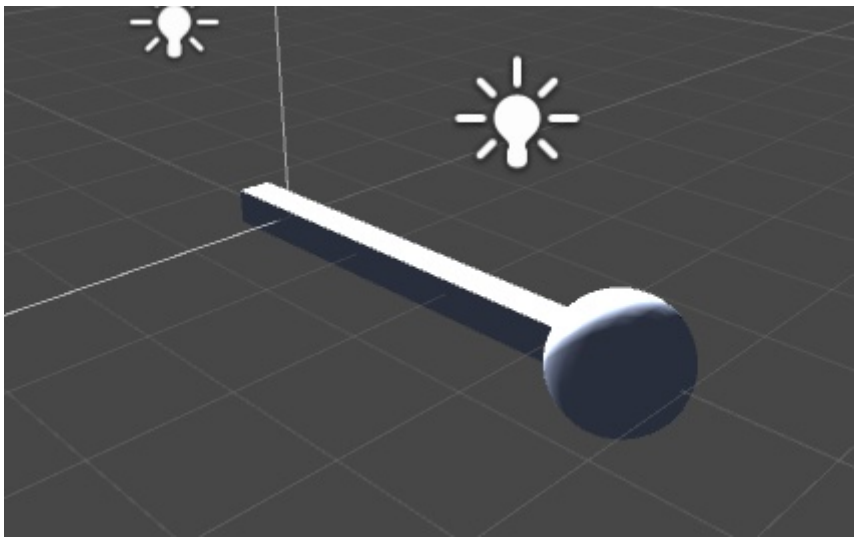

Figure 4.4: The GameObject simulating the user's arm. It consists of this hierarchy: "Joint" is the whole object, while it is split up into two children: "Stick" for the arm and "Hand" for the hand.

To identify a gestural event, it is again constantly checked in the Update() function if the user is making a hand pose.

- **Start():** This function is called at the start of the scripting instance, but it is only called if that particular script is enabled. It delays any initialization code that is not needed just at the start, as doing such would burden the engine.
- **Awake():** Awake() is always called before any Start() functions. Any code that has to be initialized immediately is located here.

4.2.5 Myo API: ThalmicHub.cs

ThalmicHub.cs allows access to one or more Myo armbands, which must be immediate children of the GameObject this script is attached to.

The ThalmicHub script is a singleton; if more than a single GameObject with a ThalmicHub attached is created, all but one will be destroyed. The single instance of ThalmicHub can be obtained using the ThalmicHub.instance property. The ThalmicHub GameObject and its children will not be destroyed when a scene change occurs through the Unity function DontDestroyOnLoad().

The Hub connects with all his children (Myo's) and ensures the Singleton structure in the Awake() function. It also constantly checks if the Myo is connected and updates

Unity with functions like ResetHub() and createHub(). It deletes the instantiated Hub when Unity is closed, so any data or changes to Myo is reset with it.

4.2.6 Myo API: ThalmicMyo.cs

This script represents a Myo armband. It has these attributes:

- bool **armSynced:** True if Myo is detected on an arm.
- bool **unlocked:** True if Myo is unlocked.
- Thalmic.Myo.Arm **arm:** The current arm that is being worn on.
- Thalmic.Myo.xDirection **xDirection:** The current x-axis direction relative to the user's arm.
- Thalmic.Myo.Pose **pose:** The current pose detected by Myo.
- Vector3 **accelerometer:** Myo's current accelerometer reading, representing the acceleration due to force on the Myo armband in units of g($= 9.8m/s^2$).
- Vector3 **gyroscope:** Myo's current gyroscope reading, representing the angular velocity in each of Myo's axes in degrees per second.
- bool **isPaired:** True if Myo is paired successfully, after which it begins to send data and a connection will be maintained.

These attributes are employed on every frame change to detect changes with Myo and to catch events like a new pose, Myo being disconnected etc.

Moreover, it has some useful and simple functions implemented like these below:

- **Vibrate():** Causes the Myo to vibrate.There are three different vibration types which vary on vibration length.
- **Lock():** Causes the Myo to re-lock immediately.
- **Unlock():** Cause the Myo to unlock with a specific type, e.g. UnlockType.Timed or UnlockType.Hold. More on these types on the section 4.2.7.
- **NotifyUserAction():** Notify Myo that an user action was recognized, such as a pose or a spatial change.

4.2.7 Myo API: Thalmic.Myo Namespace

The scripts in the Myo/Scripts/Myo.NET folder provide non-Unity3D-specific .NET bindings for the armband, which are stored in the Thalmic.Myo namespace. They can be accessed in Unity simply by referencing them above the class parenthesis and using them as the declared variable. Such an example would be:

```
1 //declaring "Pose" variable
2 using Pose = Thalmic.Myo.Pose;
3 //using the declared variable effortlessly in class parameters
4 Pose p = Pose.Fist;
```

libmyo.cs

This script is the short version of "Myo library" and provides the connection of Myo raw data with other scripts. It decrees the calls through a DLL (Dynamic Link Library) and evaluates every result as successful or erroneous. Every function call that goes to Myo or every raw data that is to be interpreted is parsed for the receiving end.

EventTypes.cs

This script parses every event Myo recognizes and converts it to relevant data, such as a variable of Myo. It consists of multiple classes that inherit from MyoEventArgs, which in turn inherits from the built-in C# class EventArgs. One example would be how a change in pose is parsed into a variable in the Thalmic.Myo.Pose class:

```
1 public class PoseEventArgs : MyoEventArgs
2     {
3         public PoseEventArgs(Myo myo, DateTime timestamp, Pose pose)
4             : base(myo, timestamp)
5         {
6             this.Pose = pose;
7         }
8         public Pose Pose { get; private set; }
9     }
```

Hub.cs

This script is a helper script for ThalmicHub.cs, as it handles auxiliary events of Myo (being connected/disconnected/resetting the hub) through the libmyo library. It also holds the enumeration(enum)[1] for the locking type of Myo. The constants for

[1] Enumerations in Unity allow you to create a collection of constants and use them like static variables.

the enumeration "LockingPolicy" are named "None" and "Standard", with the former disabling locking and the latter denying pose events from being sent while a Myo is locked.

Myo.cs

This script acts as an interface between the manually written scripts and the libmyo.cs and EventTypes.cs. It basically calls respective functions from the library and the EventHandler system to send requests back to Myo.

Moreover, it holds various enumerations[1] as explained below. Note that enumerations of type unknown is caused by loss of data, like Myo being disconnected or outputting data that does not fit into any definition.

- **Arm:** Outputs which arm is used.Note that the right arm is used in this project.
  - Left
  - Right
  - Unknown
- **xDirection:** The direction the user's arm is facing. Note that the standard type of "TowardWrist" is used in this project.
  - TowardWrist
  - TowardElbow
  - Unknown
- **Vibration Type:** The length of the vibration.
  - Short
  - Medium
  - Long
- **UnlockType:** How Myo should be unlocked.
  - Timed: Unlocked for a fixed period of time(standard: 2 seconds).
  - Hold: Unlocked until explicitly told to re-lock.
- **Pose:**The pose that is detected by Myo. Any disturbance in the signal yields the result "Unknown". No particular pose recognized by Myo results in the output "Rest". This enumeration draws its data from libmyo.PoseType, where all poses are assigned an integer except "Unknown", which has the hexadecimal value of 0xffff, which in return corresponds to -1(error result).

- Rest
- Fist
- WaveIn
- WaveOut
- FingersSpread
- DoubleTap
- Unknown

4.3 Class Hierarchy

Excluding the Unity API for Myo, there are six different scripts used in the project.Two of these scripts explained below affect the others and their functions are used by all the features. The last version of the code can have slight differences in terms of adaptation for test environments and refactoring.

4.3.1 myo_base.cs

"myo_base.cs" handles the mathematical calculations necessary for the similarity of the orientation of Myo and the user's arm, along with an unlocking function.All classes that implement features draw the mathematical baseline from this class.

It should be noted that the following functions ComputeOrientation(), rollFromZero(), computeZeroRollVector(), normalizeAngle(), and ExtendUnlockandNotifyUserAction() are included as examples with the Unity3D Myo API, and are published as open-sourced code by Thalmic Labs. Minor changes are made in order to fit the concept and preserve the integrity of the project.

```
1  public static Quaternion computeOrientation()
2      {
3          bool updateReference = false;
4          if (thalmicMyo.pose != _lastPose)
5          {
6              _lastPose = thalmicMyo.pose;
7              if (thalmicMyo.pose == Pose.FingersSpread)
8              {
9                  updateReference = true;
10                 ExtendUnlockAndNotifyUserAction(thalmicMyo);
11             }
12         }
13         if (updateReference)
```

```
14          {
15              _antiYaw = Quaternion.FromToRotation(
16                  new Vector3(myo.transform.forward.x, 0, myo.transform.forward.z),
17                  new Vector3(0, 0, 1)
18              );
19              Vector3 referenceZeroRoll =
                    computeZeroRollVector(myo.transform.forward);
20              _referenceRoll = rollFromZero(referenceZeroRoll,
                    myo.transform.forward, myo.transform.up);
21          }
22
23          Vector3 zeroRoll = computeZeroRollVector(myo.transform.forward);
24          float roll = rollFromZero(zeroRoll, myo.transform.forward,
                myo.transform.up);
25          float relativeRoll = normalizeAngle(roll - _referenceRoll);
26          Quaternion antiRoll = Quaternion.AngleAxis(relativeRoll,
                myo.transform.forward);
27          Quaternion result = _antiYaw * antiRoll *
                Quaternion.LookRotation(myo.transform.forward);
28
29          return result;
30      }
```

Listing 4.1: parts of computeOrientation() function

As it was stated earlier, the orientation of Myo is computed in every frame and applied on the GameObject "joint", which was explained in subsection 4.2.4.This is done in the computeOrientation() function (see listing 4.1.To align the orientations of the arm and the Myo in the build, the user spreads his fingers. Thus the reference orientation will be updated, which corresponds to the null angle in synchronization with the user's arm pointing forward. This is realized with the bool variable "updateReference". If it is activated, a series of computations result in the same orientation that is comprehensible by the user (see line 13 in listing 4.1).

First, the _antiYaw variable is instantiated with the Quaternion.FromToRotation() function, which is an in-built function in Unity3D and rotates the Myo such that the z-axis is 1 when the user is pointing in the forward direction. After that the _referenceRoll vector, the reference of how many degrees the Myo is rotated about its forward axis from the reference zero roll direction, is computed with the help of two functions: computeZeroRollVector() and rollFromZero(). When the reference is taken, the joint GameObject will be rotated about its forward axis such that it faces upwards when the roll value matches the reference.

The first function computes a "zero roll" vector that points perpendicular to the

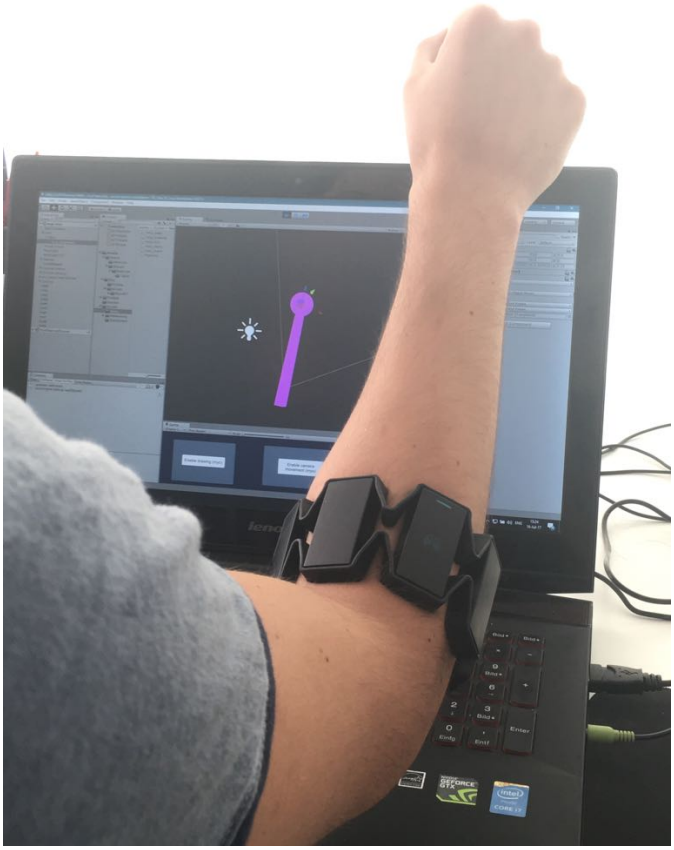

Figure 4.5: The arm and the joint GameObject are in sync after the updated reference.

forward direction while minimizing angular distance from the positive y axis. The result represents the direction of no rotation about its forward (z) axis.

The second function computes the angle of rotation clockwise about the forward axis relative to the provided zero roll direction. As the armband rotates about the forward axis the values will change and fit between -180 (counter-clockwise) and 180 (clockwise) depending on the direction of rotation (see listing 4.2 for more details). Regardless of the fact that the updateReference part is handled or not, the orientation for Myo is computed in every frame through the _referenceRoll and _antiYaw variables set in "updateReference" part. To compute the roll value, the same functions in listing 4.2 are used. Then the difference of this value and the zero roll value is stored in the float relativeRoll. The rotation resulting from this roll difference is computed and assigned in the quaternion value antiRoll. At last, the resulting orientation is achieved through

the multiplication of _antiYaw, antiRoll and the original rotation of the Myo.

It should be noted that all above calculations are done assuming the Myo armband's +x direction(axis) (in it's own coordinate system) is facing towards the wearer's elbow.

```
1
2       static Vector3 computeZeroRollVector(Vector3 forward)
3       {
4           Vector3 antigravity = Vector3.up;
5           Vector3 m = Vector3.Cross(myo.transform.forward, antigravity);
6           Vector3 roll = Vector3.Cross(m, myo.transform.forward);
7           if (roll != _lastRoll)
8               _lastRoll = roll;
9           return roll.normalized;
10      }
11
12      static float rollFromZero(Vector3 zeroRoll, Vector3 forward, Vector3 up)
13      {
14          float cosine = Vector3.Dot(up, zeroRoll);
15          Vector3 cp = Vector3.Cross(up, zeroRoll);
16          float directionCosine = Vector3.Dot(forward, cp);
17          float sign = directionCosine < 0.0f ? 1.0f : -1.0f;
18          return sign * Mathf.Rad2Deg * Mathf.Acos(cosine);
19      }
```

Listing 4.2: rollFromZero() and computeZeroRollVector() functions

After this computation, the orientation of the user's arm and the "joint" GameObject (explained in subsection 4.2.4) will be matched(see the figure 4.5).

The class also offers direct arithmetical functions to convert any given quaternion to it's respective yaw, pitch and roll angles. See the listing 4.3 for the code and the subsection 4.2.2 for the derivation.

```
1  public static float computeYaw(Quaternion quat)
2  {
3      return (float)Math.Atan2(2.0f * (quat.w * quat.z + quat.x * quat.y),
4                               1.0f - 2.0f * (quat.y * quat.y + quat.z * quat.z));
5  }
6  public static float computeRoll(Quaternion quat)
7  {
8      return (float)Math.Atan2(2.0f * (quat.w * quat.x + quat.y * quat.z),
9                               1.0f - 2.0f * (quat.x * quat.x + quat.y * quat.y));
10 }
11 public static float computePitch(Quaternion quat)
12 {
13     return (float)Math.Asin(2.0f * (quat.w * quat.y - quat.z * quat.x));
```

```
14 }
```

Listing 4.3: The yaw, pitch and roll calculations

As the last function of this class, ExtendUnlockAndNotifyUserAction() (see listing 4.4) extends the unlocking of Myo for additional gestures (if the locking policy is standard) for two seconds and notifies the given Myo that a user action was recognized. This comes in handy especially to prevent flooding of functions that are triggered when a specific gesture is made or to make sure Myo does not lock itself while a gesture is still measured.

```
1
2     public static void ExtendUnlockAndNotifyUserAction(ThalmicMyo myo)
3     {
4         ThalmicHub hub = ThalmicHub.instance;
5
6         if (hub.lockingPolicy == LockingPolicy.Standard)
7         {
8             myo.Unlock(UnlockType.Timed);
9         }
10        myo.NotifyUserAction();
11    }
```

Listing 4.4: ExtendUnlockAndNotifyUserAction() function.

4.3.2 myo_GUI.cs

The script "myo_GUI.cs" is responsible for giving out the instructions for the user and some additional information for debugging. Again, the output given differs depending on which feature is active at that time. The information is displayed in a box on the upper left part of the screen.

In the Update() function, the choice of resetting the hub is given (when the user presses "w"). If the scene is initialized, the respective UI for every feature will be set active.When any of the features is selected, the UI will be deactivated. If the user exits a feature mode, the UI will be activated again. All of this code functions with a boolean variable "enableMainMenu" (see listing 4.5).

The buttons on the UI panels take care of switching between features, and they all have functions assigned to them. Each activate a boolean that readies the scene for the upcoming feature. More details will follow in each feature chapter. See the listing 4.6 for individual boolean switches regarding each feature.

Lastly, the OnGUI() function gets updated in every frame and is used to draw anything on screen, an informative rectangle in our case. If everything is working as

intended for Myo, the instructions for each feature will be displayed with the help of this function. Other errors like Myo not being paired or not synced with the user's arm can also be reflected here.

```
void Update () {
        if(Input.GetKeyDown("w"))
            {
            hub.ResetHub();
            }
        if (thalmicMyo.pose != _lastPose)
        {
            _lastPose = thalmicMyo.pose;
        }
        //open // close menu functionality
        if (enableMainMenu)
        {
            ui_draw.SetActive(true);
            ui_menu.SetActive(true);
            ui_move.SetActive(true);
        }
        else
        {
            ui_draw.SetActive(false);
            ui_menu.SetActive(false);
            ui_move.SetActive(false);
        }
    }
```

Listing 4.5: The Update() function of myo_GUI.cs

```
public void enableDraw()
    {
        drawing = true;
        enableMainMenu = false;
    }
public void enableMoving()
    {
        moving = true;
        enableMainMenu = false;
    }
public void enableMenu()
    {
        menu = true;
        enableMainMenu = false;
```

```
15      }
```

Listing 4.6: The functionality for UI Buttons

4.4 Features

There are three features implemented, not just for the sake of the soccer tactics simulation, but for the overall use and adaptability in any similar type of simulation or game, matching the expectations for an interface for a gestural control device. The features are designed and selected in this regard.

4.4.1 Drawing with Myo

Probably the most useful and impactful feature implemented for Myo, the first feature lets the user draw lines on a 2D screen. The user can choose to erase the last line or clear the complete screen as well, with the help of some gestures. It's corresponding script is named "myo_drawing.cs". A class for the lines that are drawn is also present for easier usability, namely "myoLine.cs". The instructions for drawing are:

- Move your arm to control the cursor on the screen.
- Making a fist with and holding it draws a line until the user changes the pose of the hand.
- Waving in deletes all the lines.
- Waving out deletes just the last line.

In order to control a cursor with Myo, the orientation of the GameObject "joint" is mapped to a 2D plane, using the x and y value differences. The speed and the sensitivity of the cursor can be changed through the Unity editor, or directly through the variables within the script.

The lines are drawn with the help of the Unity component "LineRenderer". Every line has a GameObject assigned to it, with the component LineRenderer ready to use in every one of them. When a line is deleted through a pose (waving in or waving out), the GameObject remains active and it's LineRenderer component just deletes the previously mapped points, it could be said that it resets itself.

At the initialization of the feature, 10 GameObjects for drawing lines are also initialized. If more GameObjects are needed, i.e. the user wants to draw more lines, it will be dynamically recognized within the script and more GameObjects with this purpose will be instantiated. These additional GameObjects will be destroyed when the feature terminates to prevent any unnecessary burden on the Unity Engine.

myoLine.cs

In order to draw lines effectively and allocate resources better, a class needed to be created to hold the attributes of drawn lines. The class "myoLine" fulfills that purpose, as its instances are used as objects that control individual lines in the editor.

The attributes and the constructor of the class can be viewed in the listing 4.7. Each instance of this class thus has a name, an assigned GameObject in the Unity Editor, a LineRenderer, starting and ending points and a list of points that integrate the drawn line. In the constructor, the standards of the assigned LineRenderer are set, like the color of the line, line width, selected shader etc. The LineRenderer uses the world space to draw because the following calculations are done with this in mind. Moreover, a GameObject with the same name is instantiated and a LineRenderer component is attached to it.

```
1  public class myoLine : MonoBehaviour {
2      //a class for every line drawn
3      private LineRenderer line;
4      private Vector3 startPoint;
5      private Vector3 endPoint;
6      private List<Vector3> pointsList;
7      private GameObject lineObject;
8      private string namee;
9
10     public myoLine(string myName)
11     {
12         this.myName = myName;
13         pointsList = new List<Vector3>();
14         lineObject = new GameObject(myName);
15         Line = lineObject.AddComponent<LineRenderer>();
16         Line.material = new Material(Shader.Find("Particles/Additive"));
17         Line.numPositions = 0;
18         Line.startWidth = 0.1f;
19         Line.endWidth = 0.1f;
20         Line.startColor = Color.red;
21         Line.endColor = Color.green;
22         Line.useWorldSpace =true;
23     }
24 }
```

Listing 4.7: The variables and the constructor of class myoLine.cs

myo_drawing.cs

This script is the main script for handling the drawing feature. On a side note, since this was the first feature, some applications affecting all of the features are implemented in this script, like disabling the renderer of the GameObject joint (for a better visual experience) and matching the orientation of the joint with the arm wearing Myo. Since all of the feature scripts are attached to the main camera in the Unity editor and the camera is always active, the Update() functions always run in every feature script, although most of the code is not handled due to the boolean values mentioned in the section 4.3.2.

As said before, instances of myoLine are instantiated at the start throughout the execution (when needed) with the function createLine() (see listing 4.8), which crates a myoLine object and adds it to the list of current myoLine objects, so they can be accessed easily afterwards.

```
1 myoLine createLine(string name)
2 {
3     myoLine myoLine = new myoLine(name);
4     linesList.Add(myoLine);
5     return myoLine;
6 }
```

Listing 4.8: Creating a line within myo_drawing.cs

For using the virtual cursor, two methods are implemented. The user can either control the cursor with the mouse input or with the input from Myo. The reason for implementing a mouse input is to provide a debugging environment and compare the results between the methods .Normally the user can access the mouse control by pressing the key "f" while Unity Editor is running. The mapping process (acquiring the 3D input in the editor and transforming it to a 2D input so the cursor just moves on the x-y plane, independent from the z-axis) has many similarities. In sum, the x and y values from the mouse axis or the difference in pitch and yaw angles of Myo is added to the values of the virtual cursor. The z value is set to null just in case. Subsequently, the position of the cursor is clamped to the screen height and width so that the cursor is always visible. At last the position of the cursor is transferred from screen view into the world view.

After the calculation of the cursor position, the code checks for potential gestures.If the user is making a fist and holding it while moving the cursor with its arm, all the points on the screen the cursor passes through will be connected together to form a line. If myoLine GameObjects are running low, additional ones will be created. If the position of the last point and the current point is the same, it will not be added to

the line.The code snippet is shown below as "currentLine" is an instance of the class myoLine:

```
1 if (drawing)
2         {
3         if (!currentLine.getPointsList().Contains(posInScreen))
4                 {
5                 currentLine.getPointsList().Add(posInScreen);
6                 currentLine.Line.numPositions =
                      currentLine.getPointsList().Count;
7                 currentLine.Line.SetPosition(currentLine.getPointsList().Count -
                      1,
                      (Vector3)currentLine.getPointsList()[currentLine.getPointsList().Count
                      - 1]);
8                 }
9         }
```

Listing 4.9: Adding the points together to make a line.

If the user waves in, all of the lines will be removed, as all of the myoLine objects will be iterated (as they are stored in a list when instantiated) and the list of points that define the line is emptied. The number of line segments is also set to zero.

When the user waves out, the last element of the list is accessed and all of its saved points and line segments are cleared.

By double tapping, the board will be cleared again, just like waving out, and the user will return to the feature selecting menu.

4.4.2 Camera Movement with Myo

The second feature is the ability to move the camera with the use of Myo. For this, the myo_move.cs script is implemented. The controls for the camera movement are as follows:

- Fist: camera moves forward
- Spread fingers: camera moves backwards
- Wave in: camera moves to left (if using Myo on a right arm)
- Wave out: camera moves to right (if using Myo on a right arm)
- Move hand leftward: camera rotates leftward
- Move hand rightward: camera rotates rightward

While the first four commands are trivial and are activated when the pose is recognized, rotating the camera along the Y-axis needs the calculation of the yaw angle and the difference of the current yaw angle and the reference yaw angle, which corresponds to zero depending on the orientation on Myo.

The code is run only through a boolean check, the boolean value "moving" from the script "myo_GUI.cs", which is activated when the button for the camera movement feature in the UI is pressed. More information on the subsection 4.3.2.

If the code is being run for the first time -that is, if the feature is selected- the GameObjects in the scene will be set active, which are all stored in a list of type <GameObject>. The camera will also be moved to the starting location.

The yaw difference will be calculated in every frame and the camera will be rotated if the difference is greater than 25 degrees from the zero yaw angle, the direction depending on the sign. After that, any poses that are made are checked and the camera is moved accordingly.

Note that the movement speed of the camera depends on a speed factor that can be changed anytime in the Unity editor. To exit the feature, the user has to make a double tap, following the deactivation of GameObjects used in the scene and return to the feature selection menu. Parts of the code can be seen in listing 4.10.

```
1  public List<GameObject> sceneItems = new List<GameObject>();
2  public Camera mainCamera;
3  public float movSpeed;
4  bool madeActive = false;
5  public GameObject joint;
6  void Update () {
7      if (myo_GUI.moving)
8      {
9          //move the camera on the left-right axis depending on yaw angle
10         float yawDiff =
               myo_base.normalizeAngle(joint.transform.rotation.eulerAngles.y);
11         if (yawDiff > 25.0f)
12         {
13             //move to right
14             mainCamera.transform.RotateAround(transform.position,
                   Vector3.up,movSpeed);
15         }
16         else if (yawDiff < -25.0f)
17         {
18             //move to left
19             mainCamera.transform.RotateAround(transform.position, -Vector3.up,
                   movSpeed);
20         }
```

```
21          switch (thalmicMyo.pose)
22              {
23                  case Pose.Fist: //move forward
24                      mainCamera.transform.position +=
                            mainCamera.transform.forward * movSpeed * Time.deltaTime;
25                      break;
26                  case Pose.FingersSpread: //move backward
27                      mainCamera.transform.position -=
                            mainCamera.transform.forward * movSpeed * Time.deltaTime;
28                      break;
29                  }
30  }
```

Listing 4.10: Parts of the Update() function of myo_move.cs that takes care of the camera movement in every frame.

4.4.3 Circular Menu Control

The last implemented feature is the circular menu control with the Myo, with the allocated script "myo_menu.cs". The user switches between the menu items using the roll angle of the Myo. For this feature, two different types of interaction is implemented:

- **Absolute Menu:** The roll value has a fixed angle between -60 and 60. The items are divided between these ranges, depending on the item count.
- **Relative Menu:** The selected item changes to its left/right neighbor if the roll angle difference between the current angle and the reference zero angle is higher than 30. Through this implementation, the user can also jump from the first item to the last item, and vice versa.

To test these implementations and understand in which situations they fare better, two different menus are realized too, one with six items and one with twelve. For the menu with six items, the circle is divided into six equal parts, each having a range of 20 degrees (in the absolute menu, 10 degrees in menu with twelve items). The items are actually UI elements with one single canvas as their parent GameObject and item numbers (which refer to it's name in the code), source images and assigned colors as their children. To easily augment it with the code, a serializable class "MenuSelection" is instantiated within myo_menu.cs (see listing 4.11).

```
1  [System.Serializable]
2  public class MenuSelection
3  {
4      public string name;
```

```
5     public Image sceneImage;
6     public Color normalColor = Color.white;
7     public Color highlightedColor = Color.white;
8     public Color pressedColor = Color.gray;
9 }
```

Listing 4.11: Helper class MenuSelection.

As usual, the script begins to function first when the boolean value "menu" is activated through the GUI script. If the scene is instantiated after the main menu, the GameObjects related to the feature will be activated first. While selection is done through the roll angle, actually selecting an item requires a fist to be made. Doing the double tap gesture returns the user to the main menu again. In every update, the current menu item the user is hovering over is calculated through the function getCurrentMenuItem(). In the snippet below (4.12), a simplified version of the update function can be seen.

```
1  void Update()
2      {
3          if (myo_GUI.menu)
4          {
5              if (firstTimeInside)
6              {
7                  if (!madeActive && using12Menu)
8                  {
9                      canvas12.SetActive(true);
10                     madeActive = true;
11                 }
12                 //init old and new items
13                 currentItem = 6;
14                 oldItem = 6;
15                 coroutineReady = true;
16                 firstTimeInside = false;
17             }
18             getCurrentMenuItem();
19             //select the selected thing when user is making a fist
20             if (thalmicMyo.pose == Pose.Fist)
21             {
22                 ButtonAction();
23             }
24             //exit to main menu with double tap
25             else if (thalmicMyo.pose == Pose.DoubleTap)
26             {
27                 myo_GUI.menu = false;
```

```
28                  myo_GUI.enableMainMenu = true;
29                  canvas6.SetActive(false);
30                  canvas12.SetActive(false);
31                  madeActive = false;
32                  firstTimeInside = true;
33              }
34          }
35      }
```

Listing 4.12: Parts of the Update() function of myo_menu.cs

The process of getting the current item in every frame is handled in another function (see listing 4.13). First, the roll angle of Myo is calculated.If an absolute menu is being used, the angle will be incremented by 60 to avoid any negative angles in the future.After that any angle less than zero and greater than 119 will be clamped. This way the result will definitely be a value between 0 and 119. The current item is then determined as this angle is divided into the total degrees divided by item count. The reason 120 degrees is used and not the whole roll angle (360 degrees) is that it is not comfortable for the Myo user to rotate their arm to achieve any greater angle. Through testing, the conclusion was reached that a rotation of 60 degrees to each side of the arm is more comfortable for the user.

If a relative menu is being used, just the roll angle difference is calculated like it was done with the yaw angle for the movement feature. If the difference is greater than 30 degrees, the current item value will be incremented or decremented, depending on the angle. Since the update function is called in every frame, this process happens very quickly if left unchecked. Because of that, a coroutine[2] is implemented and a delay of 0.6 seconds is called within the code.

At the end, it is checked whether the old item and the current item are different. If that is the case, the highlighted color is removed from the old item and given to the current. The old item is also set to the current item so that this process goes on.

```
1   public void getCurrentMenuItem()
2       {
3           float rollVal =
                myo_base.normalizeAngle(jo.transform.rotation.eulerAngles.z);
4           if (isAbsolute) //absolute menu implementation
5           {
6               int angle = (int)(rollVal + 60);
7               if (angle < 0)
8                   angle = 0;
```

[2] A coroutine is a process in which the IDE stops processing the rest of the code and first goes through the code inserted into the coroutine function.

```
9             else if (angle >= 120)
10                angle = 119;
11            currentItem = angle / (120 / buttons.Count);
12        }
13        else //relative menu implementation
14        {
15            if (coroutineReady)
16            {
17                if (rollVal > 30) // rotate rightwards on menu
18                {
19                    currentItem++;
20                    StartCoroutine("waitSomeTime");
21                }
22                else if (rollVal < -30) // rotate leftwards on menu
23                {
24                    currentItem--;
25                    StartCoroutine("waitSomeTime");
26                }
27            }
28        }
29        if (currentItem != oldItem)
30        {
31buttons[oldItem].sceneImage.color = buttons[oldItem].normalColor;
32oldItem = currentItem;
33buttons[currentItem].sceneImage.color = buttons[currentItem].highlightedColor;
34        }
35    }
```

Listing 4.13: Parts of the function getCurrentMenuItem().

5 User Study

5.1 Purpose

Although the complete implementation functions effectively, it holds no value if it is approved by just one person. Therefore more people (or "testers") were needed to correctly evaluate the project and give appropriate feedback. As a consequence, the flaws of the project can be pinpointed and the project can be improved with the evaluation of results.

5.2 Test Runs

Three tests are carried out, representing one feature each. First, the Myo is worn and synced through Myo Connector. They are given two to three minutes to make themselves comfortable and get used to the device and its gesture recognition. Next, the simulation is run: Users have to perform the "fingers spread" gesture to match the orientation of their arm and Myo.The tests are run in the order below. At the end of each test, the users are asked to evaluate how comfortable the feature was in a scale of 1 to 10 with 1 being the worst note and 10 being the best.The tests are as follows:

- **Drawing Test:** The users try out the first feature: Drawing on a 2D board. The time in which they finish their tasks is recorded.
  - Drawing a circle: Here, the circular motions are tested.
  - Drawing a square: Here, drawing single lines and connecting them is tested.
- **Maze Test:** The users try out the movement feature within Myo. Using the spatial and gestural commands, testers try to reach to the end in a maze. The time in which they finish the task is recorded.
- **Menu Selection Test:** Four different menu selection tests, each with slight differences, are carried out here with the order below. Testers choose a declared number from the menu and the timing is recorded:
  - Absolute menu with 12 elements

- Absolute menu with 6 elements
- Relative menu with 12 elements
- Relative menu with 6 elements

As it was stated in chapter 4, absolute menu divides the menu into fixed angle values, while relative menu changes the selection based on the roll angle difference. Menus with different number of items are put to test in order to check which style fits which number of items better.

The testers were also questioned about their relevance with computer games and any previous experience with virtual or augmented reality. The reason for that was to see if there is a connection between test scores and previous affinity with the test environment. The possible answers were "yes", "no", and "little".

The user study was carried out with 17 testers, all of them aged between 19 and 24. The following table shows the test details and results. Benchmark times are also included for comparison.

	Yes	**No**	**A little**
Previous Gaming Experience	11	3	3
Previous VR/AR Experience	5	8	4

Table 5.1: Distribution of testers over previous gaming and VR/AR experiences.

	Average Completion Time (in seconds)	**Benchmark Time (in seconds)**	**Average Note (from 1 to 10)**
Test 1:			
Drawing a circle	6.235	5	6.059
Drawing a square	21.764	12	4.823
Test 2:			
Movement in maze	109.176	90	7
Test 3:			
Absolute menu with 12 items	3.529	3	8.235
Absolute menu with 6 items	3.117	3	8.117
Relative menu with 12 items	4.235	4	7.294
Relative menu with 6 items	4.235	4	7.529

Table 5.2: Results for each test.

5.3 Results

Although done with a small number of subjects, carried out tests and the feedback of users present some valuable information for the performance of Myo as well as for each individual test.

5.3.1 Myo Evaluation

Notably, most users struggled to perform precise gestures when using the device for any tests, although most of them commented on the intuitiveness of Myo. This can be caused by stress from having to perform under a certain amount of time as well as inexperience using the device. Factors like body fat, temperature and perspiration also play a role in the performance of Myo (as stated in [21] and [32]). It could also be that the classifier algorithms for gestures are not working that precisely (sometimes gestures were still classified with errors even when users were experienced with Myo), which is also stated in [22] and chapter 2. Issues of synchronization and classification errors were also present in participants with thinner arms.

It must be indicated that Myo Connect offers custom profiles for users so that muscle readings are more accurate. In our case, it would be very time consuming to create custom profiles for every user, but doing so would significantly reduce the problems written above. As the end goal of this project, one or two people will be using the interface all the time, so building profiles for these people would be both logical and beneficial for the usability.

5.3.2 Feature Evaluation

The drawing feature was tested first, which could be an error because it seems to be the hardest feature to control according to results. The users were more comfortable drawing circular lines whereas drawing orthogonal lines proved to be much more harder. The cursor movements are drawn from the yaw and pitch angles and are not prone to radical changes in one updated frame, leading to circular movements with the cursor being easier to do.

The second tested feature was the movement in the maze. Users were particularly interested and found the interface fairly intuitive. After testing out the level multiple times, the benchmark time was set to 1 minutes and 30 seconds. Many testers were able to finish the maze much sooner than expected, which also showcased itself in the evaluation of the feature.

The last test was the circular menu. Here, users mostly preferred the absolute menu over to the relative menu. The fact that angle ranges were mapped to each item made

it particularly easy for users to navigate between menu items. On the other side, using the relevant menu enabled the users to make a full circle when navigating, which was a functionality most liked. Overall, users preferred the larger menu when using absolute controls and the smaller menu when using relative controls. The reason for that in an absolute menu could be the smaller angle ranges in a bigger menu, leading to a stronger control between items. Reaching the end nodes (in our case, items 1 and 12) proved to be difficult nevertheless. Since switching between the items in a relative menu calls a coroutine, the speed is hindered. Thus the menu with smaller items seemed better due to the indicator traversing the whole menu faster. Users also mentioned the slowness of the relative menu. This could be fixed by adjusting the time the code has to wait in the coroutine despite potential complications with Unity.

5.3.3 Performance with Regard to Previous Experience

The distribution of testers were also expected: The ones that haven't played games also did not try out any VR/AR devices. People that had gaming experiences gave out equally distributed answers: Out of 11 people, 4 have never used such a device, 4 of them had no knowledge and the remaining 3 had used it once. Testers with experience performed overall better, completing the task in shorter time than others. This was also reflected on the evaluation as the first mentioned group gave overall higher points than the second group. This discovery can lead to the statement that affinity with the interface or used devices is an important factor for the usability of such gadgets.

6 Conclusion

6.1 Discussion and Future Work

The user study shows that the implementation proposed in this work is functioning, but it is far from perfect. Future improvements could be categorized as follows:

6.1.1 Myo Improvements

Myo uses some state of the art sensors to match the hand movements into gesture classes, but sometimes misclassifications occur more than expected. The reasons for that are already discussed in detail in chapters 2 and 5 and hopefully EMG technology will find solutions in the future for the accuracy problem. Improvement on sensing and machine learning algorithms can help as well.

Another issue is the scarce number of gestures, as it is stated in other papers and testers. Thalmic Labs is already working on adding more gestures to Myo armband, so the diversity brought with new tactile ways of expression will definitely improve any interfaces built in the future.

A further point that can be investigated is using two Myo's at the same time to gain access to more control in necessary simulations.

6.1.2 Feature Improvements

The most successful feature was the circular menu implementation and it seems it can't be developed much further other than minor fixes. The movement script also seems to be working well excluding gesture misclassifications, which lies on the Unity API.

The feature that needs improvement the most is probably the drawing implementation. This also shows itself in the user study, where the overall note for the feature was relatively lower than the other two. During the implementation the orientation of Myo was mapped directly to a cursor first, which failed as a recalibration was necessary to match the Myo and the user's arm. After creating a "joint" GameObject for that purpose, transformation differences of the tip of the joint were used and it's x and y values were added to the cursor. Finally the pitch and yaw angles were opted for use

for cursor control, but the movement of the cursor seems a bit lagging and hard to use when drawing concrete edges.

Since the feature implementations are mostly a standalone project, they can be put into various VR and AR projects, starting with the soccer tactics simulation.

6.2 Final Word

Myo is a great device that provides the user a very different form of interaction. The community support and upcoming improvements shows that it has a bright future. Although used in many aspects like device control and some mini-games, an interface implementation for Myo in such a scale was not done before. I was more than happy to try out the device and tweak its functionality into a user interface that is hopefully easily applicable to other upcoming projects.

Since this is a Bachelor's Thesis, I did not have sufficient time to merge the interface with the soccer tactics simulation as I intended at fist. Still I think the core of the project was to find ways of interaction and measure their usability, as the merging optionality can be carried out afterwards as the main part is done. At the end, all fundamental aspects are successfully implemented and tested for their usability and comfort.

List of Figures

List of Tables

Bibliography

[1] M. Abduo and M. Galster. "Myo Gesture Control Armband for Medical Applications." In: (2015).

[2] *Aircraft Principal Axes*. (last visited on July 2017). 2017. URL: en.wikipedia.org/wiki/Aircraft_principal_axes.

[3] A. Alkan and M. Günay. "Identification of EMG signals using discriminant analysis and SVM classifier." In: *Expert Systems with Applications* 39.1 (2012), pp. 44–47.

[4] G. C. Burdea. "Keynote address: haptics feedback for virtual reality." In: *Proceedings of international workshop on virtual prototyping. Laval, France*. 1999, pp. 87–96.

[5] A. T. Cabreira and F. Hwang. "An analysis of mid-air gestures used across three platforms." In: *Proceedings of the 2015 British HCI Conference*. ACM. 2015, pp. 257–258.

[6] F. H. Chan, Y.-S. Yang, F. Lam, Y.-T. Zhang, and P. A. Parker. "Fuzzy EMG classification for prosthesis control." In: *IEEE transactions on rehabilitation engineering* 8.3 (2000), pp. 305–311.

[7] *Euler Angles*. (last visited on July 2017). 2017. URL: en.wikipedia.org/wiki/Euler_angles.

[8] L. Euler. "Novi Commentarii academiae scientiarum: Petropolitanae." In: (1776).

[9] C. Goodine. *Bootstrap Week: Myo Development 101*. 2015. URL: https://www.youtube.com/watch?v=sJv2AMLz3CA.

[10] A. S. Hardy. *Quaternions*. "Ginn, Health & Company", "1881".

[11] hatIs.com. *Gyroscope Definition*. 2016. URL: http://whatis.techtarget.com/definition/gyroscope.html.

[12] A. Hiraiwa, K. Shimohara, and Y. Tokunaga. "EMG pattern analysis and classification by neural network." In: *Systems, Man and Cybernetics, 1989. Conference Proceedings., IEEE International Conference on*. IEEE. 1989, pp. 1113–1115.

[13] H.-P. Huang, Y.-H. Liu, L.-W. Liu, and C.-S. Wong. "EMG classification for prehensile postures using cascaded architecture of neural networks with self-organizing maps." In: *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*. Vol. 1. IEEE. 2003, pp. 1497–1502.

[14] H. Huang, P. Zhou, G. Li, and T. Kuiken. "Spatial filtering improves EMG classification accuracy following targeted muscle reinnervation." In: *Annals of biomedical engineering* 37.9 (2009), pp. 1849–1857.

[15] T. Labs. *Myo Market*. 2017. URL: https://market.myo.com/.

[16] T. Labs. *Thalmic Labs*. (last visited on July 2017). 2016. URL: https://www.thalmic.com/.

[17] T. Labs. *Thalmic Labs Information Page, Part:"The Myo Armband"*. 2017. URL: https://www.thalmic.com/.

[18] M.-F. Lucas, A. Gaufriau, S. Pascual, C. Doncarli, and D. Farina. "Multi-channel surface EMG classification using support vector machines and signal-based wavelet optimization." In: *Biomedical Signal Processing and Control* 3.2 (2008), pp. 169–174.

[19] G. D. Morais, L. C. Neves, A. A. Masiero, and M. C. F. de Castro. "Application of Myo Armband System to Control a Robot Interface." In: *BIOSIGNALS*. 2016, pp. 227–231.

[20] A. Nair, M. Jagtap, S. Patil, and S. Teli. "DESIGN OF A GESTURE CONTROLLED ROVER WITH 5 DOF MANIPULATOR FOR EXPLORATION AND RESCUE APPLICATIONS." In: ().

[21] C. Nordander, J. Willner, G.-Å. Hansson, B. Larsson, J. Unge, L. Granquist, and S. Skerfving. "Influence of the subcutaneous fat layer, as measured by ultrasound, skinfold calipers and BMI, on the EMG amplitude." In: *European journal of applied physiology* 89.6 (2003), pp. 514–519.

[22] K. Nymoen, M. R. Haugen, and A. R. Jensenius. "Mumyo–evaluating and exploring the myo armband for musical interaction." In: (2015).

[23] A. M. Okamura. "Methods for haptic feedback in teleoperated robot-assisted surgery." In: *Industrial Robot: An International Journal* 31.6 (2004), pp. 499–508.

[24] P. for People. *MAX MSP: GETTING STARTED WITH GESTURE - MYO ARMBAND*. 2016. URL: https://www.youtube.com/watch?v=FeZ5fC5JiJ8&t=5s.

[25] I. Phelan, M. Arden, C. Garcia, and C. Roast. "Exploring virtual reality and prosthetic training." In: *Virtual Reality (VR), 2015 IEEE*. IEEE. 2015, pp. 353–354.

[26] N. M. Planning. "Analysis Division. Euler Angles, Quaternions, and Transformation Matrices. Document Linkhttp." In: *ntrs. nasa. gov/archive/nasa/casi. ntrs. nasa. gov/19770024290. pdf. Last Accessed–March* (2015).

[27] A. D. Program. *"Get Started with Gesture Control Using the Myo Armband"*. 2015. URL: https://www.youtube.com/watch?v=w6cKglRcpmo.

[28] M. Sathiyanarayanan and T. Mulling. "Map navigation using hand gesture recognition: A case study using myo connector on apple maps." In: *Procedia Computer Science* 58 (2015), pp. 50–57.

[29] H.-S. Shin, A. Ganiev, and K.-H. Lee. "Design of a Virtual Robotic Arm based on the EMG variation." In: *Proc. ASTL* (2015).

[30] E. C. Silva, E. W. Clua, and A. A. Montenegro. "Sensor data fusion for full arm tracking using Myo Armband and leap motion." In: *Computer Games and Digital Entertainment (SBGames), 2015 14th Brazilian Symposium on*. IEEE. 2015, pp. 128–134.

[31] WhatIs.com. *Accelerometer Definition*. 2014. URL: http://whatis.techtarget.com/definition/accelerometer.html.

[32] J. Winkel and K. Jørgensen. "Significance of skin temperature changes in surface electromyography." In: *European journal of applied physiology and occupational physiology* 63.5 (1991), pp. 345–348.

[33] www.myo.com. *Myo Homepage*. (visited on June 2017). 2016. URL: http://www.myo.com.