universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Interaction possibilities in the VR - Framework to handle different devices for input and gesture detection“

verfasst von / submitted by

Michael Pichler BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Diplom-Ingenieur (Dipl.-Ing.)

Wien, 2018 / Vienna, 2018

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 935
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Medieninformatik
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Ing. Dr. Wolfgang Klas

Declaration of Authorship

I, Michael Pichler, BSc, declare that this thesis titled, "Interaction possibilities in the VR - Framework to handle different devices for input and gesture detection" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

Acknowledgements

I would like to thank the University of Vienna for the education, Univ.-Prof. Dipl.-Ing Dr. Wolfgang Klas and Phillip Carlos Chlap BA MSc. for their support and guidance.

Also my parents who have always supported me energetically and all of my colleagues and friends who helped me during the master thesis.

Abstract

Virtual Reality (VR) is a research field which becomes more and more important. In the last two years, immersive reality has reached the next level to leave the laboratory and is now available for normal user. After releasing some headsets for the VR, the expectation was high, but currently it seems that the big hype is over. One reason is, that solutions like Oculus Rift or HTC Vive are not cheap (around 500$ and more) and a PC with high performance is needed. In opposite of expensive solutions, Google Cardboard is a cheap alternative. For this headset, the user only needs a Smartphone and a "Cardboard Headset", developed by Google. The target group are beginners, which are looking for to collect first experience in the VR, and those who do not want spend too much money for the equipment. From the idea to make it accessible for everyone, this work concentrates on how interaction in the VR-world can be done with devices, which users maybe have at home. Professional solutions often have own developed controller for this purpose to make the feeling of interaction more realistic, but for Google Cardboard, there do not exist gadgets like these. Currently it is possible to use a normal controller (the same as it is used for a PC or a gaming console). However, the user is limited to create a natural interaction (no motion/gesture). The output of this research is a framework to handle different devices, which are suitable for interaction with a standard controller. As well, some alternatives are better than other. So the second goal is to investigate advantages and disadvantages of each device and compare them. The focus is on the Microsoft Kinect and a Android-Smartphones.

Zusammenfassung

Virtual Reality ist ein Forschungsfeld, welches immer mehr an Bedeutung bekommt. In den letzten zwei Jahren, VR ist nicht nur mehr in diversen Labors zu finden, sondern es gibt bereits viele Technologien, welche der User auch ohne Problem zuhause verwenden kann. Das Problem zurzeit ist aber, dass viele Headsets, wie zum Beispiel Oculus Rift oder HTC Vive, nicht gerade billig sind (um die 500$). Auerdem wird zumeist ein Hochleistungs-PC benötigt. Eine Alternative zu diesen Headsets stellt Google Cardboard dar, wo lediglich ein Smartphone und die Cardboard Brille benötigt wird. Die Zielgruppe für diese Umsetzung sind Beginner bzw. Einsteiger, die ersten Versuche in die VR starten möchten. Das Hauptaugenmerk sind die Kosten, die hier nur gering ausfallen. Damit der User auch mit der VR Welt interagieren kann, werden verschiedene Eingabegeräte benötigt. Diese Arbeit konzentriert sich darauf, geeignete Alternativ-Geräte zu finden und zu untersuchen, welche Geräte sich dafür am besten eignen. Für den PC z.B. kann man einen normalen Controller verwenden, aber diese Interaktion mit der VR wirken keinesfalls natürlich (keine Bewegung/Gesten möglich). Das Ergebnis der Arbeit stellt ein Framework dar, mit dem es möglich sein soll, verschiedene Geräte einzubinden, mit denen Interaktionen in der VR umgesetzt werden können. Manche eignen sich dafür besser als andere. Somit ist das zweite Ziel der Thesis eine Gegenüberstellung von verschiedene Geräten durchzuführen, um Vor- und Nachteile für diverse Interaktionen in der VR aufzuzeigen. In dieser Arbeit werden hauptsächlich zwei verschiedene Typen von Geräten untersucht, nämlich die Microsoft Kinect und ein Android Smartphone.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

VR is a research field with much potential. It's not restricted only for games how many people think, the unreal world becomes more and more important for simulations and training applications. With it, the developer can create a new view of something that was unreachable before. It is possible to visualize much more details and the customer gets a better premonition how it would look like, even better than it's a simple drawing on paper. The restriction for experts gets lost and normal users have the chance to get a first look on buildings, parks and new technologies as well. The communication is much easier with VR for that as a complex drawing.

As mentioned before, VR can be used to train persons for certain tasks. This task can be, e.g., to show how a work flow must be done or in the medical area to educate surgeons. In the real world, every mistake can be fatal for the patient, in the VR it doesn't matter. In the unreal world, the user can learn from mistakes and how he can do it better next time. A really big area of VR is the gaming industry which is very widespread. Since last year, there was a high increase of new technologies on the marked to create a new feeling to make it more real. The meaning "real" in this context is not only to make photo realism video games that the user cannot differ from real world with fake world, although this is also a goal. No, in this context it means that the user should be able to move free in the immersive world and have the possibility to do the same interactions with objects, persons and movements like in the real world [1].

Every application for VR needs a way which allows the user to manipulate and interact with the environment. For medical purpose, there are sensors which emulate different operation tools to simulate the steps during a chirurgic intervention. In games, the common gadget to manipulate the VR world is the usage of a controller which is currently the preferred device by players. Another point which must be considered is the fact which can't be ignored is the underlying technology and platform for creating the VR. The aim of this master thesis is to create a framework which can handle different input devices. The focus is on interaction and easy adaption of functionalities for the developer needs. This mainly affects the part of gesture recognition and data transport. So it should be possible, that the user can only concentrate on the main part of his project and does not worry about the underlying parts which are always the same. One advantage of this framework is that it's not restricted for one VR technology, but for this project, the framework was tested with the "Google Cardboard" headset.

A small part of this project, which is necessary, is the data transfer between the headset and the input device. The communication have some requirements which must be considered:

- **Reliability:** One of the condition is that the information riches the aim, where it's needed. Sometimes it's allowed that the data gets lost, if this doesn't happen often. A mechanism will handle different kind of data (continuous and discrete values). For both, other requirements are needed. Used technologies for this application will be IP over network and Bluetooth.
- **Immediately response:** To make an interaction possible, the data about the gesture or other information must be transferred in a speed that the user can't determine any or not high delay between carrying out and the noticeable result in the VR. If it isn't possible, the feeling to control things in the VR gets lost.

- **Expandability:** If other features or data are required which the framework doesn't support from the beginning, the developer should have an interface, where he can add the necessary information and define, how it must be serialized and de-serialized for the transfer.

The possibility to extend the framework is an important point, because it will be necessary to define interfaces, where the user can set-up other devices, gestures, or other information. The flexibility ensures that it can be used in many application scenarios and minimize the developer resources like time and planning for the parts which always occur in a project. The user of this framework can test new devices and prove them if it is suitable for VR interaction or not. Figure 1.1 represents an overview of functionalities the framework developed in the work of this theses includes.

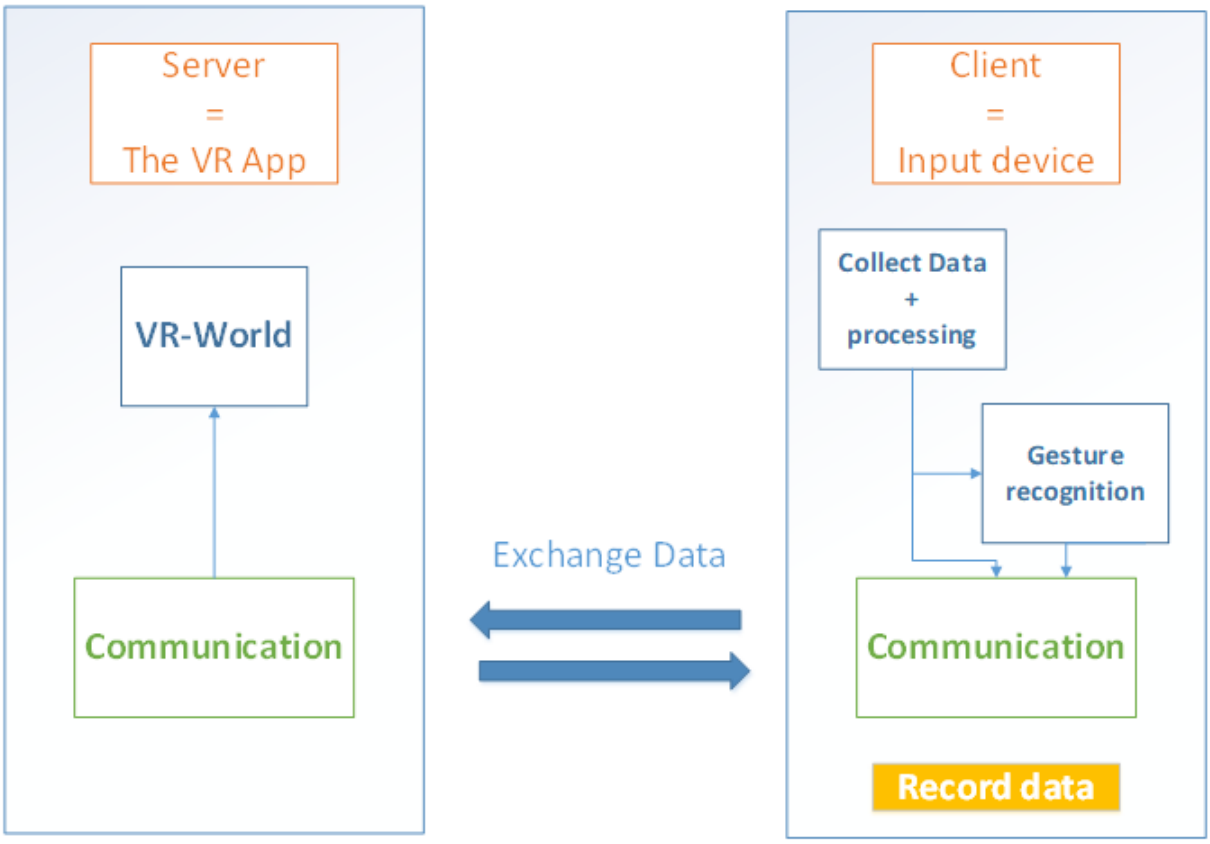


Figure 1.1: Overview of the framework

One of the important parts of the extendibility concept is, how the developer can define own gestures and change the settings. This is important to set-up the optimal detection for certain gestures. Settings can be which algorithm should be used and specify threshold. The gesture detection system also includes a record part, where the user of the framework can record data from a Smartphone for further processing. This part will collect diverse sensor data and save it in a simple format. With extern tools like Octave or Matlab, the developer can select a certain range of the record and save them as a template. The framework can read this template into the memory, and at the application start, the necessary features can be pre-calculate based on the template. More information about this part can be found in chapter 5 and 6.

1.2 VR-Technologies

Meanwhile there exist many VR headsets for creating an immersive world. In this section the work briefly presented the technologies which are currently available and how the problem of interaction for each technology is solved. Each VR company has another approach and area, where it can be used. They all address different target groups. Some

are concentrating only for PCs in the high end sector, other companies are building headsets for Smartphones to show simple environments or for playing 360 videos. Also the area of gaming with regard to consoles becomes more and more important.

1.2.1 Oculus Rift

Oculus Rift is one of the first technologies which have been released and is known by many people. The company behind is Facebook. Oculus Rift is for user who wants to run more complex applications with breath-taking graphics. For that, a PC is necessary with a high end graphic card to calculate the images for both eyes in real time [2]. It supports different platforms for developer to create application for them. For example there exist a SDK for the Unreal Engine as well as for Unity. The Oculus Company has developed and released currently many extra devices, to make the experience better. Most of them are to interact with the environment. If the user wants to have extras, he can buy earphones, different sensors to locate the user in the room and much other applications where localization of the person is important. [3].

Relating to this thesis, an interesting part is the controller, which is offered by Oculus. It's designed to generate input data like movement and buttons which can be used to manipulate the VR. Beside the digital buttons, an analog stick is available, for example it can be used for moving a player avatar. Another feature is, that the system can track the position of the user hands in the real world. This can be used to simulate hand movements, or gesture detection [4].

1.2.2 HTC's Vive

The HTC's Vive headset is a creation from Valve and HTC. Exactly as by Oculus Rift, HTC's Vive is concentrating on the PC sector. It produces high quality stereoscopy images and also needs much computing power to create them. The usage is mainly for gaming [2].

For interaction, with the headset, there will be deliver two cameras to scan a room where it is and create obstacle if in the real environment is an object. Furthermore, two controllers managed the input and position of the player arms [2]. It has also habitual feedback, trigger buttons and multi-function trackpad. For tracking the users head motion, HTC's Vive has sensors like accelerometer and gyroscope in the headset. If the developer will create an application for that, various plugins for the most common engines are available. For example, Unreal and Unity Engine are fully supported [5].

1.2.3 Samsung Gear VR

In contrast to HTC's Vive and Oculus Rift, Samsung Gear VR is designed for Smartphones, more precisely for Samsung Galaxy and Samsung Note phones. For this headset, there is no need of a high end PC [2]. Meanwhile, there exists a new headset (model 2016) with revised design for better handling. On it, there is located a touchpad, designed for menu input, and a button, to confirm or trigger an event. As well, now the user is able to charge his devices at the same time during he runs a VR app [10].

With the headset, basic interaction is possible, but if someone wanted to have more possibilities, he can use a special controller designed for the VR. Oculus Rift and Samsung Gear VR have the same producer and they share their experience between both technologies. If the user has a wireless controller at home, he can also use it [10].

1.2.4 Google Cardboard

The aim of Google is to make VR for every one touchable, to collect first experience and it should be cheap as well. Like Samsung Gear VR, Google Cardboard applications will run on Smartphones, but not restricted to special devices. A point which must be considered during the developing process is, that the app should be able to run on widespread devices with different hardware specification. The range goes from high-end-model to middle and cheap models. So a simple app for introduction into the VR can be execute on every Smartphone, but if developer wants to create a simple game or simulation, the quality (mainly frame rate) can get miserable for certain devices. Google Cardboard has the advantage that everyone can buy it, because the cost of the material is low. As well, if the user wants to have a unique headset, he can also build his own "Cardboard-headset" [11].

To provide a high quality feeling, Google has developed another headset called DayDream. Smartphones only can use the DayDream headset, if they have some minimum requirements, for example screen size (between 4,7" and 6"),

support for Vulcan, screen resolution (at least 1920x1080), screen technology (AMOLED) and more [12]. Currently only few devices are supported (Google Pixel, Google Pixel XL, ZTE Axon 7, Motorola Z, Mate 9 Pro) [13]. Google Cardboard and DayDream have the same SDK and can be used by Unity or Android Studio.

Regardless to the used headset, input detection for interaction is for both very limited. There exists only one button on the top or on the side and it can detect the rotation of the head (tracking the orientation for looking around). If the user can to more as looking around in the VR, the developer can support some kind of other gadgets like a wireless controller.

1.2.5 PlayStation VR

The PlayStation VR headset is the first one for the PlayStation console. It will work for PS4 as well for PS4 pro and is cheaper like Oculus and HTC's Vive. It can be used for gaming and entertainment applications. Like other technologies, the head tracking system is one of the important parts to create a fantastic immersive world. A camera tracks the motion 1000 times per second and the player has a view field of 360 without any destrictions [9].

As usually for a gaming console, there are lots of possibilities for interaction and the user can buy as well lots of extra gadget for it. Sure, it's also possible to use a simple DualShock Controller to play VR games, but he will miss the feeling to be a part of the world. A special controller is the PlayStation Move Controller, where the user has one stick in every hand. This isn't a new device and has been used before for other games. It detects the movements of his hands and has simple buttons for triggering events. For the PlayStation VR, there would be a new version of this controller released with improved detection and stability. For Shooter games, the player can buy a PS VR Aim Controller to imitate a pistol or other weapons, where the player can aim a target and shoot them down [9].

1.2.6 Conclusion

In this section the work briefly presented the most important properties of some explained headset. As it can be seen, every solution has other goals and target groups which should be reached. With more effort, the user can have an excellent experience in the VR. For this the user needs a PC in the high end sector which is not cheap. The headset also caused costs in his budged. On the other hand with lower effort the user have the possibility to make first steps in the VR, or he can test simple applications are viewing videos. The most interesting part with regard to this research work is which possibilities are available to generate data for input and how it can be used for interaction. Often, some solution already exists, but every producer has solved them differently from each other. Table 1.1 represents the important facts to each headset.

Basically it is possible to choose any headset to use this framework to create any applications. The showcases at the end of this theses use Google Cardboard, because there is no need of a special device to run the app. Also the costs of the headset is low and all resources, information, and SDKs are freely available. It's ideal to get the first touch in the VR.

1.3 Goal of the thesis

The goal can be slitted up into two parts. The first part is to write a framework which can be used the create programs for the VR. But the framework doesn't concentrate how to render the world, but how the logic and implementation effort for interaction strategies can be reduces. In other words the developer only needs to record some data, mostly a certain gesture which is needed in his application, and tells the framework where the data can be found. All calculations will be done during runtime to determine if a gesture was made or not.

Another important point is that the framework should be as much flexible as it's possible. This means it shouldn't be static where the developer can't alter the behaviour. A dynamically behaviour can be reached by creating interfaces where he can inject some own logic. For example to extend the framework with more gestures or other algorithm for gesture detection. Also it should be possible to choose the communication technology what the developer needs. An overview of the frameworks features:

- A menu to select between different communication realization (Bluetooth, IP over network)
- A gesture detection system
- Interfaces to add new gestures

Property/Headset	Oculus Rift	HTC's Vive	Samsung Gear VR	Google Cardboard	PlayStation VR
Target platform	PC	PC	Smartphone	Smartphone	Console
Producer	Oculus/Facebook	Valve, HTC	Samsung/Oculus VR	Google	Sony
Area of usage	Gaming	High end Gaming	Video, Image, simple gaming	Video, Image, simple gaming	Gaming
SDK	Unity, Unreal	Unity, Unreal	Unity, Unreal (same Unity/Unreal for Oculus) [6]	Unity, Android Studio	Unity, Unreal (planning)
Input/Interaction	Oculus Touch, Keyboard, Controller, Oculus Remote	Own HTC Vive Controller, Keyboard, Controller	Has also own controller to make inputs	Only one button on the headset, Wireless-Controller	S4 Controller, PlayStation Move Controller
Setup	PC, Oculus Rift headset, HDMI, USB 3.0 [7]	PC, HTC's Vive headset, HDMI, USB 3.0, jack for headphones [8]	Samsung Galaxy S6 - S8, Note7 -; Headset	Any Smartphone with minor version Android 5.0, Cardboard Headset	PlayStation 4, PlayStation VR Headset
Resolution	Total resolution in the consumer version: 2160x1200 with 60/90FPS [7]	Total resolution in the consumer version: 2160x1200 with 60/90FPS [8]	At least in total: 2560x1440	Depending on the used Smartphone	Total: 1920x1080 up to 120FPS [9]
Extras	Positional tracker	All included	-	-	PlayStation Move Controller, PS VR Aim Controller
Price	$599.00	$799.00	$99.00	Low cost, few dollar (can build yourself or buy one, e.g., $7.99)	$399.00

Table 1.1: Compare different headsets

- Possibility to add other algorithm for gesture detection

The second part is to analyse how good or bad the framework works in practice, and show the user which is possible to develop. For this point, a showcase will be create with the help of framework for all used input devices. With the showcase it can be determined important facts of the framework:

- Is it easy to use
- What kind of technologies, other libraries are needed
- How good does the gestures work for a certain use case
- What is to be considered between different devices (how it works, which data, what can be done with the devices)

Chapter 2

Related Work

Implementing this framework needs lot of different research areas. The range goes from gesture detection to computer graphics. Research fields for creating a VR Project are Realtime3D, Stereoscopy, 3D Tracking, Haptics, 3D Sound and 3d Interaction. Each area has its own question which must be solved, for example to create more realistic graphics (photo realism) and needs to be fast (real-time requirement). In other words, VR isn't only a single field, it's a combination of many [14].

The following sections presents solutions that already exist in this area, the basics of different algorithm and what can be used to create a framework for VR interactions.

2.1 Gesture detection

In the area of gesture detection, many previous works address optimization of the recognition rate and minimization of wrong detection. At first, we must define which states a gesture can have and which kind of errors exists. The following simple table represents the possibilities for one gesture:

Gesture detected/carried out	Gesture detected	No Gesture detected
Gesture made	True	True-negative
No gesture made	False-positive	True

Table 2.1: Gesture detection states - Categorization of the states

If the gesture detection system locates a gesture in the upper left cell, than it has works correct. Also the system works right if no gesture was made and no gesture was detected. Any case where a user has carried out a gesture, but it hasn't been detected by the system correctly, it's called True-negative. On the other side, sometimes the systems recognize a gesture, but the user did not carry out any or another gesture, this miss detection is classified as True-negative [15, pp. 27].

There are many works in this area available which are about how good a certain algorithm works. To determine if the system is reliable, the errors explained above are used to compare different solutions.

Gesture detection can be used everywhere the developer has continuous data, in real-time applications as well as for recorded data. Data can be collect from different sensors like accelerometer, touchscreen, image (camera), and much more. Some algorithm have a high precision, other are not as good as others, but they are faster to calculate. For the framework it's important that the calculation can be done in real-time. If the user has carried out a gesture, it should recognize it fast. Another goal is to minimize both sources of errors (True-negative and False-positive). One problem related to the framework is, that the information for gesture detection which are collected with a device has different accuracy (value range). This makes it more complex, because one must prepare the data in the correct form before an algorithm can use it. A second important point is that the developer, who will use the framework, should be able to extend the existing functionality with his own gestures.

There already exist works which are concentrating on gesture detection with different devices. For example a preview work describes how it can be done with an iPhone [16], another work has the goal to recognize gestures with a Wii Remote Controller [17]. The goal of this works only was to recognize gesture and don't answer how good it works for interactions. They are based on a machine learning algorithm, namely on the Hidden Markov Model (HMM) [16, pp.12] [17, pp.12]. For the HMM, there are many steps needed to realize a system. At first it is necessary to train a model for every gesture. This is called training phase. The first step is to prepare the data. Often it will be transformed in a space where only the important parts of the information are represented. This step is called Vector-Quantisation [16, pp.28-42]. The aim of Vector-Quantisation is to reduce the amount of data, but without reducing the containing information [16, pp.8]. The HMM consists of two steps, called forward- and backward-algorithm. Both steps are necessary to calculate the properties of the network, which it starts at the beginning with the data and the output will be assigned to a predefined category. In this case, the gesture can be true or false or it is possible to train it for more gesture than the system also has more output for every gesture. To train a model, first the network will be setup with different states in the middle network and calculate the probability. At next it compares the data with the training data and do backward correction from the output to the input to actualise the network with the correct information [16, pp.18-21].

Training a HMM network is the part with the most complexity, but if it is done once for one gestures, the recognition rate is really good. The recognition for difference devices are near the same. For example, the iPhone, it lies around 84-93% [16, pp.41] and for the Wii controller the recognition rate is near the same [17, pp.14]. But what the work also tells is that training takes lots of time and is far away from real time [16, pp.42]. This is a negative point, why this method would not be used in this work.

Another approach to check the data if a gesture was made comes from the audio analysis, namely to compare two different signals. If they are enough similar (under a certain threshold), it can be said that are the same. If this is the case, the system can say both signals are the same. Basically, there exist two methods. One method is to calculate the correlation between two signals to determine if they are the same. The problem is that gestures are not always made in the same speed and the similarity between a template and the current movement becomes quick too little. *Dynamic Time Wrapping* (DTW) is a further developed method for detection gestures. It compares the values of two dataset and determine the lowest distance between all points. The distance measurement can be the Euclidian, square or other techniques. If the distance is under a certain value, the system detects it as correct. The thresholds can be determined by recording a gesture several times. With these records, the developer can train a model for a gesture, or choose a value manual [18][19].

2.2 Interaction

With the possibility to detect a specific gesture, the developer is able to build an interaction system, independent of any domain. One aim for VR is to make the manipulation with the environment as realistic as possible. That includes making gestures like golf swing, simulating a tennis game, or other real world interactions. On the other hand, this technique feels not much realistic, because in real, the user doesn't have a tennis racket. The feeling will be better, if the person, who is located in the VR, can move his hands or even the whole body freely. Then the user can catch other objects like a bottle, throw it, or hit a ball with the foot like a football.

Chapter 3

Device analysis

3.1 Requirements needed for interaction

Nowadays we are surrounded by digital devices and we can't escape anymore. Some of them are useful to manage our day, we get notification for upcoming appointments and the communication between our friends is much easier than before. But our devices are not only suitable for human interaction in the real world. They gave us information about our activities, e.g. if we were running, the steps will be count. This is only possible with different sensors which the devices have built in, and it becomes more and more. With the properties of data collection for further analysis, it's possible to create an application to react of different user behaviour. The only requirement for interaction with the VR is that a sensor can collect data without interruption and there exists a way to transfer the data between components. In our case the data must be transferred from the producing device to the VR headset application. Another requirement is that the reading and analysing of the data is simple to keep the system fast.

The aim of this work is to find alternative devices in opposite to VR input controller to create a realistic behaviour of movement and interaction. This could be arm movements to translate them into the VR or simple gestures for certain key input. Another important point which must be considered is that that many users have the gadget at home. Also it should fit in the hands, not to be heavy and easy to use. Last but not least for some requirements, e.g. gesture detection, a high performance is needed for the calculation. It makes the analysis of the data easier, if it's possible to pre-calculate the necessary information in the producing devices to minimize the data treatment in the VR gadget. This minimizes the transferred data and processor power in one device because of parallelization.

Basically, sensors measure values which can occurs in different forms. Some of them can be seen by humans like the colours and depth of the environment, other can be felt like temperature. Another category is forces which can't be seen, but we can recognize them by their effects (gravity). All that kind of physical attributes can be detected with different sensors. We can divide them into following types [20][21]:

- **Position or Location sensor:** This type of sensor has the purpose to determine the position of the user or parts of his body. An area of application is for example Google Maps to locate the user in the world. It needs sensors like GPS, magnetometer,
- **Physical sensor:**
  - Motion sensor: The group of motion sensor handles the measurement of the current movement of the user. Basically it works with the measure of forces which are supplied to a device. Accelerometer, gyroscope and gravity sensor are one of the main components in this category.
  - Environmental sensor: As the name indicates, the user can collect information about his environment. E.g. light, temperature,
- **Camera/Optical sensor:** Can also be categorizes in the group of environmental sensor, but here, it often needs further processing steps to get the desired information. A picture with extra information like depth data needs to be combined meaningful (different sensor data) or image processing, etc.

For VR, the most useful category is the physical sensor. With them, the user is able to detect gestures and other movements. The same feeling can be done with a camera with optical sensors, if the system has a robust recognition of the humans' body. The following parts of this chapter describe different devices, how they are used in everyday life and which sensors they have built-in.

3.2 The Smartphone

3.2.1 Motivation

Smartphones are the all-rounders under the devices. They are the preferred communication gadget and almost everyone has one in his pocket. It's not only used to to call someone, but also for sending small messages to the opposite person with the help of messenger services, to look up on the internet for information, current location, healthcare (sport) and a big area is the gaming industry. This combination between the services makes it interesting for a VR application. The advantages of Smartphones for the VR are:

- It has different sensors that can be easy used. There exist a set which are supported by the most Smartphones.
- There is not only one way to exchange data with other devices. The developer can often choose between network and Bluetooth.
- You can write applications for them. This can be useful to collect and pre-calculate the data.
- Almost (not all) are small and fit in the users hand. This is necessary for the usability and simplifies the input.

The disadvantages which can be identified are:

- The accuracy of the sensors is not the best. They have much selection of different types, but they must share a small place and can't be so good like professional gadgets.
- The computing power varying much between different devices. Mostly, they are significant lower than personal computers and this must be considered for further implementation design.

3.2.2 Available sensor

As mentioned before, Smartphones are all-rounders and have many sensors, which the developer can access. It has physical as well as environmental sensors. For the VR, it doesn't make much sense if the system measures the luminous intensity, if the user is in a dark room or not. Maybe it is possible to simulate with this data the lighting in the application, but the reliability is questionable because the user can easy cover the sensor with his fingers by holding the phone.

More interesting for VR application are the physical sensors. With acceleration, orientation and gravity information it's possible to detect the movement of any kind of body parts, usually hand movements. You can determine with the help of the acceleration the direction, in which it's moving, with the gyroscope the rotation rate and orientation of the device. Often, it is only the Accelerometer necessary for a simple application, but with more than one sensor, the stability of the recognition system can be increased. In other words a modern phone supports everything the developer needs to realize a reliable gesture detection application.

With Smartphones, it is also possible to detect touch events carried out by the user. Wiping movements, as well as movement patterns can be detected or if it's necessary, how many fingers touch the screen.

3.2.3 Smartphone conclusion

Smartphones seems to be an ideal input device for VR application. The handling is easy and the user only moves his hands for gesture detection, can detect touches and mostly everyone have one. The only think the user should care is that the data which are available by the sensors don't have the accuracy like professional systems. The data has much noise and are not suitable to determine the current position of the device with regard to the body.

3.3 Microsoft Kinect

3.3.1 Motivation

The Microsoft Kinect has some restriction, because it only works for XBOX and Windows PC. This technology is from Microsoft and there exist currently two versions, the actual is KinectV2. In this work, only the KinectV2 will be discussed in detail. Many people think, the Kinect is only for gaming, but that's not true. The fields of application are diverse, for example the user can use it for translating sign language, to retrieve data with gestures for medical area, scanning different object to create a 3D model which can be used for 3D printing, and for this work as an input device for the VR [22].

To get an overview, a short list of description about the advantages and disadvantages will help to understand the device more:

- The usage of a PC allows the developer to make more complex calculation for analysing and transformation the data
- Microsoft Kinect was designed to create games, this opens the possibilities for new and intuitive interaction
- The optical sensors are stable and the recognition of the players' body is pretty good.

The disadvantages regarding VR are:

- The user can't use it where he wants. It always needs a PC and much place and a place where a socket and much room for the player is available. The user must be able to move free without any obstacle between the body and the sensor.
- They only can detect the body if the users' face looks near frontal to the devices, but in the VR, the user has a 360 view field. The developer must provide to alter the camera rotation with gestures in the application.

3.3.2 Available sensor

The Kinect works not with physically sensors like Accelerometer and Gyroscope, but there exist many possibilities with others like optical sensors. At first, the user can record the environment with a camera, which shows the user the currently situation in his room. These images can be used for image processing tasks, e.g., to recognize objects or detect faces and other body elements like hands and head. If the user moves his body, the camera can recognize certain patterns which can be used for interaction. Another feature which can be extracting from the camera data is to detect the face. For example it is possible to create a texture from a face and use them to put it on a model/avatar in an application. It's possible to create a virtual room where the opposite avatars look like a real player (virtual conference) [23].

A depth and infrared sensor goes hand in hand and is often used at the same time. It can be used to determine how the environment looks like (which objects are in the field view of the camera) with more information. In opposite the camera which recognizes what kind of objects are in the room, with the other sensors it's possible to scan the environment and create a 3D model from this data. The model can be used for the VR and simulate with extra features the real room. Another application of area is to detect a human and his movements. The API from Microsoft allows the developer to read the data from up to 6 users at the same time and split them up in different parts. These parts are called Joints and can be visualized as a skeleton. As well some states can be also recognized, for example if a hand is closed or not [23].

3.3.3 Microsoft Kinect conclusion

The advantage of Microsoft Kinect is the stable body recognition up to 24 Joints (points which can be detected). If the developer needs more information as he gets from the API, he can use the image to recognize objects in the desired accuracy. For example sometimes it's useful to detect the hands, but also the position of the fingers (virtual keyboard). With the delivered data it's possible to define gestures for movements, player rotation, etc., depending on certain parts of the body. Translating the users' skeleton into the VR world enables the emulation of catching or

hitting objects. One problem is, that the developer must design the VR application in a way that the user has not to look around much, because the device is stationary in the room and only works optimal if the user looks to it. All in all, Kinect has a high potential for an interaction application.

3.4 Smartwatch

3.4.1 Motivation

Smartwatches are relative new gadgets which are currently not much widespread like Smartphones, but they have some benefits as well. They are small like normal clocks and extend the usability and function of Smartphones, e.g., to make calls; see if the user has got a new message and many more. Mostly, the user can to the same task with a Smartwatch like he does with his Smartphone. The only think which must be considered is the limitation of resources and computing power. In other words there are more restricted than by Smartphones.

3.4.2 Available sensor

Like in Smartphones, modern watches have near the same available sensors. With the combination of accelerometer and gyroscope, it's also possible to build a gesture detection system or determine in which direction the user punches his hands. If a touchscreen is existing, it can be used for simple input trigger.

3.4.3 Smartwatch conclusion

It can be said, that everything someone can to with a Smartphone, it would be possible for a watch, but it is important to consider that they are very limited related to computing power and inputs possibilities. With computing power is mean that an application can only do low cost calculations and not to complex operations.

Chapter 4

Requirements for the communication

4.1 Design

The communication has the task to exchange the data between different devices. The data can occur in different forms, for example information which can have states (gestures detected or not, touch screen touched, etc.) and sensor data like position or body information. For data transferring, different technologies can be used. Two prominent approaches are Bluetooth and network over IP. Nowadays, many devices can handle both methods.

Regardless to the used technology, the structure is always the same. Basically, it can be distinguished in two types of applications, namely the server and the client part. The server application will be the device, where the VR headset is used. The client produces the information and pre-processes the data and brings it to the correct form. After the calculation, the data will be transferred to the server. Currently, the communication only goes in one direction, namely from client to server, but with not much effort it's also possible two make it as a two way communication. The design decision for one way communication has been chosen, because at the moment, the server hasn't any information which can be shared to the clients. It also makes it easier to manage the states of the communication. As well, for advanced communication and future work, different states can be send to clients, for example which gesture detection should be activate to save performance. 4.1 shows the architecture and elements of the communication part.

Most Smartphones provide Bluetooth and Network communication. The developer can use both and has no restrictions between the used technologies. On PC, there also exists a possibility for Bluetooth communication. The reason, why the framework doesn't support this kind of data transfer is, that not every PC has built-in a Bluetooth module (most notebooks has one). A second reason is that the used IDE for developing has a bad support for this kind of communication, because it isn't used often in this area of application. Due to the restriction, the framework determine at the beginning on which platform the application is running and offers the user only the type of communication, which are supported. More information can be found in chapter 6 "Implementation".

Regardless to the used technologies, it has following requirements:

- The data can be exchanged in **real-time**
- The communication is **reliable**

4.2 Requirement - Real Time Data Exchange

One important requirement for the frameworks communication part is that the communication between the different components is fast enough without high delay. The user shouldn't notice any lag between carrying out the input and the result in the VR. Well, this isn't always possible, because some sort of input need more time as other. For example some gesture detection algorithm needs time to determine, if the user has made a gesture or not.

To guarantee the real-time requirement, sometimes it's necessary to through some incoming data away, if the VR application can't manage them all. This problem can occur if the server framerate is lower than the client device. So it's necessary to design a solution that the missing data are not noticeable by the user.

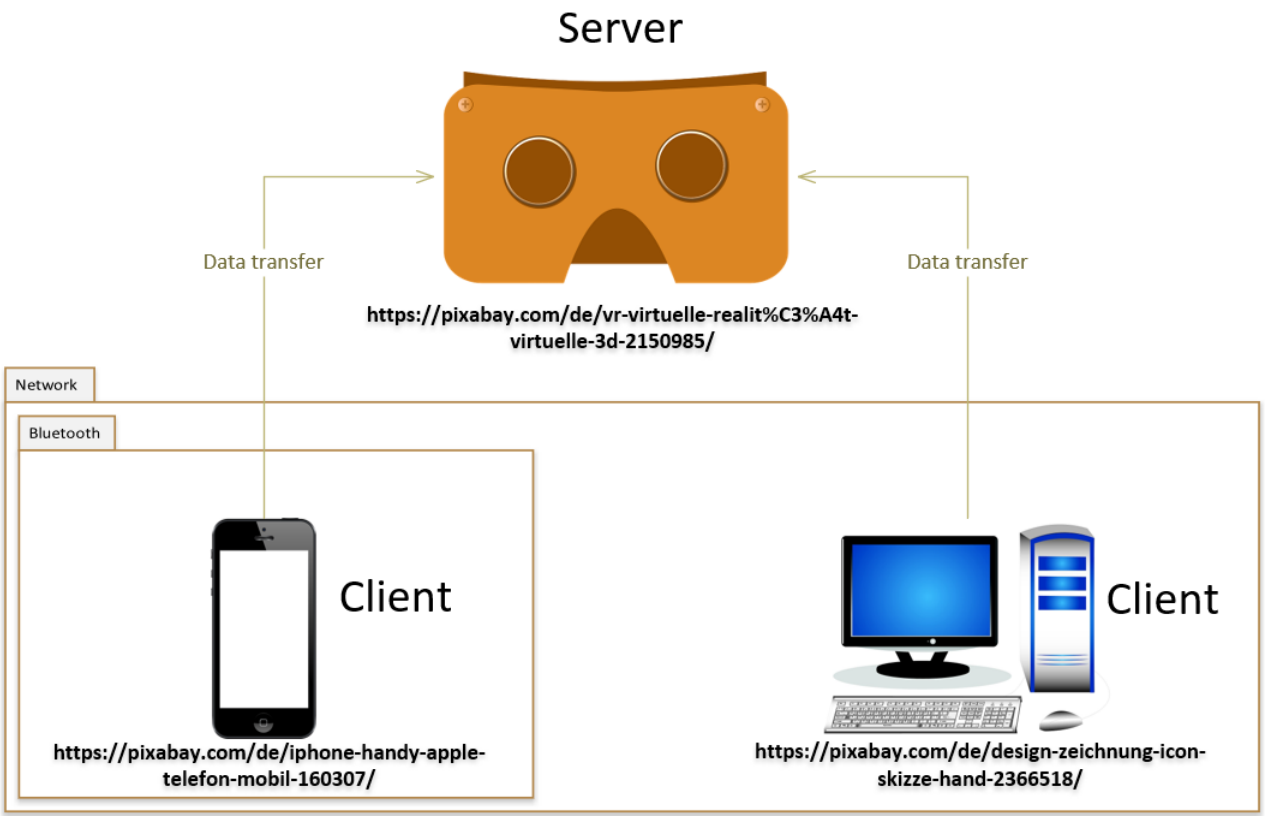


Figure 4.1: Overview of the communication architecture

4.3 Requirement - Reliability

As mentioned in the previous section, it isn't always possible to get all data from the input device, because of the different framerate between server and client. It also depends on the type of the data. We must distinguish between continues data (the application gets information which are depending on time) and discrete data (states are transmitted which doesn't change much over a certain time). In continues data the problem isn't as high as in the discrete case. Continues data are for example raw acceleration values or another example is the user carry out touchscreen input (measure the distance between fingers in different frames). Discrete data like states are for example determine if a gesture was made or the user touches the screen. That kind of data isn't allowed to throw it away, because the information has a high priority. Basically, this project has determined following data:

- **Continues data:**
  - Smartphone:
    - Acceleration data (normally not transferred)
    - Gravity data (normally not transferred)
    - Gyroscope rotation rate and orientation (in the normal case not transferred)
    - Touchscreen swipe and determine the distance between touch points
  - Microsoft Kinect:
    - Body Joints data
    - Sensor data (infra-red, depth, colour) (in the normal case not transferred)
- **Discrete data:**
  - Smartphone:

    - Is touch detected
    - Is gesture detected
    - States of the application
- Microsoft Kinect:
    - Is body detected
    - States of the application

Continuous data has the property that the value between different frames doesn't change much. If a frame gets lost, it's possible to interpolate the missing data from the previous frames. This can be only done, if the server doesn't lose too much frames. To get an image of the type of data and concept, an example will be described. If the user touches a screen on a phone, he has the feature to detect the current position to a certain time. During the input, he moves his finger to make a gesture. The distance of the touch point doesn't change fast. Only an error can occur by interpolating points is if the user change the direction.

In the case of discrete data, if a state was changed and the information doesn't arrive to the VR application is always a big problem. The server can't react if the user has made a gesture and it doesn't happen anything there. It can be annoying for the user and the lust for VR vanish. In the worst case he won't use it anymore. There exists an easy way to provide the reliable of this sort of data. One solution is if the state of the gesture detection changes to true, the client sends this information more than one time. This can be done as long as it's necessary, for example of the gesture detection system as long as the data window is full (see later). The same thing can be done by other states to transfer them more than once.

Chapter 5

Gesture detection methods

This chapter explain which method(s) the framework use for the detection of gestures, which data are considered and how to optimize the results. Another task will be how different gestures can be distinguished from each other if they are near equal. The gesture detection is based on data which can be collected by Smartphones. The algorithm can also be used for other devices like Microsoft Kinect. The framework supports only gesture detection for phones, but it can be easily adapt.

5.1 Introduction

At the beginning of the master thesis, a short overview has been presented, which kind of algorithm exists for gesture detection. The recognition rate is often the same, but some are more complex than other. For the work, some points are important to minimize the effort during the developing and maximize the positive result:

- Fast in calculation
- Good recognition rate
- Developer can extend the framework

5.1.1 Requirement - Fast calculation

The first point is, that the calculation of the current data must be done in real-time or in a speed that the user doesn't recognize the delay. If the requirement can't be reached, the interaction in the VR becomes weird and isn't suitable for long usage, because the user will get inpatient. Regardless to the used method, to speed up the calculation, the pre-calculation should be done in the client device. Only the state of each gesture will be send to the server.

The most time in Machine Learning methods is needed to train the system, where the input are record templates which includes the sensor data from the input devices. After that, the calculation of the current data can be done faster. The DTW can also be trained, but it must not. It's enough to compare two signals and run the algorithm on it to determine the threshold for further similarity measurements. The speed of this method depends on the length of the signal. Another method is to calculate features like min, max, zero crossing rate, etc. In this case, the calculation doesn't take much time and can be done in real-time.

5.1.2 Requirement - Recognition rate

The recognition rates of the discussed methods in the related work section are near equal to each other (for certain cases). In this point, it doesn't matter which method will be used.

5.1.3 Requirement - Expandability

An important point for this framework is that the implementation is hold as general as possible. The framework doesn't know from the beginning, which gesture it should detect and how the values of each thresholds are. It must be trained before the first usage. For Machine learning methods, the system must be trained and for that, the developer must record some data of the gesture. This must be done in a separate program or at the start of the application. But as mentioned before, it can take lots of time and not suitable for a short usage of the VR app.

In DTW, there is only need a template which can be loaded on start. To determine the similarity, the current data must be compared with the template data and a distance will be calculated. If the distance is smaller than the set threshold, it will detect the signal as a gesture.

For the feature detection method, there is also need a pre-recorded set of data to compare them later with the current data which is produced by the device. The calculation can be done at the beginning and is fast. Advantage of this method is that it can also be done during runtime.

5.1.4 Requirement - Conclusion

Table 5.1 represents an overview of the important points which must be considered for the framework.

Requirement/Method	Machine Learning	DTW	Feature-Extraction
Training	- -	-	++
Pre-Calculation	- -	-	++
Recognition rate	++	+	-
Real-time	-	-	++
Extendibility	- -	+	++
Effort	- -	-	++

Table 5.1: Compare different gesture detection methods

As it can be seen, the method with the best recognition rate, the resource which is needed is high to realise a gesture detection system. This is a problem because the developer should be able to extend it easily, but the results at the end should also be acceptable. DTW seems to be ideal, but if the size of the signal is high, the consuming time to calculate the distance can be too much. This can be a problem if many gestures must be detected at the same time. The method with the fastest calculation, which the column Feature-extraction shows, has a bad recognition rate.

The idea of the following discussion of gesture detection is to combine the DTW method with the Feature-extraction method. At first the features will be calculated and eliminate signals which couldn't be the same. At the next step, the DTW calculation runs over both signals and calculates the distance and finally makes the decision. How the DTW works and which features will be calculated will be discussed in the next section 5.2.

5.2 How the algorithm of the used methods works - DTW and Feature-Extraction

5.2.1 Dynamic Time Warping (DTW)

DTW is a method to compare two different signals. The easiest way to explain DTW is to take all data points of a signal and calculate the distance between them. If the sum of the values is under a certain threshold, which can be learned or set manual by the developer, the signal will be recognize as the same. One signal is the reference data set, for example a template which was recorded before, and will be used to compare it with other templates or with real-time data [24, pp.69].

In the case of the framework, a window with a certain size will be considered. The size depends on the template length. There are two data handler, one for the template and one for the current collected data. If the window

is full, the calculation starts. One of the questions is why the DTW will be used and not, for example Correlation which also looks if two signals are linear depending from each other. Correlation has one big disadvantage because the two signals must have the same duration, but gestures wouldn't be executed always with the same speed. So the recognition rate will be really bad if the user made a gesture slower or faster as the template is. Well, to solve the problem of the time depending component, DTW is used.

In DTW, the decency of time will be minimized, so the recorded template and current data can have a different velocity. To show how it works, there exists a framework for C# which doesn't only implement the algorithm, but also has included a visualisation tool [25]. In this tool, the user can choose two different countries and which variable should be investigated. To explain the algorithm, the country Austria and Germany has been chosen. The DTW will be used on the variable fish and seafood.

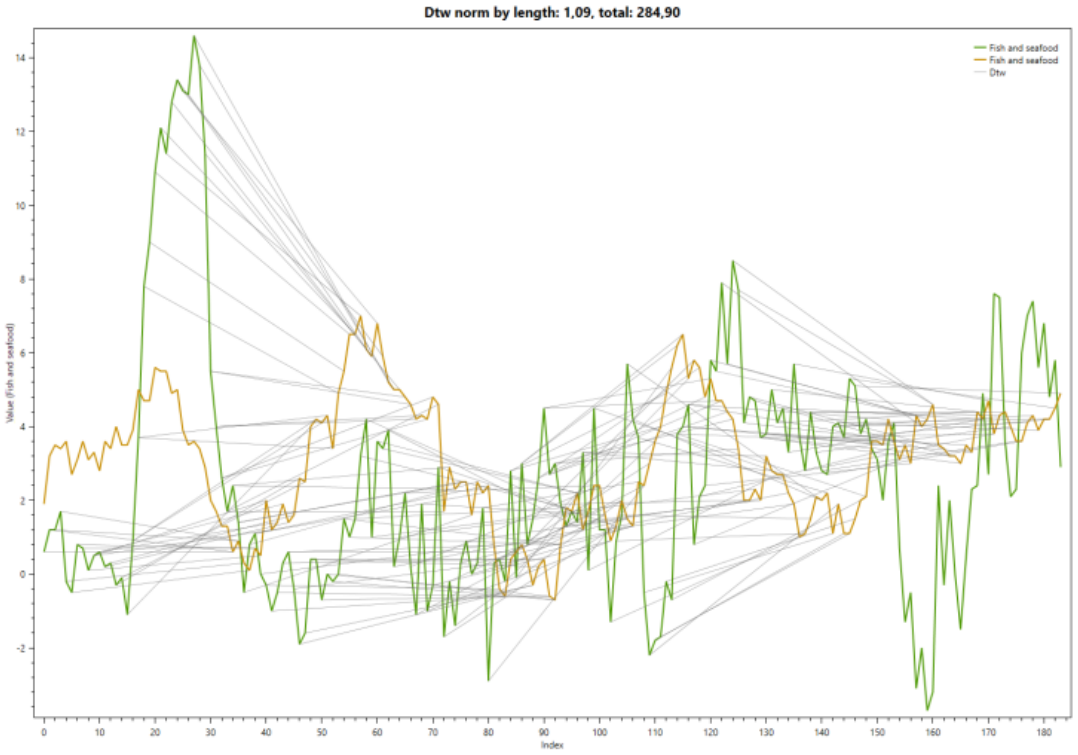


Figure 5.1: Example of the two time series
x-Axis: Index - 0 - 180; y-Axis: Value (fish and seafood) - -2 -14
Green: fish and seefood data Austria, Orange: fish and seefood data Germany, Gray: Dtw

The green line representing the behaviour of Austria (signal A), the other line is associated with Germany (G). The grey line in the graph between the two signals shows which data point from A has the lowest distance to a point of G. The aim is to find a path with the minimal cost between two signals. The first which must be done is to calculate a cost-matrix of all data points. This depends on which distance measurement the developer prefers. The ndtw-framework supports four different methods:

- Manhatten Distance
- Euclidean Distance
- Squared-Euclidean Distance
- Maximum Value

For the showcase in chapter 6, the Euclidean distance has been used. Figure 5.2 shows the cost matrix of the example above. The first step is to initialize a matrix with n columns and m rows with an initial value, for example

zero. The size is given by the length of the signals. If the matrix exists, the next step will be to calculate the distance between every data point. As mentioned above, the developer can chose any method what he wants. The pseudo code represents the simplest way how the calculation can be done. Note that the variable **c** is a placeholder for the used measurement:

Algorithm 1 Pseudocode - Cost calculation

```
1: procedure CREATE MATRIX
2:     n = size(signal1) //length of signal 1
3:     m = size(signal2) //length of signal 2
4:     M = new Matrix(n,m) //create matrix with the size of the signals
5: procedure CALCULATE DISTANCE
6:     for i=0; i<n; ++i do
7:         for j=0; j<m; ++j do
8:             m[i][j] = c(signal1(i), signal2(j)) //calculate the distance with the used method
```

Every field in the graph represent a matrix entry. A black field means that the values are the same. It is the best result the application can has, because the distance is in this case zero. The brighter the blue is, the more the distance for these two points is. The worst case is red, followed by green. To determine the shortest path, following algorithm can be used [24, pp.71]:

Algorithm 2 Pseudocode - Find shortest path

```
1: procedure FIND PATH
2:     DTW(n,m) := c_p(n,m) = min(c_p(n,m) | p is an (N,M)-warping path) //find the path in the matrix, where
   the cost are at lowest (minimum)
```

This is the simplest case to calculate the DTW, but isn't really efficient. For real time gesture detection means that it's necessary to calculate for every frame the whole matrix. If many signals should be compared, it can lead to a calculation overhead and the frame rate breaks down. The same problem occurs if the length of the signals is high, than the time for the calculation increase exponential. Related to this example, A and B has the same length, the runtime behaviour is $O(n^2)$.

To reduce the numbers of calculation, there exists some optimization for the DTW algorithm. One step to reduce the count of operation is to define a band. The band has an upper and lower boundary which is not allowed to cross. The path can be only between these two lines. The boundary is defined by the middle line of the matrix (n,n) and consider the x and y dimension. Outside of this will be not considered anymore and doesn't need to be calculated. Figure 5.3 shows a band with a degree of freedom by +- 50 [24, pp.76-77].

There are exists many methods to find the path. One way is to determine how many points should be considered. That has the effect that not every x value has a y value (see 5.3). To define this behaviour, a parameter called step size can be set. Higher value means that less calculation must be done, but is not exactly. With a step size of one, every value of one signal (the shorter one) will map to a value of the second signal (5.4) [24, pp. 74].

A gesture has a certain start- and an endpoint. This can be also seen in the data. At these certain points, the values are often near zero. At the moment of moving the device, the values are changing and at the end, it becomes to a resting position again. With this assumption, another option for DTW can be introduced. It is possible to tell the algorithm that the path must start at the point (0,0) and should end at the point (n,m). This forces the detection system to have a certain start and end behaviour. It helps to distinguish different gestures from each other. A second benefit is that the user has a reduction of the distance calculation, as it can be sees in figure 5.5 and 5.6.

5.2.2 Feature extraction

Feature extraction helps us to get a fast overview of a signal to check if it's possible that the data represents a gesture. The calculation for the most features in the time domain can be done with little computing power. This is the first step of the recognition system. The following features will be supported by the framework:

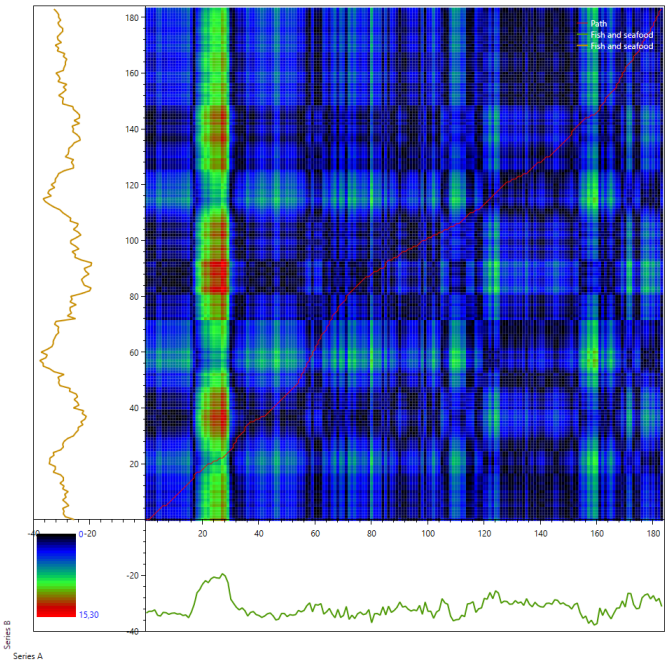


Figure 5.2: Distance matrix with Euclidean measurement
x-Axis: Series A (Austria), y-Axis: Series B (Germany)

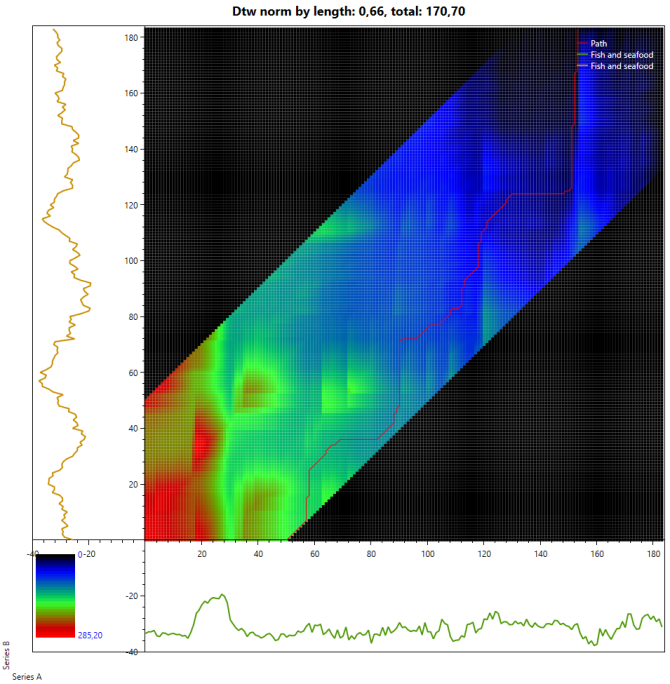


Figure 5.3: Optimization 1: max band shift (Sakoe-Chiba band)
x-Axis: Series A (Austria), y-Axis: Series B (Germany)

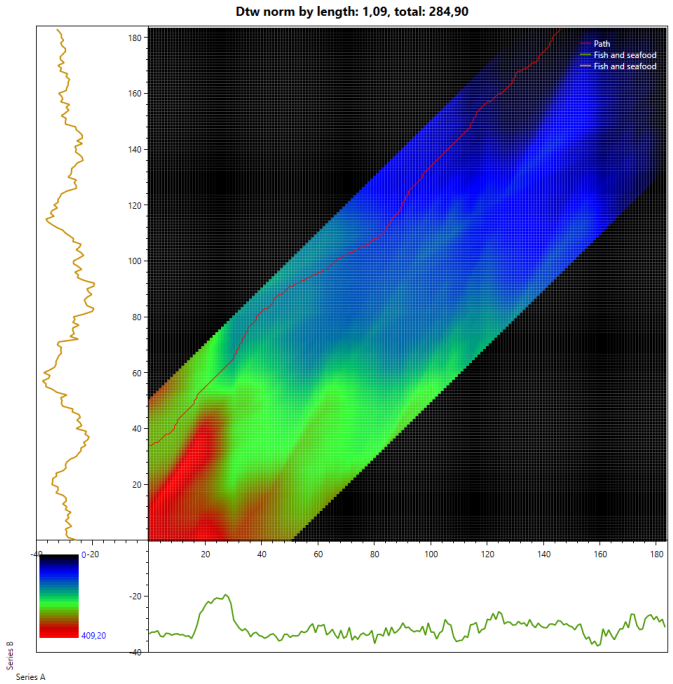


Figure 5.4: Optimization 2: For every x-value assign a y-value
x-Axis: Series A (Austria), y-Axis: Series B (Germany)

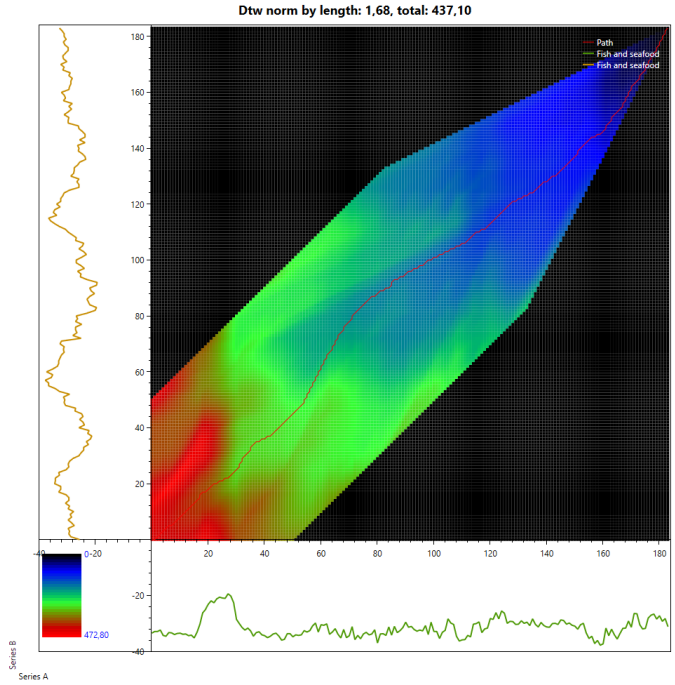


Figure 5.5: Optimization 3: Set end and start value to the same
x-Axis: Series A (Austria), y-Axis: Series B (Germany)

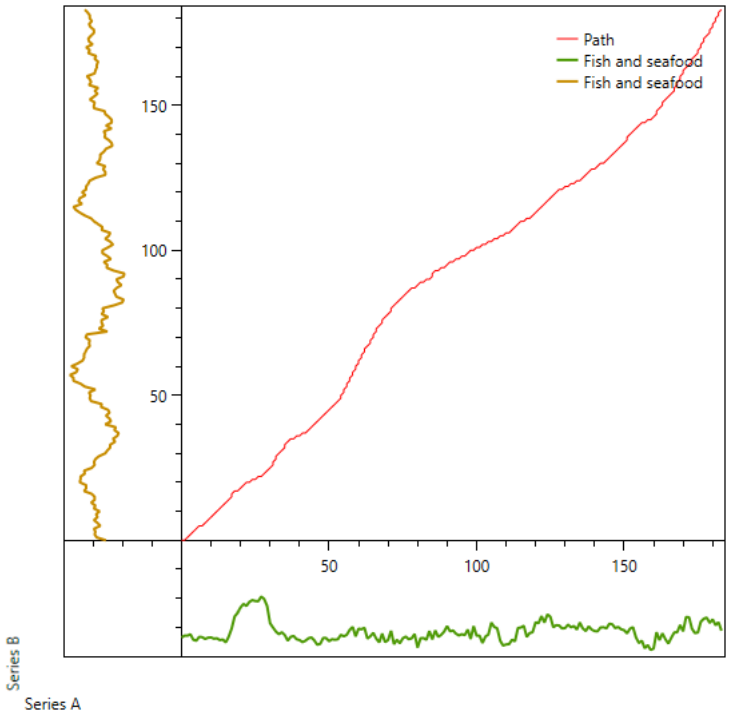


Figure 5.6: End result with restriction with minimized cost path
x-Axis: Series A (Austria), y-Axis: Series B (Germany)

- Same sign
- Below min/Above max
- Below max/Above min $\Longrightarrow$ Define a band where the data should be
- Correlation (deprecated $\Longrightarrow$ wasn't suitable because of different speed of carrying out a gesture)

Same sign

Sometimes the values of a signal don't change the sign over the whole time. Either the measuring points are positive or negative. A special case exists, where it isn't clear to which category it belongs, namely zero. It will be categories in both areas and doesn't affect the test result. This is one of the fastest calculations which can be done. A possible application area is for example to use them on gravity data. Sometimes a device has always near the same orientation and the consequence is that the sign doesn't change (figure 5.7).

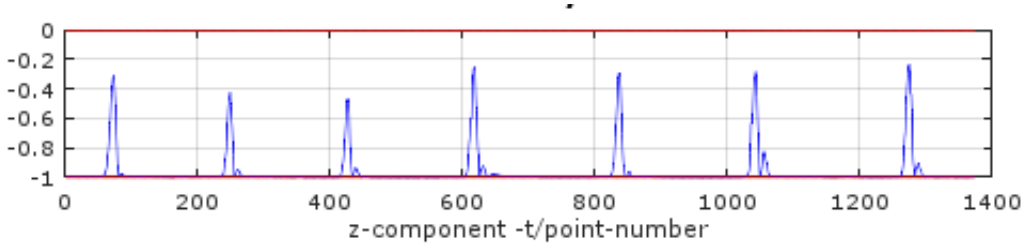


Figure 5.7: Example feature same sign - all values are lower than zero

Algorithm 3 Pseudocode - Same Sign

```
1:  counterPos = 0;
2:  counterNeg = 0;
3:  procedure SAMESIGN
4:      for i=0; i<n; ++i do
5:          if value[i] == 0 then //if zero, it will count for positive and negative
6:              ++counterPos;
7:              ++counterNEg;
8:          else if value[i] < 0 then //for negative counter
9:              ++counterNeg;
10:         else
11:             ++counterPos; //positive counter
12:     if counterPos == windowSize || counterNeg == windowSize then //returns only true if whole window has
    the same sign
13:         return true;
14:     return false;
```

Below Min/Above Max

This feature can be used for cases, where certain peak in the signals appears. Every non zero signal has a maximum and a minimum. But it's only suitable if a value exits which can be determine as important, for example a measuring point is much higher or lower as the mean of the time series. Figure 5.8 is a window of a punch gesture which is based on the acceleration data. The two red lines are thresholds which must be crossed by the data. If this isn't the case, the gesture won't be detected.

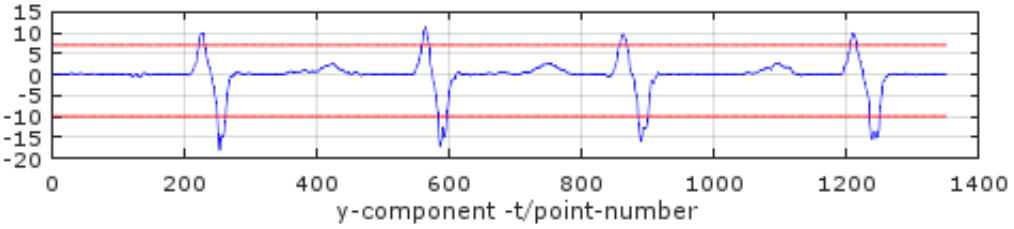


Figure 5.8: Example feature below min/above max - the signal must cross these thresholds

Algorithm 4 Pseudocode - Below Min

```
1:  procedure BELOWMIN
2:      threshold = ...
3:      for i=0; i<n; ++i do
4:          if value[i] < threshold then //test if one point in the data is lower as the threshold
5:              return true;
6:      return false;
```

Below Max/Above Min

The equivalent to the previous feature is that a signal is not allowed to have a higher (Below Max) or lower (Above Min) value. If this is the case, the gesture detection system will report it as false. A scenario where it can be used is to detect a gesture with minimal changes in one axis. An example where it can be used is the gravity sensor. If the device is tilt too much, the absolute value gets higher than the threshold. Figure 5.9 shows a combination of the last two features. It can be combined freely how the developer needs them.

Algorithm 5 Pseudocode - Above Max

```
1: procedure aboveMax
2:     threshold = ...
3:     for i=0; i<n; ++i do
4:         if value[i] > threshold then //test if one point in the data is higher as the threshold
5:             return true;
6:     return false;
```

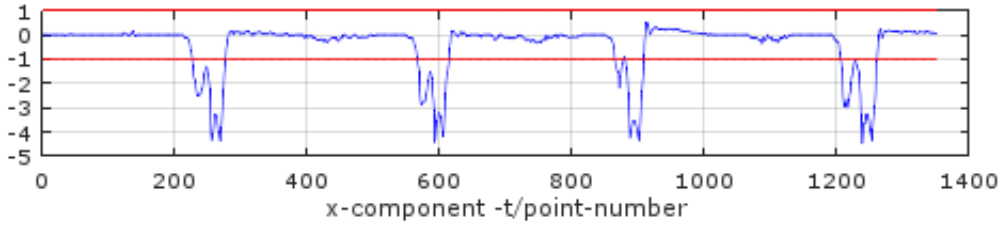


Figure 5.9: Example feature below max/above min - combination upper red line not allowed crossing, the lower red line must be minimum

Algorithm 6 Pseudocode - Below Max

```
1: procedure belowMax
2:     threshold = ...
3:     for i=0; i<n; ++i do
4:         if value[i] > threshold then //test if one point in the data is higher as the threshold, if this is the case, return false
5:             return false;
6:     return true;
```

Algorithm 7 Pseudocode - Above Min

```
1: procedure aboveMin
2:     threshold = ...
3:     for i=0; i<n; ++i do
4:         if value[i] < threshold then //test if one point in the data is lower as the threshold, if this is the case, return false
5:             return false;
6:     return true;
```

Optimization

Each of these algorithms can be optimized. It's not necessary for every frame to consider all values, because in most cases, only a new value arrives and the latest gets useless. The framework uses a window, where all data are saved. If a new data frame starts, it took the first element of this window and through it away and reset the influence of the value. After that step, the new data will be checkt, if of condition are met and return the current state.

5.3 Conclusion

This section has explained different methods and which options exists for building a reliable gesture recognition system. It's possible to combine different methods to improve the performance on different measure data. Also the sensibility of the recognition can be optimized to avoid wrong detected gestures. There are many ways how the aim can be reached. It depends on how simple it should be at the end. Simple systems are fast and can extend the functionality with not much effort, but with the disadvantage that the recognition rate is not as high as by complex methods. As mentioned before, the framework will support a combination of DTW and Feature-extraction. Figure 5.10 represents a block diagram with the process step which the data must pass:

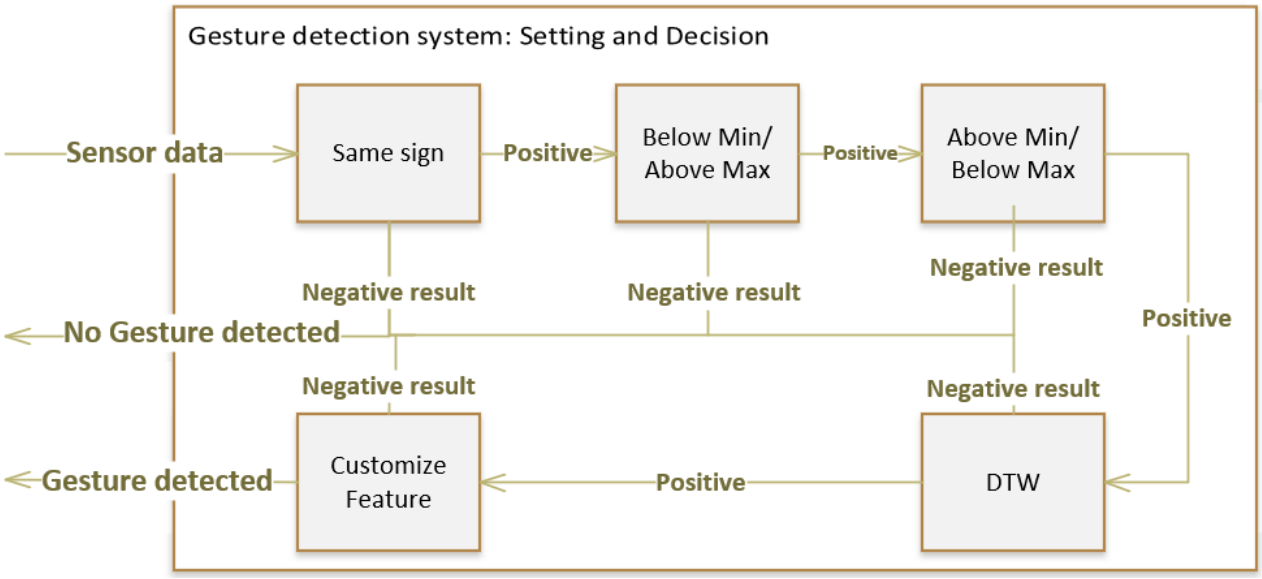


Figure 5.10: Sequence for gesture recognition

The data goes through different calculation steps, shown as a block. In each block, another feature will be calculated. It isn't restricted on any kind of data. This can be for example acceleration, 3D data or orientation information, it's only important that they are changing over time. At the beginning, the effort for the calculation and making a decision is simple and doesn't need much computing power. The more the signals goes to the right side, it becomes more and more complex. If a feature doesn't match with the settings (threshold), the pipeline will immediately abort and becomes the state false. If the signals pass through all station, the systems detect it as a gesture.

If we take a look on the two middle blocks, the order has a certain reason. At first the system looks after peaks in the signal. If there isn't trigger a min or max value, it returns false. The next is that the signal doesn't get to high or too low on the viewed window. If the value is higher or lower as the given boundary, it becomes false. The first features which triggers with peaks, the user must move the device to reach the value. The next feature with the boundaries, the system will detect them always as true, if the user doesn't move the input gadget. In other words the second block has a higher dropout rate as the third one. To avoid unnecessary calculations, this sequence was chosen. On the right side, the last block represents the DTW part where the template signal and the live data are compared to equability. This part needs the most computing power and will only carried out, if the other states were all successfully tested.

Later in the work with regard to the extendibility, the developer can add his own features, called customized feature. Because its not known how complex the calculation is, it will be calculated as the last one after DTW.

Chapter 6

Design and Implementation

6.1 Introduction

This chapter presents most important parts on the thesis and implementation of the framework. The most interesting questions for the developer is how he can use this framework, how he can extend it and what is needed to successfully create a project. The following overview presents which parts are included:

- What kind of technologies are needed to realize a VR application
- Explained the different parts:
  - Devices and his data
  - Bluetooth/Network
  - Gesture recognition
  - Entry menu
- To explain the implementation process, a showcase will be created. This includes the integration of a device, the setup of the gesture detection system and an example to evaluate the functionalities.

Important for this section besides the implementation part is the analyses section. The framework will deliver data in unprocessed form which can be loaded by other, extern programs. Important is that for analysing and editing the signal data, it can be saved after the calculations (thresholds, remove noise, etc.). This template file can be used by the framework and it will automatically calculate the necessary features as discussed in the last chapter 5.

The showcase(s) described in the next chapter 7 will show how the framework works and explains the options in detail. Another reason for the implementation is to have some data for the discussion presentation chapter 8. For every used devices, an own showcase will be create to test the limits and possibilities for interaction methods. For the Smartphone, a fairground with different attraction will be designed which needs a good interaction realisation. For the Microsoft Kinect, it should be shown, how the user can move in the VR with simple gestures. That includes move forward/backward and rotate the player object. For further testing and interaction, a skeleton of the user should be also drawn into the scene and translate the body motion 1:1 into the application.

The entry menu is the start point of the application. It depends on which platform the program is running. For example on Smartphones, the user can select different connection options (Bluetooth, IP over network), but for PC, only the network option is available. Another point is to distinguish the type of the device. For the server as well for the client, different requirements are needed. The server will start immediately the VR app and setup the connection settings automatically, whereas the client needs information about IP address or UUID.

6.2 The tested devices

In chapter 1, a short analyse of available and upcoming VR headsets has been done and in chapter 3, there can be found information about different devices, why and how it can be used for VR applications. Based on this, following combination is used for this project:

- VR headset: Google Cardboard
- Input devices: Smartphone with Android OS, Microsoft Kinect

Basically, every VR headset can be used, but "Google Cardboard" is simple and cheap so that a fast start is guaranteed.

6.3 Used technologies, frameworks, engines, and devices

There exists many ways to realize a project with different tools. Some of them are more useful than other. This section will describes, which tools are available for creating a VR application, what properties they have and if and how it is possible to combine all components to a complex structure, which represents the finished framework. The structure is composed of implementation and analysis part. Not included is the showcase at the end of this master thesis. It's used to evaluate the project under different perspectives like gesture detection rate or how easy is it to build a new application with it.

The framework itself has not the aim to use it for commercial purpose, so this makes it easier to handle licences during the development process. The developer, which will use the framework, must handle this for his own, if he is using it for commercial purpose. Another aim is to use only programs and tools which are free available. It simplifies the usage of the framework for other persons, who haven't tools for much money. So everyone, who is interested in developing an interactive VR, can do this.

6.3.1 Programming part

At first, we must solve the problem of the used devices, on which platform the "programs/tools" are running and how complex the integration of diverse plugins into the environment is. As before mentioned the work concentrates on two devices, namely Android Smartphones and the Microsoft Kinect.

Basically the goal is to do something in a VR, in this case to interact and manipulate the unreal world. So the first question will be how it's possible to create a VR app. As discussed in the preview section about the master thesis, the prototype is based on the Google Cardboard technology, but it's not restricted to this special kind of headset. So the framework should be compatible with different solution and exactly for this problem, Unity is most suitable for that.

For Unity3d, there exist various plugins for different VR technologies and other useful tools like OpenCV, etc. (not needed, but maybe for the future to handle object detection with a camera). If the developer needs some functions which are not available, he has the possibility to extend the framework for his own special case. Basically Unity has the purpose to create games, but it is also possible to create simulations, frameworks and many more applications. The biggest advantage is that it runs on many different platforms (Windows, Linux, MacOS) and the developer can also create applications for PC, Web-browser (WebGL), iOS, Android, XBOX, etc. There exist two versions, one version is for the free usage and the other one, the developer must pay a small amount per month. The different between the two versions are how the developer uses the finished application. For non-commercial purpose, the developer must be under a certain profit (under 10.000$), otherwise he must choose the payed version. There is no different which features they have included. This means that for developing the framework, the free version is suitable.

With Unity, the development is very fast, because it has a build in physics- and rendering system, so the developer can fully concentrate on his own problems entails the project. The concept is to have building blocks, which can be created with different properties and behaviour. The behaviour can be influenced dynamically with scripts. Two prominent script languages which can be used in Unity are C# and JavaScript. Both of them are suitable and only depend on the preferred language of the developer. As well it's possible to switch between the languages. For this project, C# is the preferred script language.

Well there exists a problem, namely Unity doesn't support Bluetooth, so an extern plugin must be used or the developer must write it by his own. There is no free plugin for this purpose available, so the plugin will be created per own. The main task what it should solve is to handle the communication between two Android Smartphone devices. The easiest way to create a program for Android is to use Java and there exists a free available development toolkit called Android Studio. Unity allows the usage of plugins from other programming language in form of dlls, libs, etc. With Android Studio it's possible to generate a lib file of the finished code. After that, this library can be used as a reference to any project where it's needed. The developer can access different parts of the plugin with AndroidJavaClass to instantiate objects from these classes or call methods.

For the communication over network, Unity has a collection of classes and objects which handles the data exchange. You can use finished GUIs and settings, but to have the best result, the exchange of data will be handled by the framework itself. The reason for this decision is that Unity Network synchronizes the data from the server to one or all clients, but in this case, it's necessary that the clients can send the data to the server. A client doesn't influence the other clients. In other words they are completely independent from each other.

As mentioned before, it should also be possible to recognize different gestures. It represents another part of this projects with own questions and solutions. For the most feature, it's possible to calculate them by own. For DTW, there are many possibilities to solve them, to restrict the solution space and optimize the calculation performance. To avoid wrong solutions or performance problem, an external library is used. For C#, the ndtw library is suitable for this project. It's free and easy to use, but it must be adapted. The library was created with .Net 4.0, but Unity works with .Net 3.5. Some feature doesn't exist like the Tupple class.

Beside the Smartphone, the Microsoft Kinect is used to test interaction in the VR. A SDK exist for Unity [26] which can be downloaded and integrated to the project. In the SDK, a sample program is included to read the different sensor data like camera picture, Joints and depths. Parts like to read the body data will be used. It takes the classes and extends the functionalities for drawing the skeleton.

6.3.2 Analyse part

At the previous part we have examined the different tools which are available to realize coding this project, but there exists also the problem, how to analyse the data which the devices produce. The question is why we need the analyses part. One of the aims is to recognize gesture made by the users. But this is only possible if we look after how the values changed over the time. As well, it's important to see if different gestures can be reliable distinguished from each other to avoid wrong results. Every signal has a certain window which represents the gesture signal (not everyone does take the same time to perform it). This should be able to select it and determine which features are suitable. On this time window, an easy comparison is possible of different data which are splitted up into various parts, e.g., acceleration, gravity, touch sensor and much more.

For analysing the data, Matlab is state of the art. A reason why it's not used for this project is that there is a licence needed, which isn't cheap and breaks the idea of everything is free. But there exists an alternative to Matlab, namely Octave. You cannot expect that everyone who uses this framework has a Matlab licence and thus ensure the possibility to analyse the own gestures to extend the system later, if it's necessary.

6.4 Handling different devices and their data

6.4.1 Concept

Basically, Smartphones and Microsoft Kinect will be supported by the framework, but it should be also possible to integrate other types of gadget for testing interaction concepts in the VR. The developer should have an interface, where he can add his own information. To provide the flexibility, placeholders must be defined which can be easy access anywhere in the project.

The entry point for all devices is the class InputData, which is a singleton pattern (figure 6.2). The developer can ask for the instance with a static method. If the call is made by the first time, the object will be created. As before mentioned, the Smartphone and Kinect device are used in the showcase and will represents, how a customized device

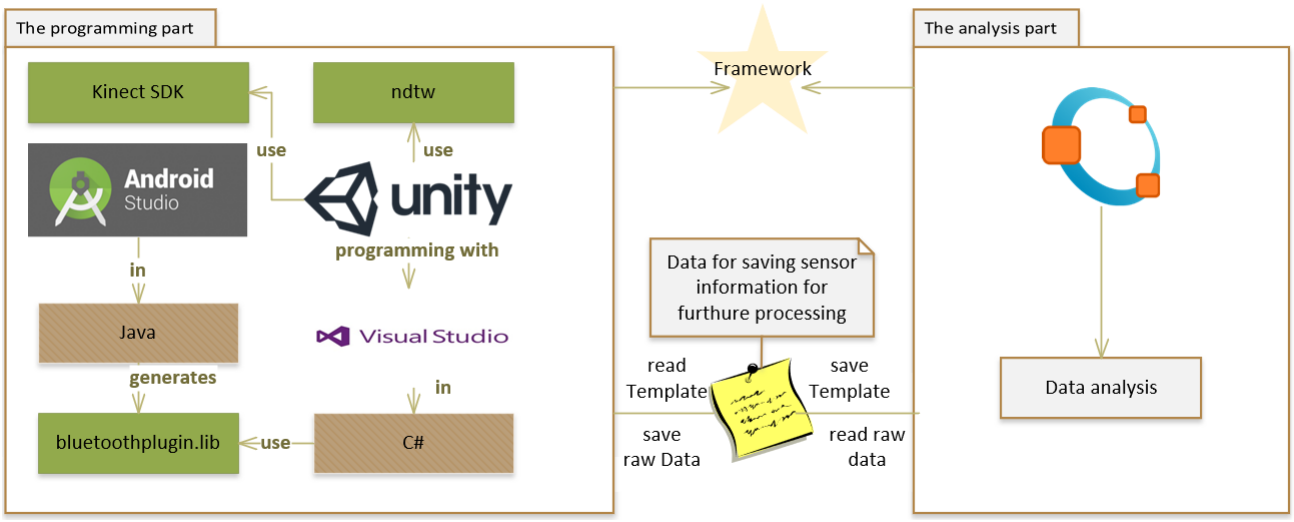


Figure 6.1: Engines, libraries and tools used for this framework

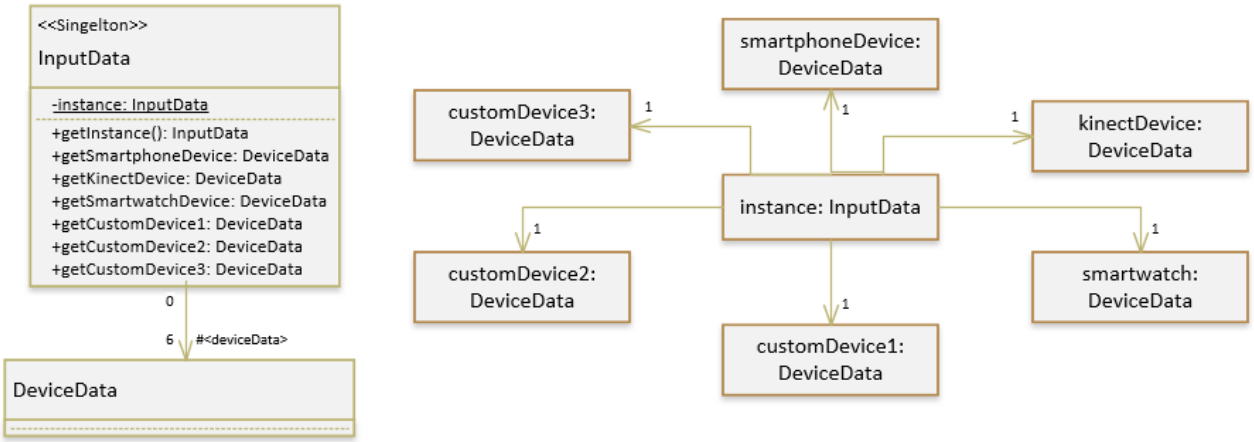


Figure 6.2: Concept overview of the devices

can add to the project. Mainly all devices in the InputData object are instantiated, but at the beginning, they are empty. If the developer wants access for a certain DeviceData, he can call the associated getter method. A setter is not allowed, because the device shouldn't be changed directly, but it can be changed indirectly with references within an object. Figure 6.3 shows the parts of a DeviceData:

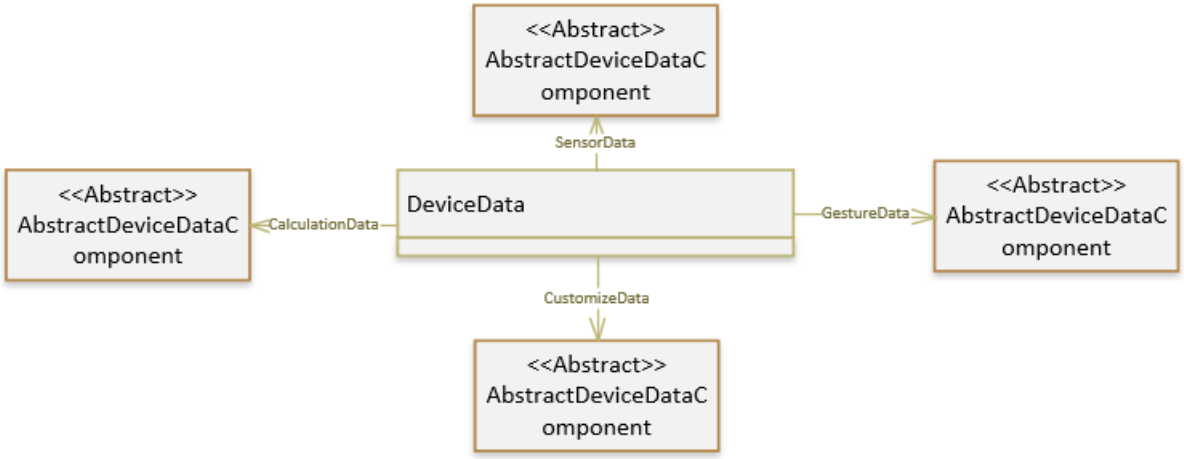


Figure 6.3: Concept of the DeviceData

The DeviceData class contains the information of diverse sensor and gesture data, but also information about states or necessary calculations. The developer can decide for his own, what kind of data are defined in the different groups. The framework offers four variables for define the data. The categories are:

- **SensorData:** As the name says, this area is to collect information about diverse sensor data. This could be acceleration, gravity, touchscreen information, but also position and depth data.
- **GestureData:** The gestureData section holds the information of diverse gesture states which can be detected by the application. In the following sections, more detailed information how it works can be found. Basically, all devices with the possibility of gesture detection have the same basic class for holding the data.
- **CalculationData:** This section contains data which are results of some kind of calculations. For example rotation angles of a Smartphone.
- **CustomizeData:** The last area can be used for any kind of data. Sometimes it can't be related to a certain group. The developer is free to put everything in it, what he wants.

As it can be seen above, the four categories are all from the abstract class AbstractDeviceData. What the AbstractDeviceDataComponent is will be discussed in the section "Network implementation", because it contains primarily methods which must be implemented for the communication (prepare the data). Moreover, the child class must be marked as Serializable. At the start of the application, all sections are null. The developer must set the references to an object which represents the data structure. A recommended function to do that is the Awake function. Unity will call this method at first, before the first frame is starting.

The framework has setup the Smartphone and Kinect with basic information:

- **Smartphone:**
  - **SensorData:**
    - This area has information about the touchscreen sensor. The developer can set different states like if the screen is touched by the user or the distance of the touch point between two frames. It doesn't contain data like acceleration or gravity, because they are not needed in the server application. Often it's used to determine if a gesture was made and only states will be transferred. This minimizes the amount of data which must be sent by the client. But if the information is important, the customized data area can be used.

- **GestureData:**
  - Initialized by the standard gestureData class which includes the name and the current state of a gesture.
- **CalculatedData:**
  - Contains a 3D-Vector which contains information about the rotation angle around the three axes (x, y, z). This calculation is based on the gravity vector and depends on the device orientation.
- **CustomizeData:**
  - Empty

- **Microsoft Kinect:**
  - **SensorData:** This section is setup with a structure to provide information about the detected body. It's divided into different parts and every Joint is represented as a 3D vector.
    - MainBodyData
    - LeftArmData
    - RightArmData
    - LeftLegData
    - RightLegData
  - **GestureData:**
    - Empty, but setup with the standard gesture information class.
  - **CalculatedData:**
    - Empty
  - **CustomizeData:**
    - Empty

6.4.2 The setup in detail for a Smartphone

Basically, the data will be setup at the start of the application by the framework. Three out of for areas are used as it can be seen in the last section, namely the SensorData, GestureData and the CalculatedData. If the developer needs more information, he can use the CusomizeData variable.

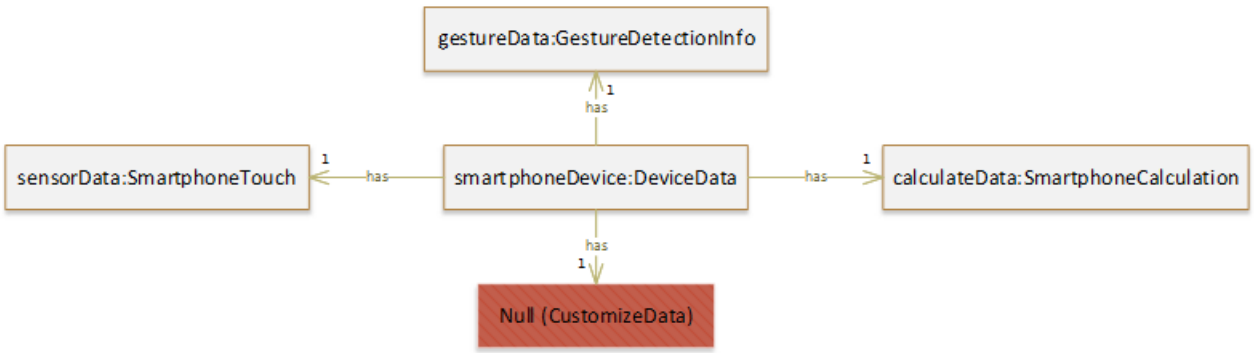


Figure 6.4: Objects included in the smartphone device

In the Unity scene, there exist a class for every AbstractDeviceDataComponent which has the task to actualise the information. To do that, a game object must be created to attach the collector. The base class is the MonoBehaviour which will be run through different states every frame. It can work independent from the rest of the project. This objects only have one task, namely to collect the data, calculate information or set the states to the associated category. Figure 6.5 represents the simplified model for the Smartphone device and his collector scripts:

More information how the gesture data will be actualised can be found in the implementation section.

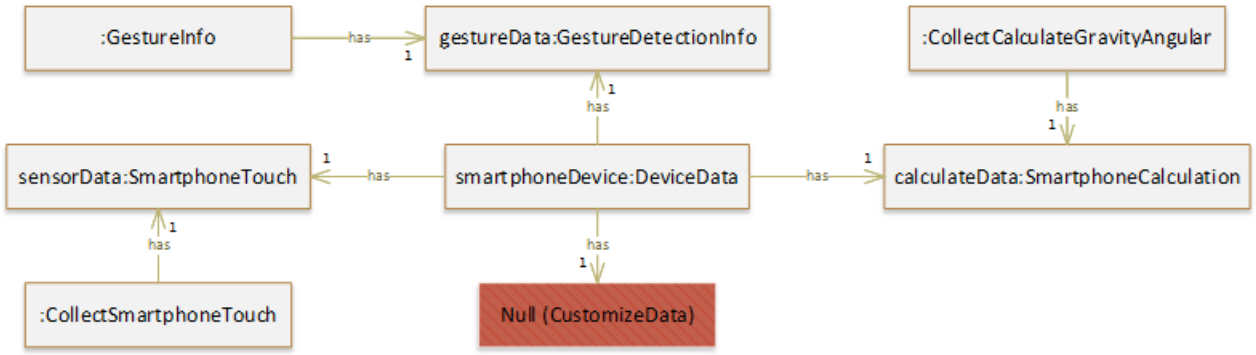


Figure 6.5: Simplified model, but complete for the Smartphone device

SmartphoneTouch

As the name indicates, the smartphoneTouch class has information about the touchscreen. It belongs to the area of sensor data. There exist different variable which handles the touch input. Important is to have information about the current state of a touch input. To realize it, two enumerators are needed.

- **TouchGesture:** The enumerator contains information about which gesture was made with the touchscreen. For detection, a simple method will be used, other than the discussed methods before like DTW or feature extraction. It is considered in which direction the finger goes along the touchscreen. At the beginning, up-, down-, right- and left-movements can be detected by the system. To avoid permanent detection of a touch gesture, the user must move the finger a certain distance. This threshold can be set during the development.
- **TouchGestureMode:** Sometimes it's important to have a possibility to distinguish between different modes for the same touch gesture. To distinguish the mode, a simple method is used. This depends how often the user taps the screen. For a simple touch, the mode named "slow" is set. If the user carrying out a double tap, the "fast" mode will be used. An area of usage is to realize movements in the VR environment. As the name indicates, the user can walk in two different speeds, fast or slow.

This is not the only information which handles this script. Sometimes it's important to know, how much fingers are currently touching the screen. For that, a normal integer variable is used. A second important thing is to know how the distance between the first touch and the current position is.

GestureDetectionInfo

This class has two arrays which are depending of each other. One array contains the name of the gesture and the other one handles the states and it's of the type "Boolean". It will be created and set at the application start automatically from the gesture detection system. The default value is false, equated with not detected. In the case that the user carried out a gesture during runtime, the gesture system set them to true for the related device(s) of the InputData object.. This happens every time at the end of a frame, more precisely in the LateUpdate function. The gesture detection system manipulates these arrays and at the end of every frame, the data will be updated in the InputData instance object. How it exactly works, the information can be found in the section "Gesture detection concept for Smartphone".

SmartphoneCalculation

Unimpressive but sometimes necessary is to have information about the orientation of the device. For this purpose, there is only need a 3D vector of the type float for every axis. It contains the value about the current angle which depends on the gravity vector.

Important is to know, in which direction the axis are to calculate the different angles with the gravity vector. Every phone has following axis alignments which figure 6.6 shows. The developer must set the orientation at the beginning

and restrict which are allowed or not. This is necessary because every time, the program orientation is changing, the axis changes too. In this case, all pre-recorded data becomes useless. The same applies for the calculation of the rotation angles.

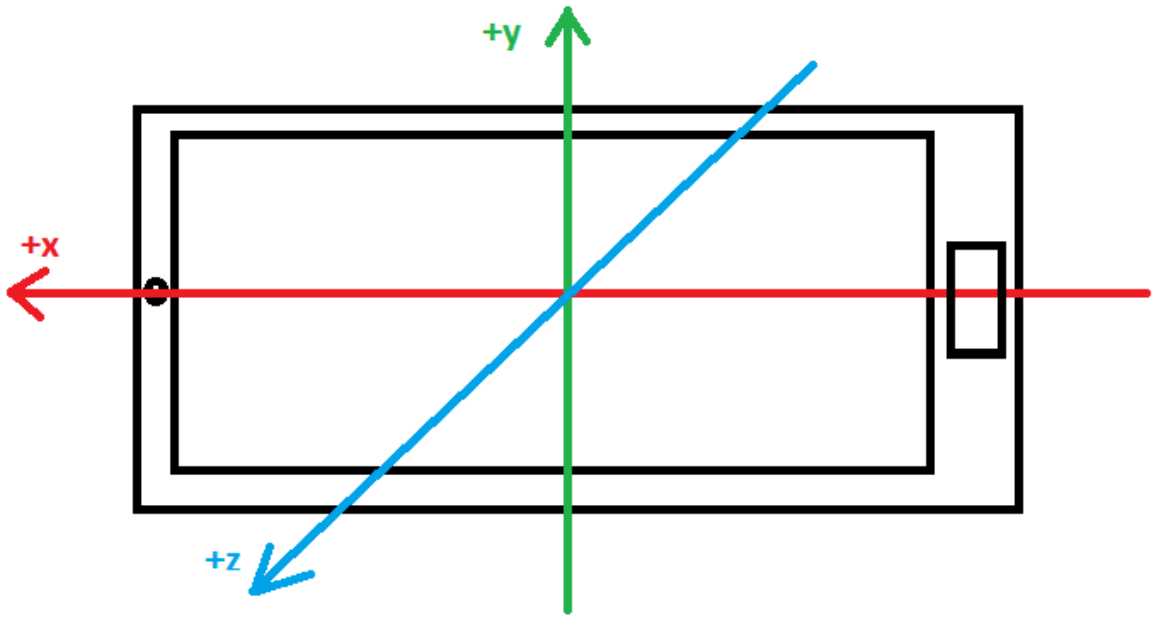


Figure 6.6: Axis of a phone, if program is set to the orientation "landscape left"

A simplified calculation helps us to determine an angle of a certain axis. Note that it works only for two axis at the same time (x and y). The z axis can be used, if the user holds it like a car wheel in his hands. In the following formulas, g is meant as the gravity vector which the Smartphone currently measures.

$$cos(angle_Z) = ((g.y, g.x, 0.0).normalize).(0.0, -1.0, 0.0) \tag{6.1}$$

$$cos(angle_X) = ((0.0, -g.z, -g.x).normalize).(0.0, 1.0, 0.0) \tag{6.2}$$

$$cos(angle_Y) = ((g.y, -g.z, 0.0).normalize).(0.0, 1.0, 0.0) \tag{6.3}$$

Note that the equation above are only optimized for the following showcases and can handle only simple device orientation. For more complex tasks it's necessary to use the gyroscope. This delivers a quaternion about the current orientation, but it's more complex to use them. Reasons for the complexity are the axis of the Smartphone doesn't match with them in Unity, and second it also depends on the magnetic field. To use it, the developer must calculate all vectors to a reference space to make it usable for the interaction.

CustomizeData

CustomizeData is a placeholder for the developer to add more information to the Smartphone device. Currently it's empty.

6.4.3 The setup in detail for the Microsoft Kinect

As with the Smartphone, for the Microsoft Kinect, there exist also pre-defined data structure and collector classes. Only one area is used from the beginning, namely the SensorData part. It is of the type of "KinectSensorData". The information is splitted up into five groups, which can be determined by the Joints [27]. The Kinect delivers the information about the position of the different Joints as a 3D vector. That has the consequence that every part of

the body (hands, head, etc.) is represented as a vector. To get the data, the class "BodySourceManager" is used, which is available by the SDK itself. The other three are empty and can be free used by the developer.

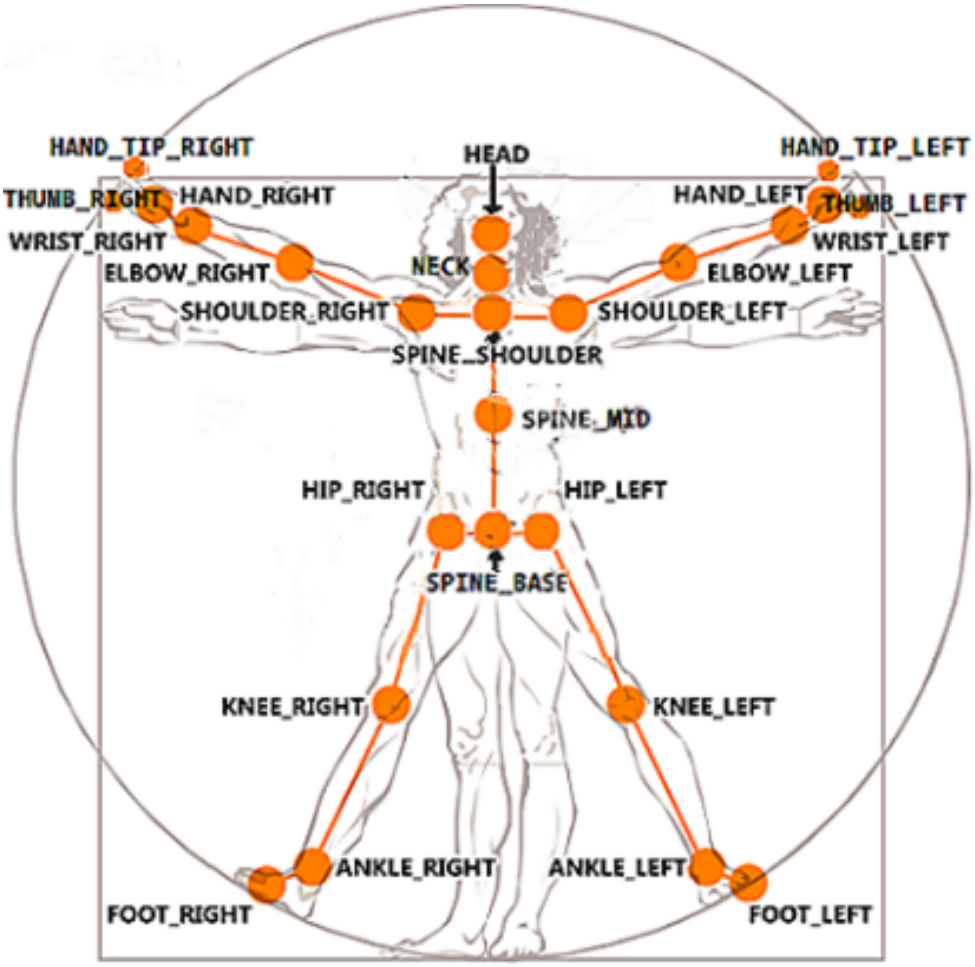


Figure 6.7: Joints which can be read from the Kinect [27]

As mentioned before, the sensor data is splitted up into five groups. One category is a collection of Joints which represents a part of the body, for example the left arm. The advantage is that later in the VR application, the developer can turn on or off some body parts during runtime (avoid interaction/drawing). The following list shows which information can be found:

- **Main Body:** The main body data structure contains information about the position of the middle part of the body. The fix point of all measured points is the Spine_Base Joint. For that point, the 3D vector has always the value zero. The value of all other points is represents the distance to the Spine_Base. Consequence is that if the user moves the whole body (move away from the camera), only minimal changes can be determine, because the Spin_Mid has done the same movement. In the case, that only parts of the body are moving, it can be translate to the avatar in the VR.
  - Head, Neck, Spine_Shoulder, Spin_Mid, Spine_Base ⇒ MainBodyData class
- **Left Arm:** The left arm has information from the left shoulder to the left thumb.
  - Shoulder_Left, Elbow_Left, Wrist_Left, Hand_Left, Hand_Tip_Left, Thumb_Left ⇒ LeftArmData class
- **Right Arm:** Same information like the left arm, but all for the right side.

- Shoulder_Right, Elbow_Right, Wrist_Right, Hand_Right, Hand_Tip_Right, Thumb_Right ⇒ RightArmData class

- **Left Leg:** The left leg is defined from the left hip to the left foot.
  - Hip_Left, Knee_Left, Ankle_Left, Foot_Left ⇒ LeftLegData class
- **Right Leg:** Same information like the left leg, but all for the right leg.
  - Hip_Right, Knee_Right, Ankle_Right, Foot_Right ⇒ RightLegData class

For both arms, there also exists a Boolean for the hand state to distinguish if it's closed or opened. With the information about the position of body parts and hand states, it's possible to define more complex gestures. Necessary is this to activate some functions which are important for the VR interaction. A possibility is for example to rotate the camera or move the player. A good rule is to start a gesture with closed hands and end it by opening them.

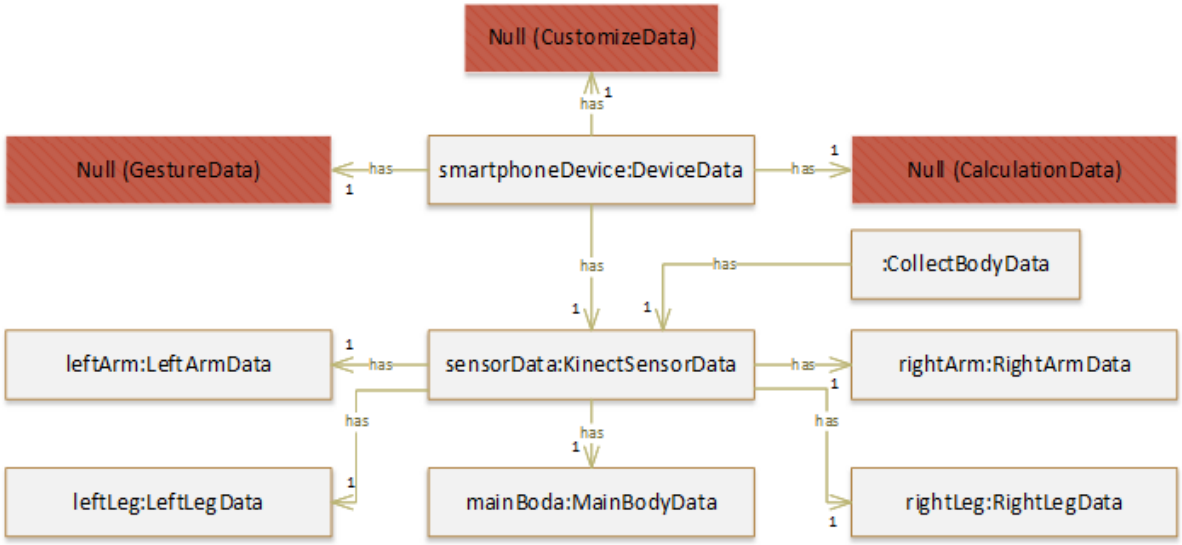


Figure 6.8: Realisation - Microsoft Kinect device data

The framework has a GameObject with the name "CollectBody". This can be used in every scene, where the developer needs to ask for and actualise the sensor data. There exists an option, which kind of data should be collect by the system. It's possible to activate or deactivate different groups. If it's not activated, the data for this section will not be collected. It also effects the VR application, where the interaction and drawing function will be deactivated for these body parts. The options are main body (MainBodyData), arms (LeftArmData and RightArmData) and legs (LeftLegData and RightLegData).

6.5 Communication with devices

The developer has two possibilities to transfer the data from the client to the server. One factor which method is available depends on the used device. On the one hand, some devices have a Bluetooth chip build-in, on the other hand every modern gadget has the option to create a network connection. An aim for this framework is to design the communication in a way, that it supports both communications. The challenge for this framework part is to minimize the effort to build the communication. In the previous section, the work has shown how the developer can add new data to a specific device. As mentioned, they all have the same parent class, namely the abstract class AbstractDeviceDataComponent. Another requirement is that a class which should exchange data must be marked as Serializable. This is important for the usage of Bluetooth, where the data will be serialized into binary code, sending the data over the connection to the server. If the data rich the aim, it will be transformed back into the origin objects.

The purpose of Unity is to create games. It's possible to implement applications which are running local, but also it's possible to connect with friends over the internet. Unity offers an own collection of network implementation which can be freely used by the developer. In the normal case, the communication direction is from the server to one or more clients. During the developing, variable can be defined with the keyword "synchronise". As well game objects are available to setup the information for the connection. Important attributes are IP address and port number for the server. For the client it exists an own GUI to set the information during runtime.

For this project, the communication direction must be reversed. The clients must send the information to the server, where the VR app is running. Also the data must be assigned automatically to the right device (Smartphone to Smartphone etc.). The concept to do this is for Bluetooth and IP over network near the same. A problem occurs because network logic must be derived from the class "MessageBase" which is responsible to handle the data transfer between the devices. Also it has defined different methods which can be overwritten by the developer. Two important functions are Serialize and DeSerialize. The developer can override them, but it's not mandatory. If it's not explicit declared, all public variables will be automatic transferred, otherwise the developer can decide by own which data should be send and the order can be defined for the serialization. To ensure that for all classes the transfer works, the abstract class "AbstractDeviceDataComponent" also defines the two methods to handle the transformation.

Another problem which must be solved is that the MessageBase class is not Serializable, so it can't be used by Bluetooth at the same time. To guarantee it works for both communication methods, a container class is defined, which split up the functionalities. The network definitions don't affect the Bluetooth components and vice versa. So far to illustrate this section, 6.9 represents a model of the concept of the different parts:

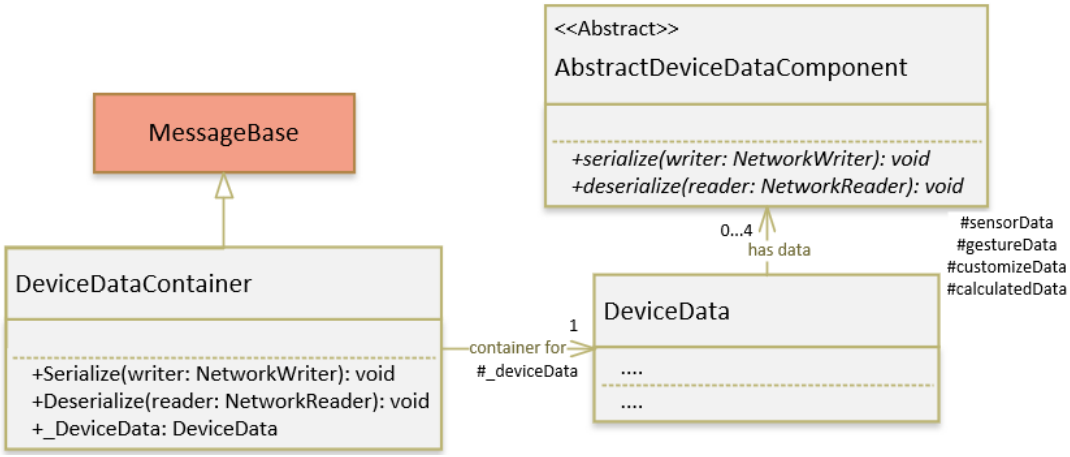


Figure 6.9: Concept of data handling for both communication methods

Note: For the Bluetooth part, a plugin has been written in Java to handle the connection building and data sending/receiving. The reason for that is, Unity doesn't support any possibility to create a Bluetooth communication.

6.6 Gesture detection concept for Smartphone

The largest part of the framework is the gesture detection system. It has many tasks to do. One task is to collect the data in real-time which is needed to calculate the defined features. Also a second feature is that the user can record a gesture and save them as a simple format in a txt file. An advantage of the txt file is that everyone can read it (user as well as applications). The raw data can be used to process the data and bring them in a form, that a template can be used by the framework. Later in the showcase, only the Smartphone uses the recognition system, but it's not restricted to that device. General for all devices the possibilities the developer can create gestures with all available gadgets.

6.6.1 Introduction which data is needed

As mentioned before, the developer can use different Smartphone sensors which deliver different kind of data. Sometime it's better to use as many data as it's possible to make the recognition more robust. Because of the simple methods which are used in the framework, other data is needed. Figure 6.10 are signals of two different gestures, based on the acceleration.

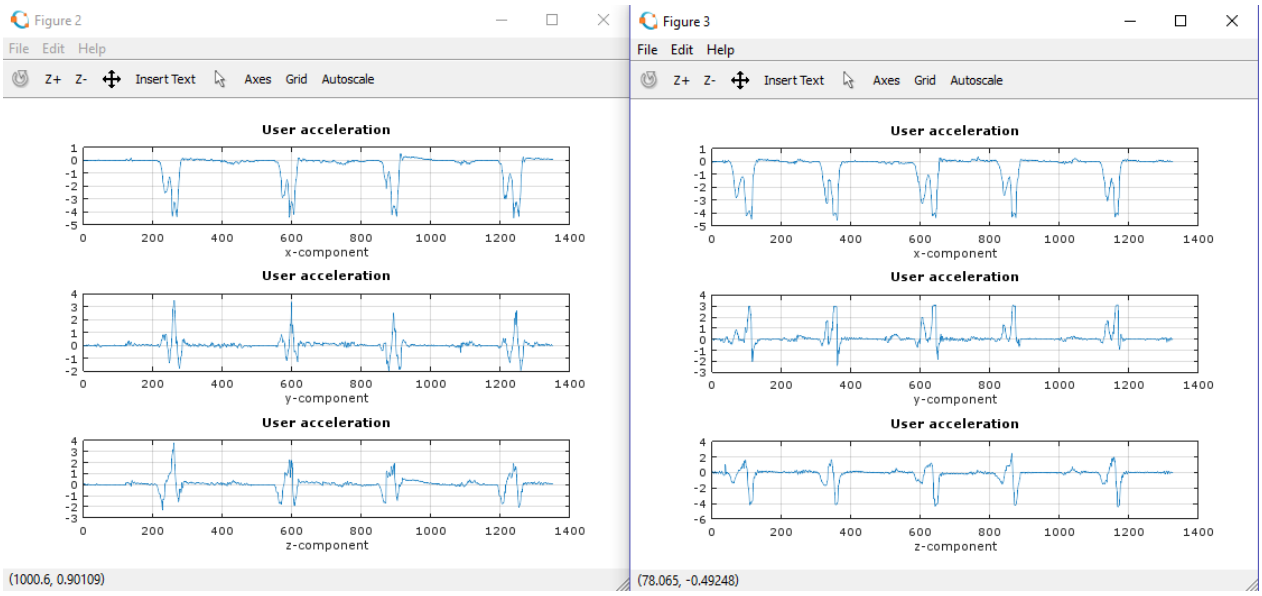


Figure 6.10: Two gestures which looking near the same; Left-Golf, Right-Tennis

As figure 6.10 shows, the tow signals don't distinguish very much from each other. Especial the x and z component of the data seems to be identical. Only the y component is a little bit different, but for robust gesture recognition, it isn't enough. So other data must be considered to improve the performance. For that, the Smartphone also deliver data like gravity and rotation rate which can be combined as it's needed. During the feature settings, the developer can decide which sensor data should be used and the associated threshold.

6.6.2 Recording data

The first problem which must be solved for this application part is how the data should be collected. There are two areas where the data is needed. It must be available for the real-time recognition system as well for the part which is responsible to save the raw data in a file. To solve this problem, the framework offers a game object called "SmartphoneCollector". It is designed in a way that every class which needs the information can get it. The only thing which must be done is to registry it to this class. If the frame ends, the classes which are registered become a notification and the data of the actual time point. This frame-view contains the current acceleration, gravity, rotation rate data as well as magnetic heading and gyroscope orientation. It can be used for all tasks without restriction,

but ideally a class solves only one problem. The last two types of sensor data (magnetic heading and gyroscope) are currently not used in the showcase.

Both parts of the recognition systems, the recognition itself and the saving part, holds the data into a class object called "WindowSmartphoneData". The idea behind is to make the data available for a current time span. If the callback reaches the class where the information is needed, the data will be saved in different arrays. The size of the window for the real-time recognition depends on the length of the template which is read at the start of the application. During runtime, some feature needs all data points, like DTW, and compare the template with the actual window. If the window is full, the oldest data point will be removed when a new one arrives per callback. In the case of saving the data, no limit of the size of the window exists.

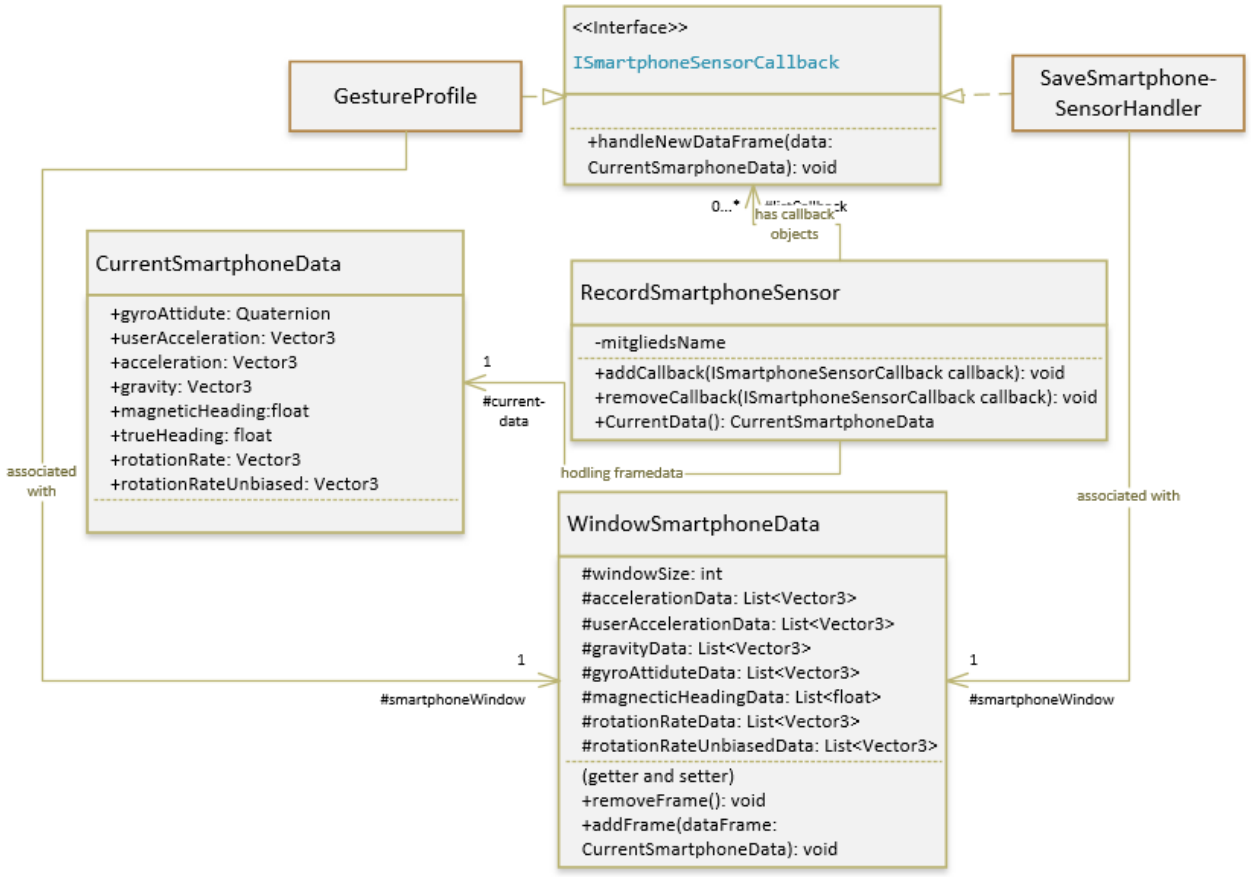


Figure 6.11: Overview of the Smartphone data record

In the case of saving the data, no limit for the window size exists. The window will be filled if the user clicks a "start record" button to begin the recording. This will happen as long the user decide to save them. After the saving, all data points are removed and as a result, the window is empty again. This can be replayed as much it's necessary. To avoid overriding the data with existing files, the possibility to give the file a name is offered in form of an input field. The corresponding menu can be seen in figure 6.12. The top GUI element represents the input field and the button below starts the recording. If the button is clicked by the user, the data will be collected as long as the user clicks again the same button. Always it happens, the text changes. During the data collection, the user sees the information to "Save data". After clicking the button, the file is saved in the application folder of the app with the specified file name. To use this part, there exists a finished scene and is part of the frameworks menu system where more information can be found.

The format in which the raw data will be saved has a certain form. It's always the same and if new data is defined by the user, he can append them at the end of the file. After reading the data in an external program for

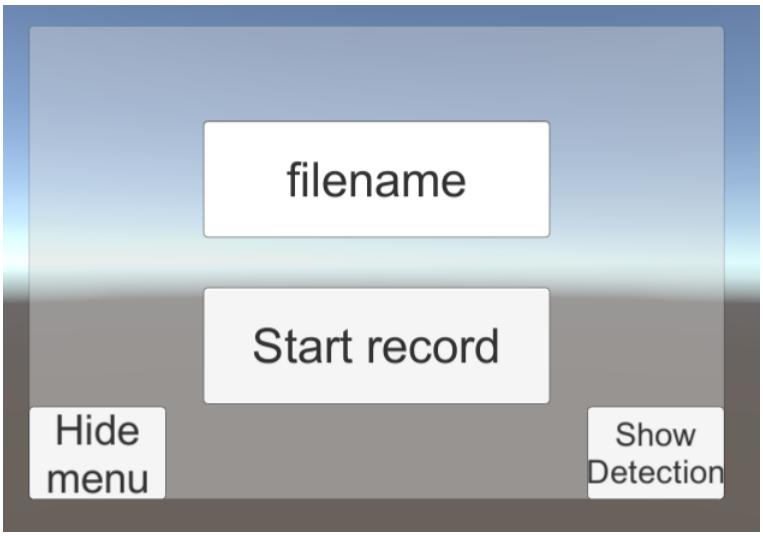


Figure 6.12: Record detection menu

analysing, it can be saved in the same format as the framework structure them. If this restriction will be considered by the developer, the framework can read in the template with no problems and the gesture detection system will work correct.

All values in the file are of the type float. A row consists of a series of values which are separated by a comma. Every value represents a certain time point during the recording process. Each row must have the same number of values or the system throw an exception. If the default sensors are used which are supported by the framework, it will not happen. The rows are again divided into three groups which represents a vector. These groups are the recorded data of different sensors. For example acceleration, user acceleration, rotation rate and rotation rate unbiased. How it looks like, a picture of the different rows can be found in Appendix A.

6.6.3 The gesture detection system

The last section has shown how the data for the Smartphone are collected and how the relation between collector and object, where the data is needed, is. Also a short introduction and characteristics of the two parts has been done. Related to the gesture recognition system, at first it's necessary to define the elements which can be identified to realize such a system. As shown in figure 6.11, it contains only classes for handling the data collection. Figure 6.13 specify the gesture detection system in more detail. The entry element is the "GestureProfile" class. With the help of the callback, it gets the data from the game object SmartphoneCollector. It also represents the start point of the gesture detection system, where all gestures can be defined. For every defined gesture, it handles the data transmission to each GestureInfo objects. Later in Unity, the GameObject which is responsible for the recognition is called GestureDetection and has two child objects. The first is called "DefaultGestureDetection", where the GestureProfile and GestureInfo scripts are attached, and the SmartphoneCollector for collecting the data acceleration.

As mentioned before, the GestureProfile is the entry point of the system. On start, the class setups all gestures with their settings. This means that in the first step it looks after what kind of gestures should be recognized. If at least one GestureInfo scripts is attached on the responsible GameObject (DefaultGestureDetection), the templates are read in the system and pre-calculate all information based on the selected feature. Also the window is created where the data about a certain timestamp is available. The size of the window depends on the template with the most measurement points.

After the reading of the template data, it looks after the activated feature with their thresholds and options. For every gesture, the developer can set a name to distinguish it from each other. This name will be used in the next step to set the state in the InputData instance in a hash table. The data structure is always actualizing if the application gets new GestureData from a client or at the end of the frame. Due to the characteristics of the InputData, the developer can access the states everywhere in his project.

An important property of the framework is that the developer can setup as many gestures he wants without

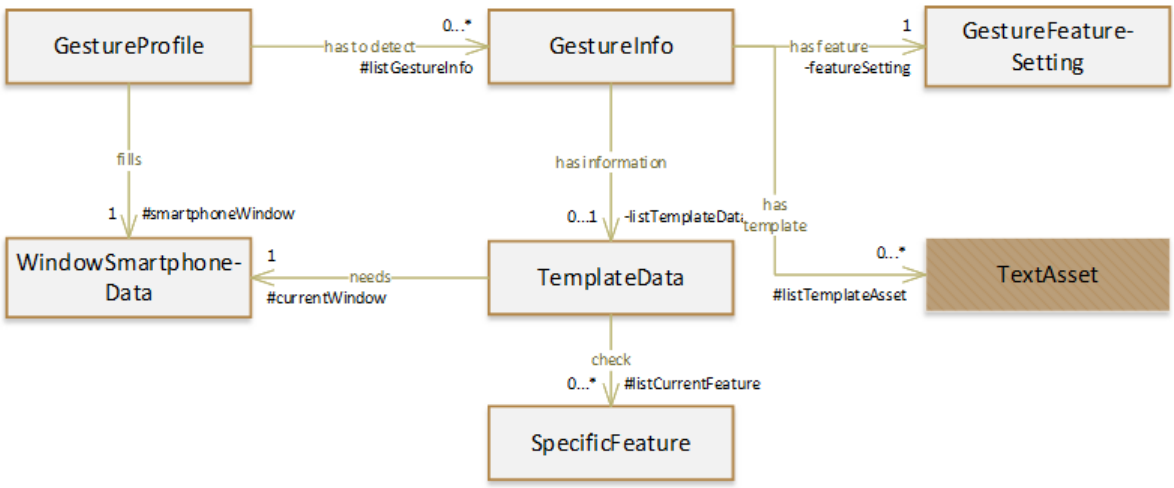


Figure 6.13: Overview of the gesture recognition system

restrictions. To handle the different features, the GestureFeatureSetting is used. It represent of different features, which has been discussed in chapter 4 (in the section "Feature extraction"). Not every feature has the same properties, it depends on the information what it needs. But there exists an intersection for all features, which can be specified in the abstract class "SpecificFeature".

During runtime, the data must go through different processing states. For every step, an own method is offered by the abstract class "SpecificFeature". The advantage for the developer is that he can split up the logic into small parts and is called automatically by the system when the time has come. Table 6.1 shows the abstract methods and how it should be used:

The methods ensure that the developer can specify his own features. He has any restriction and can adapt according to his needs. All pre-defined features work in this way, but sometimes it's necessary to add some new one. The framework offers an interface where the developer can inject own calculations. For this case, the game object where all features are defined has a placeholder to add them. To do that, only one object is needed where his features are attached in form of scripts. Only one condition must be fulfilled, namely the scripts must be a child of the AbstractFeature class. It follows the same call rules shown in table 6.1. Figure 6.14 represents the available features which are offered by the framework and how the relation between the classes is.

Options for the Feature Same sign	Description
Smartphone sensor	Select which sensor data should be used and the type of the option is an enumerator. The possibilities which the developer can select are Acceleration, User-Acceleration, Gravity, Rotation rate, Rotation rate unbiased and Customized1-4.
Enable	The type is a Boolean and can be used to activate or deactivate the feature.
For every Axis enable x, y and z	Activate a certain axis on which the feature should be used.
For ever Axis Must all positive	The data in a certain window must be greater or equal to zero for getting true, otherwise if it's unchecked, the values must be lower or equal to zero.

Table 6.2: List of options for the Feature same sign

Method name	Time called	Usage
+void preCalculation (WindowSmartphoneData)	During the setup of a gesture	To calculate necessary information at the beginning to compare feature properties during runtime (for example minimum/maximum, etc.)
+void addDataFrame (WindowSmartphoneData, bool)	At the beginning of a frame if new data has arrived	To calculate the current features which doesn't need any information about the template or the whole time span
+bool checkState(WindowSmartphoneData currentData, WindowSmartphoneData templateData, SpecificFeature templateFeature)	Near the end of a frame	This method calculates the feature related to the template. It compares the data and thresholds and makes the decision, if it is a gesture or not. It delivers true if gesture was detected, otherwise false.
+SpecificFeature copy()	At the application start	Copy the information from the GestureFeatureSetting and make it available for the specific gesture
+void reset()	After a gesture was detected	If a gesture was detected, the data for the feature will be reset (default values). After this step, the recognition starts from the beginning.

Table 6.1: Compare different gesture detection methods

Options for the DTW calculation	Description
Smartphone sensor	The same selection possibilities like in every feature.
Enable	Activate or deactivate the whole feature.
Automatic	A Boolean which can be checked if the developer wants to determine the thresholds automatically by the system. If it's activated, all other set thresholds will be ignored.
Multiplication	At the start, the min and max value of the template will be determined. This value will be multiplied with a factor for calculating the threshold. The range is 0.0-1.0 for Min and Max, for Min not lower and Max not upper can be choose a value between 1.0-2.0. This will be only done if automatic is activated.
For every Axis enable x, y and z	Activate a certain axis on which the feature should be used.
For ever Axis x Threshold, y Threshold, z Threshold	This option depends on the feature. For min and max, the value must be higher or lower as the value, for the other, the value must be within the band.

Table 6.3: List of options for the Feature min, max, above min and below max

Options for the Feature Min/Max/Below Max (Max not upper)/Above Min (Min not lower)	Description
Smartphone sensor	The same selection possibilities like in every feature.
Enable	Activate or deactivate the whole feature.
Cut-off-value	For a certain data point, the absolute value of the measurement must be greater than the cut-off-value. If this is not the case, the point is set to zero. With this method, the noise in the signal gets removed which can have a negative effect for the calculation.
Distance measurement	The developer can select the method how the distance should be calculated. It is realized as a drop down list. He can choose between Manhattan, Euclidian, Squared-Euclidian and Maximum method.
Boundary constraints start	The start point for the calculation is (0,0) if this feature is activated. That means that the first value of both signals must be used.
Boundary constraints end	The end of the calculation must be (n,m) if it's activated. In the case of the framework the last data point of each signal.
For every Axis enable x, y and z	Activate a certain axis on which the feature should be calculated.
For ever Axis: x Threshold, y Threshold, z Threshold	In this option the developer can define the thresholds for each axis. The way how it works is that the distance measurement of the current signal and template must be lower than the threshold. If this is the case, the system detect them as the same, otherwise it's not the same gesture.

Table 6.4: List of options for the DTW calculation

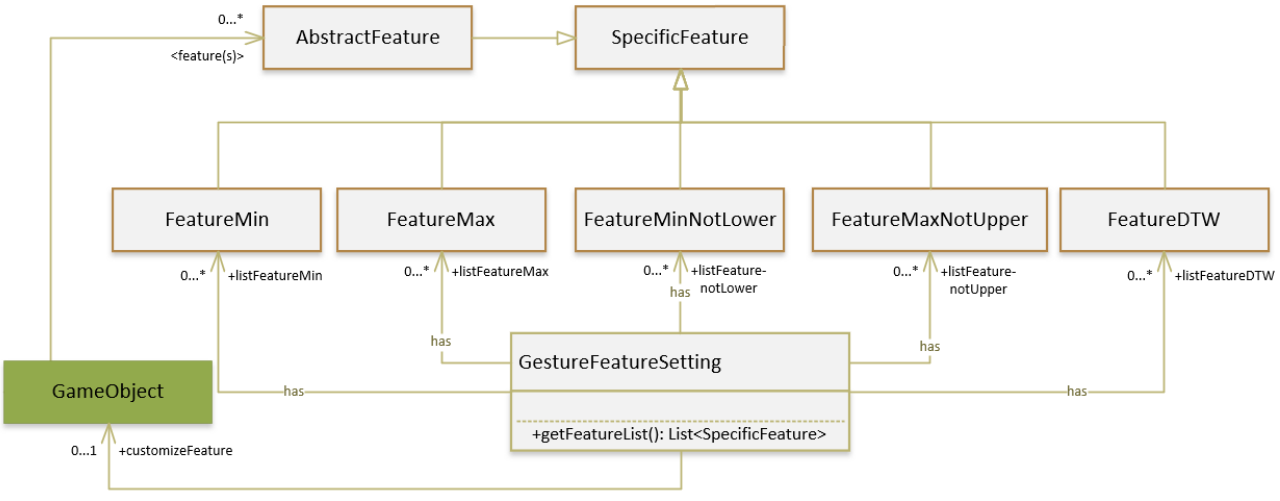


Figure 6.14: Features which are offered by the framework; The green block is the placeholder for new features

6.7 The menu system

The framework solves different problems which can occur during the developing process. One part is responsible to show the user information or a possibility to setup certain parts. In this case it's necessary to tell the client application how it can connect with the server. For that, input fields must be offered in form of a menu where the user can tell the system the important states of the other device(s). A challenge is to design the menu that it changes the look given by the underlying platform. For the developer the effort should be also keep small. At the current state of the framework, it distinguishes between two types of devices. For example, a server must create all communication elements at the start of the application like setting up the stereoscopic rendering. This is needed for the VR headset. Another point is the network communication which must be considered. For the server, the input possibilities are limited. You have no keyboard or other physical device to enter a string, but this is needed to set the IP address and port number. To avoid this, the setting can be done during the developing process and create the related objects also during the application start.

On client side, it is easier to give the information of the opposite device. For Smartphones and PCs the client can tell the application the IP address. For Bluetooth, the UUID can be set during the developing. The UUID is needed to identify the correct services for which application the incoming or out coming data is.

6.7.1 Start menu

The start menu is responsible for setting up the communication. An important fact is to determine, if the program is running on a server or client device. As mentioned before, a server doesn't need any information, because the necessary information like port number, channel number (to assign the data to the correct consumer/producer) or UUID must be defined by the developer during the implementation. For the client, it's necessary to show a menu for the communication setup. A second point which must consider is what possibilities a device offers. What kind of menu the user sees depends also on the running platform. This should be determined automatic by the framework without the developer must define it.

To remember, the communication should support different types of devices. Not everyone has the same possibilities to transfer the data from A to B. A Smartphone is an all-rounder, so both methods can be offered. Also most PCs have a Bluetooth module, but it isn't standard. So the decision for the framework was made that for PC, only networking over IP will be supported.

In Unity, the developer can create an application for different platforms. Therefore, in the player settings the developer can choose the desired type. As well the selection can be asked in the code and used for controlling the logical flow of an application. How the flow looks like, figure 6.15 shows the behaviour.

The actions which are coloured in red represent different menus. It depends on the type of the application and will only render if some conditions are fulfilled. To control the flow, the developer can set in Unity (Player settings) if Virtual Reality support is desired or not. Another option is, if it's checked, which kind of headset it is and loads the necessary library. For the showcase, Google Cardboard is the preferred technology. On the base of this option, the application becomes to a server, otherwise it's a client.

As mentioned in previous sections, the server doesn't need anything. It sets up the communication components immediately after the app starts. Only the type of the communication must be selected by the user. From the perspective of the client, it must know information about the server. If the program run on a Smartphone, the user see a menu shown in 6.16. The first step is to select the desired communication. If IP over network is wanted, the user gets an interface where he can set the address (figure 6.16). After the confirmation (click the start button), the client application stars. In the case of Bluetooth, the application immediately starts, but during the loading process, all components will setup with the information set in the developing (UUID). Note before clicking the button, the Bluetooth devices must be paired, otherwise the connection can't be created.

6.7.2 Gesture detection menu

The gesture menu is splitted up into two parts and represents the client application of the showcase, but it can also use for other projects. One part shows the user how often a gesture was carried out. For that a list exists with all initialized gestures. The initializing process is done at the scene start and includes all information which was set by the developer. For every gesture an own counter is shown, and it only increase if the menu is active. The second user

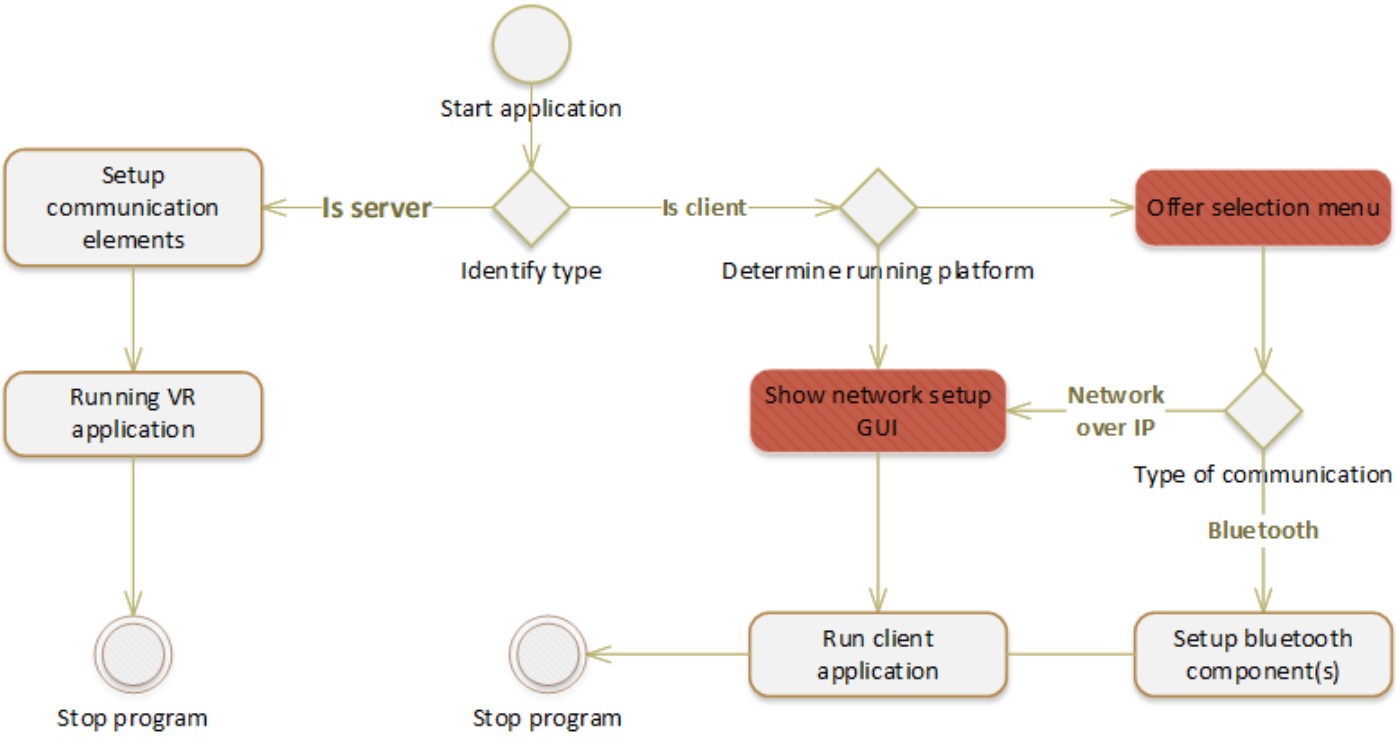


Figure 6.15: Concept of the communication menu

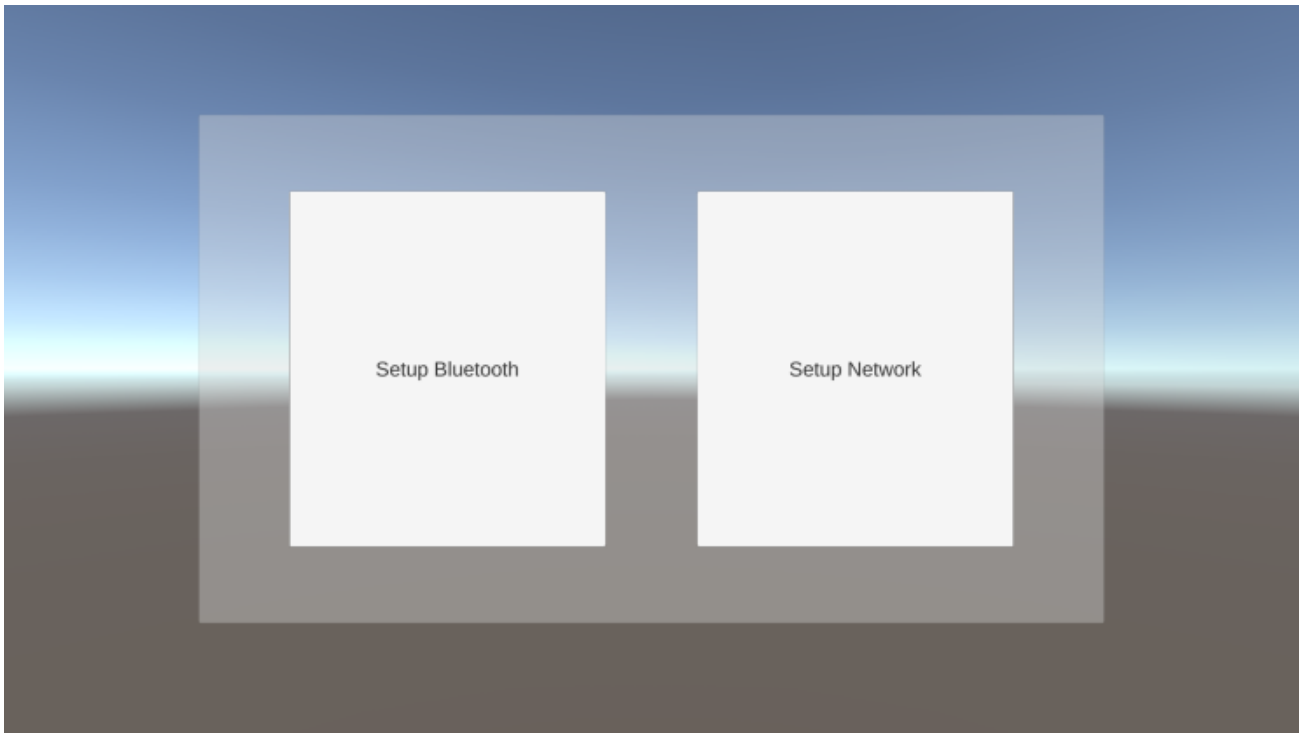


Figure 6.16: Start menu if the application is a client

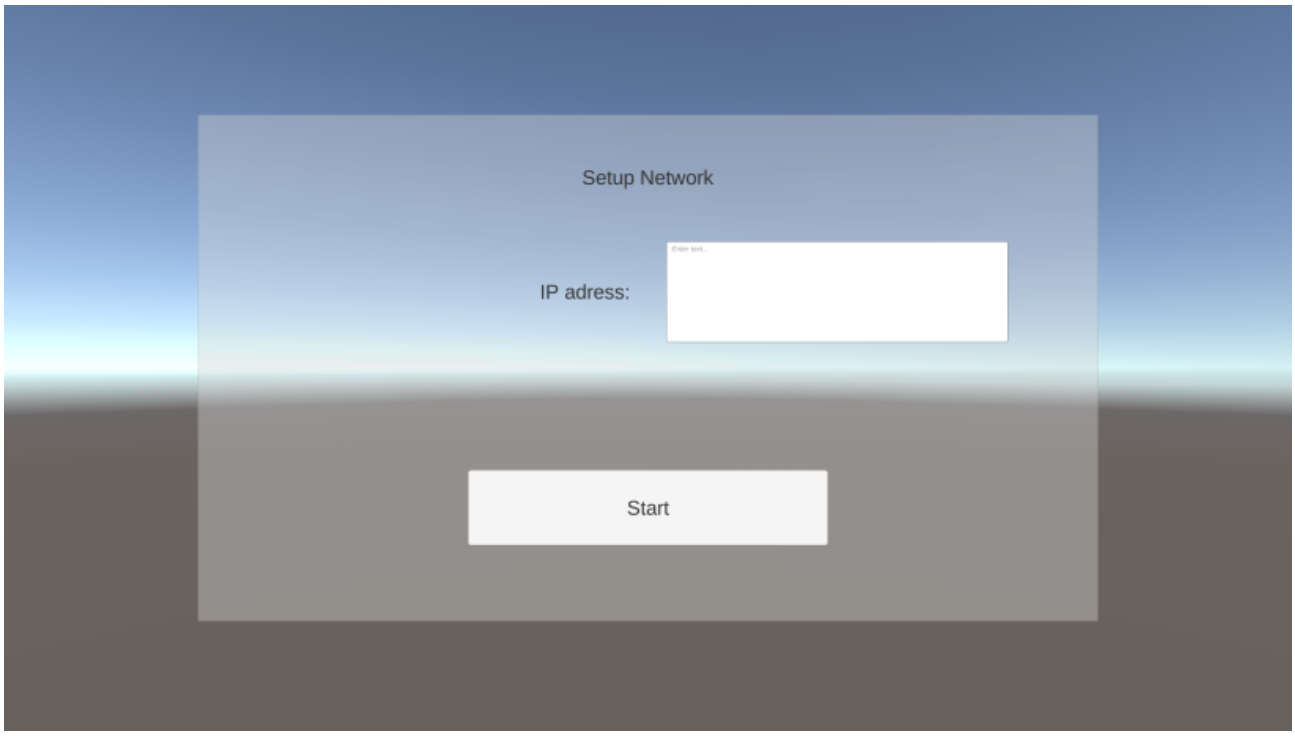


Figure 6.17: Interface to set the IP address

interface solves the task to save the data to a file. Elements are an input field, where the file name can be specified, and a button, to start and save the data. One characteristic of the button is that it changes always his functionality if the user clicks it. At the beginning, nothing is happen until the user starts the recording. The signal gives the system a sign to collect the data and hold them in the Window class. This can be used for the analysing steps to create a template for the end application. During the data collection, the state of the button is changing. After the gesture is recorded, the user can click it again to save the data.

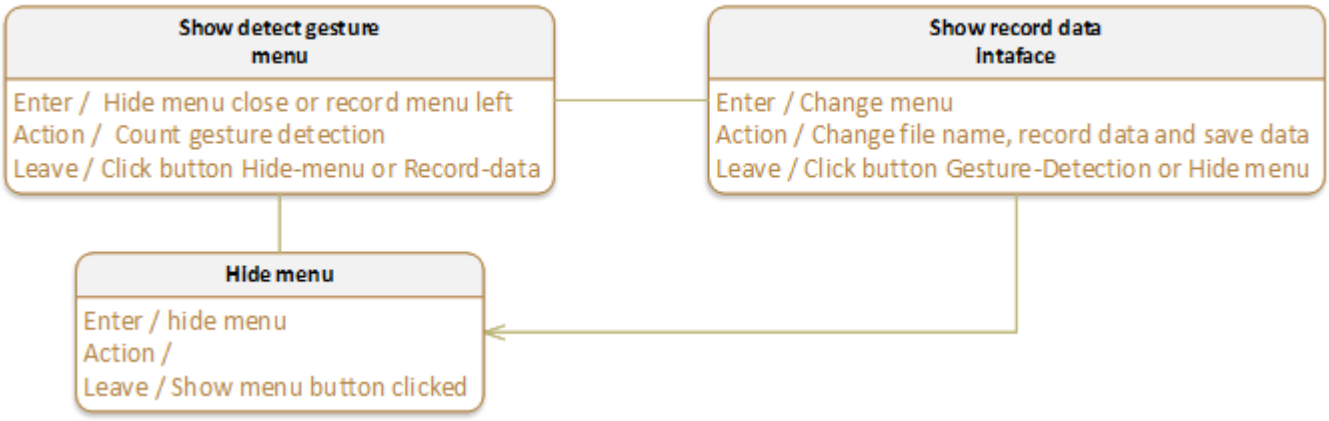


Figure 6.18: States of the menu

Sometime the menu is a handicap, for example if touchscreen input is supported. The user can activate a button or other elements, but it wasn't his intention. To avoid them, every menu has a button to hide them. But also a button is necessary to show it again. It's placed in the middle of the screen and is kept small.

Figure 6.19 will make the things clearer of the available gesture detection menu which are offered by the framework. The images are made during the evaluation of the showcase applications. All defined gestures are shown in the

list with names and related counters at the right side.

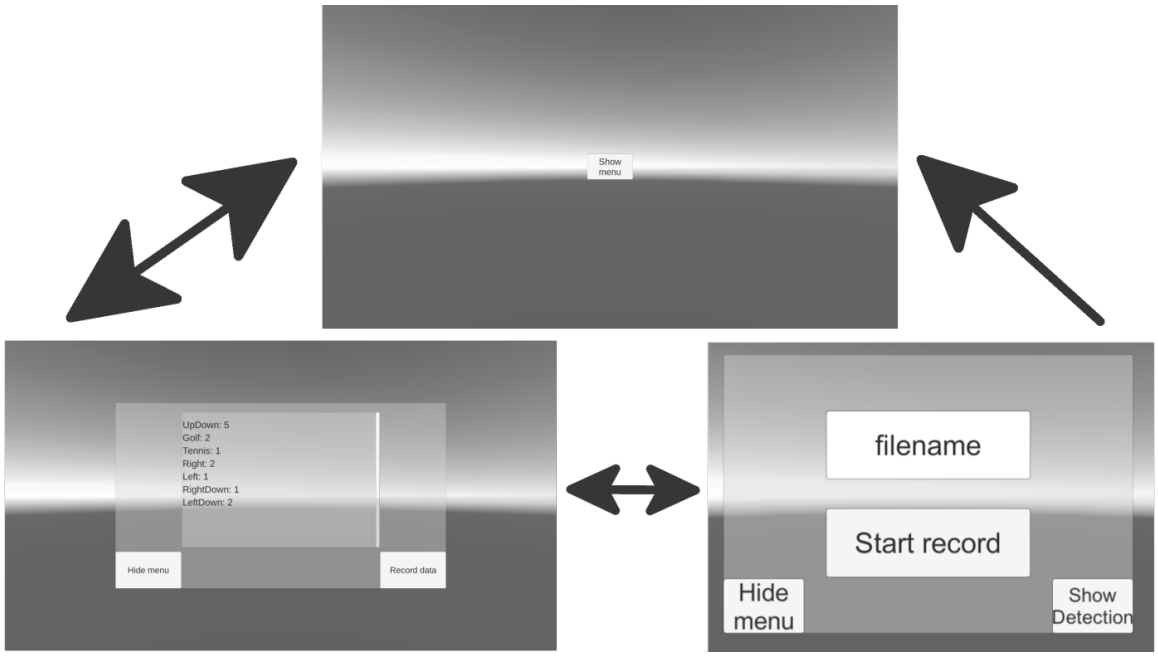


Figure 6.19: Sequence of changing the menu - Upper image text: Start menu, bottom left: some gesture - see later

Chapter 7

Prototype implementation - Showcase

The showcase has the task to test the framework, included are the usage and the functionalities. For that an application for the server must be programmed where it's possible to interact with the VR. It should contain different kind of interactions with one or more devices. In the case of the showcase, the tested devices amount to Android Smartphones and Microsoft Kinect version 2. A concept must be defined as clear as possible to show the different interactions in the VR, like moving in the world or activate some special features with gestures. The showcase for this project isn't only for building a test application, but also for analysing how good the functionalities are working. Important for the developer is how good the recognition system works and how he can extend the framework with new devices or gestures.

The following part of this section describes how the showcase works and which idea stands behind them. As well it explains how the network and gesture setting will work and which values has been chosen for thresholds. At the end, the whole project will be evaluated for each device.

7.1 What the framework includes

The first step is to collect some ideas to define the VR world and the possibilities to interact and manipulate them. As before mentioned, two devices for generating input has been choose, namely an Android Smartphone and the Microsoft Kinect v2. The aim is to use as much sensors as it's possible to look what kind of interaction for which devices is possible and how good they are working. Also the computer human aspect should not be ignored how user friendly it can be realized. A question is, how intuitive is the input and how the feeling is over a long time. To handle all points, at least two independent projects are planned where the functionality of the devices will be used. Because of a bug of the used Unity version, the client and server are separate application at the end. There are two apk's. The expenditure during the developing is small, because it must only activate the VR support for server, otherwise a client app will be created.

7.2 The idea

One idea for the Smartphone part is to show the functionality of the framework with the help of a theme park. In the park, there exist different attractions. Every attraction has only a goal, namely to show a certain test scenario. For each scenario, different sensor information is needed, for example the orientation of the device, touch screen events or gestures to interact with the elements in the unreal world. Also a solution for walking through the world from one attraction to another must be found.

For the Kinect, a simple test application will show how the device can be used for interaction and manipulation. To see what the avatar is currently do, a skeleton of the Kinect user should be detected and visualized. Elements of different Joints are simple meshes connected with lines. Like in the case of Smartphones, a solution for walking must be found. The test application provide for further work more advanced interaction possibilities like catch objects and threw it through the world.

At the end a short discussion can be found which explains how a new device can be injected into the existing construct or new data can be used.

7.3 The world of the Smartphone

A Smartphone is really an all-rounder, so the showcase for this device must consider different input methods and sensors. To show the functionalities, three attractions are planned, namely:

- Shooting Gallery
- A attraction called Circle-Spin
- (Mini-)Golf area

The following part presents an explanation about the possible interaction for each attraction, how the user can move and some information about the implemented gestures. At first, the shooting gallery handles different type of information. It needs a touchscreen for handling the arrow and the gravity sensor. With it, the rotation angles are calculated for setting the shot direction. Also the circle spin needs the orientation data, but the idea behind this attraction is to test the Smartphone whether it's suitable for simulate a car application. For further work the device should be used as a steering wheel. These two attractions cover the sensor data and the corresponding calculations. To test gestures, the last one is used. For a golf application, interactions with a ball and a golf club are needed.

Furthermore there are implemented more than the necessary gestures which are needed. A reason for this decision is to have a look about the efficient of the system. One requirement is that all must be happen in real-time. Another point which has a high significant is if the recognition is stable enough. A system is stable if there are not too much false detections (false positive is a big problem).

One question which is open, how the projects has been realized. All objects, graphics and logic were implemented without any use of third party frameworks, libraries or models. The models in the world were created with the open software Blender and for texture or graphic manipulation, Gimp was used. One think which is important is that the models should be as simple as possible to reduce the power which is needed to render the scene. In other words low polynomial objects and small texture size. As discussed in preview sections, a Smartphone is restricted in computing power and this point helps the developer to have better performance for the VR app.

7.3.1 Movement

An important problem which must be solved is how it's possible to move in the VR reality. With every headset the user can look around the world with no restriction, in other words he has a 360 view. With the Smartphone, the touchscreen seems to be the best input possibility to solve the problem of walking. The user can carry out touch gestures to start or stop the movement. Also more precise input can be done. At the beginning, a type is determined how it should be realized. Types of walking can be distinguished between normal (slow) walking and (fast) running. To tell the avatar in the VR which kind of movement the user wants is how the gesture is initiated. He can do a single or double touch before carrying out the gesture. This mode isn't restricted only for the movement. It can be used for every kind of input or gestures made by the touchscreen. With this feature, the developer has more possibilities and fine-tuning options for near identically interactions. The two mods for the type of movements can be determined by the enumerator TouchGestureMode. Following states are allowed:

- Slow Single touch on the screen
- Fast Double touch on the screen

As mentioned, with the touchscreen the user can carry out some gestures. There are four possibilities implemented which are very simple and self-explanatory for the user. Like to determine the type, the gestures are handled by an enumerator called TouchGesture and have following states:

- No No gesture was detected

- Up The touch was made in the up direction (positive y-Axis)
- Right The user swiped his finger in positive x-Axis
- Down Negative y-Axis has become the focus of the recognition
- Left The opposite direction of the right gesture

As in all gesture detection algorithms, a threshold is necessary to determine if the error is small enough. In this case it is the gesture, otherwise it's nothing. For touchscreen, the threshold can be specified in various dimensions. At first, and the simplest method is, to define how many pixels must be involved from start- to the end-point of a gesture. This method has a big disadvantage. Not every Smartphone has the same resolution and/or pixel density. The distance in relation to the screen dimension can be significant different. The possibility exists that for one Smartphone, the distance is pretty small, for others it's not enough (too little resolution).

For solving a problem like that, the developer can use relative values/percentages for thresholds. The threshold can be in the range between 0 and 1. For example, a value of 0.2 means 20 percent of the pixel of an axis must be involved. The calculation isn't difficult and can be done in that way:

$$ScreenMoveX = (1.0/Screen.Width) * (endPosition.xstartPosition.x) \tag{7.1}$$

$$ScreenMoveY = (1.0/Screen.Height) * (endPosition.ystartPosition.y) \tag{7.2}$$

$$if(threshold < ScreenMove) => gesturedetected \tag{7.3}$$

As mentioned before, the framework can distinguish between 5 states. The "no" state doesn't do anything, the avatar in the world stands only still on his place. With the up touch gesture, it's moving forward and by tipping the display again, he stops moving. The same applies to the down gesture, only that the avatar moves backwards. Also the mode is used and handles the speed in which the user is moving in the world.

It's not the only problem, how the user can move in the world. Also it must be found a solution, to enter and leave diverse attraction. In the front of a mini game, the player can find a crosshair. If he is near of them, it begins to rotate and a particle system starts his animation. This effect should visualize the user that he can carry out now the touch gestures to enter (right touch gesture) and leave (left touch gesture) an attraction. For that, no more accurate specification has been done. It makes the same work in different modes. The distance can be also set with the associated game object. The value specifies a radius where the player must be in.

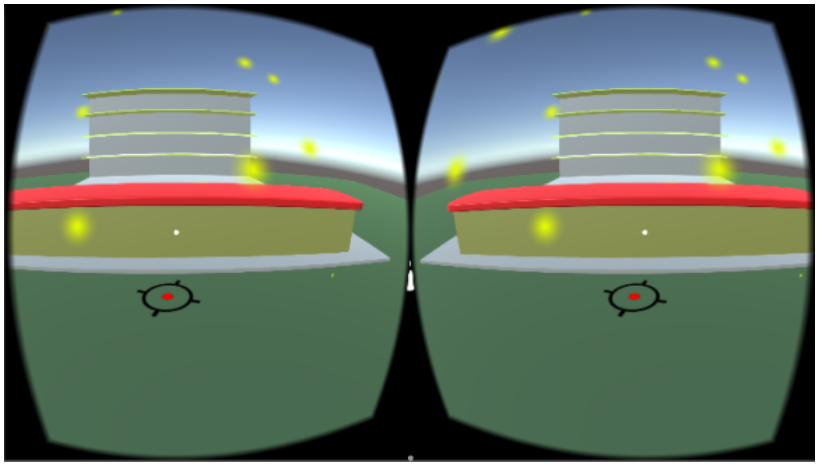


Figure 7.1: The crosshair rotates and a particle system will be activated when the player is close enough to the attraction (in this case the shooting gallery)

Now he is able to interact with it. During the interaction, the player isn't able to move in the world anymore. If the user decides to leave the attraction, in the most case he must look back. The left touch gesture, which is responsible

for leaving it, is at the beginning deactivate. By looking back, the point which determines the look direction becomes to a circle (figure 7.2). This indicates that it's possible to carry out the gesture. By leaving an attraction, all state will be rested and it's possible again to move in the world.

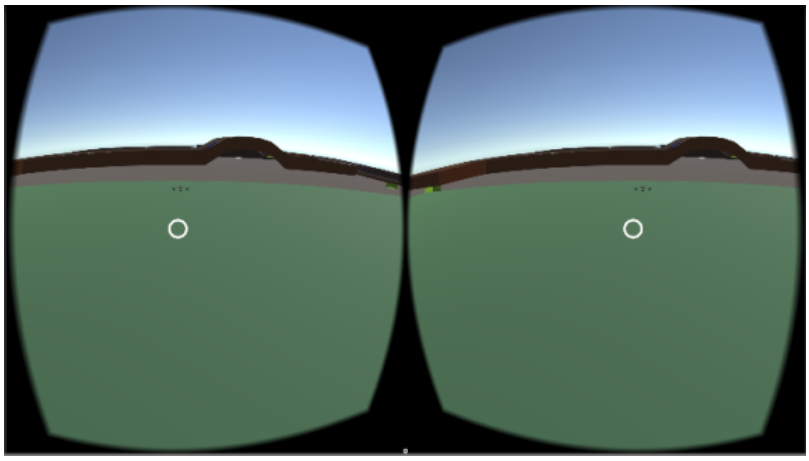


Figure 7.2: A circle textures in the look direction visualize the user that he can leave the attraction

7.3.2 The shooting gallery

The shooting gallery has the aim to show how the orientation of a Smartphone can be used to manipulate the VR. This can be done in two ways. At first, the gyroscope delivers the necessary information about the orientation of the Smartphone. Unity calculates with the help of rotation velocity and acceleration the current angle of each axis. A problem which occurs in this method is that the axis of the device doesn't match with the scene axis. The developer must find a transformation which solves this problem. The direction of the axes are not only the problem, also the direction of the rotation around the x axis isn't the same.

A second method, and which is used for this project, is to calculate the angular with the help of the gravity sensor. How the calculation has been done can be found in the previous section (). The idea is to handle the arrow direction with physical sensors to aim a target object on a shelf. To move the arrow or reset its position, the touchscreen sensor is responsible to address these problems.The figure 7.3 shows the design of the Shooting Gallery.

The shooting gallery has following elements:

- A ground - To visualize that all elements on it belongs to the parent object
- Arrow - The white cuboid is the object which will be shot. The direction is determined by the forward vector. This is the vector from the top to the end
- Table - The table is a boundary two have an orientation where the user is (know that he must stand here; Note that if the player enters an attraction, he can move anymore
- Shelf - The shelf is the place, where the objects, which can be shot down, are placed. It has four rows with free degrees in the x direction.
- Object – The object has the shape of a simple balloon. This balloon can be shot down by the user. If the user hits them, it will be randomly placed on the shelf.
- Two crosshair - One handles the entering of the shooting gallery (lays in front of the attraction), the other one visualize the user where the shot will hit (see figure 7.4).
- Two colliders (invisible for the user) – If the arrow miss the object and flow out of the area, it will be destroyed. The other one handles the leaving of the shooting gallery. Looking back in the opposite direction of the shelf, the touch gesture gets activated as descripted before.

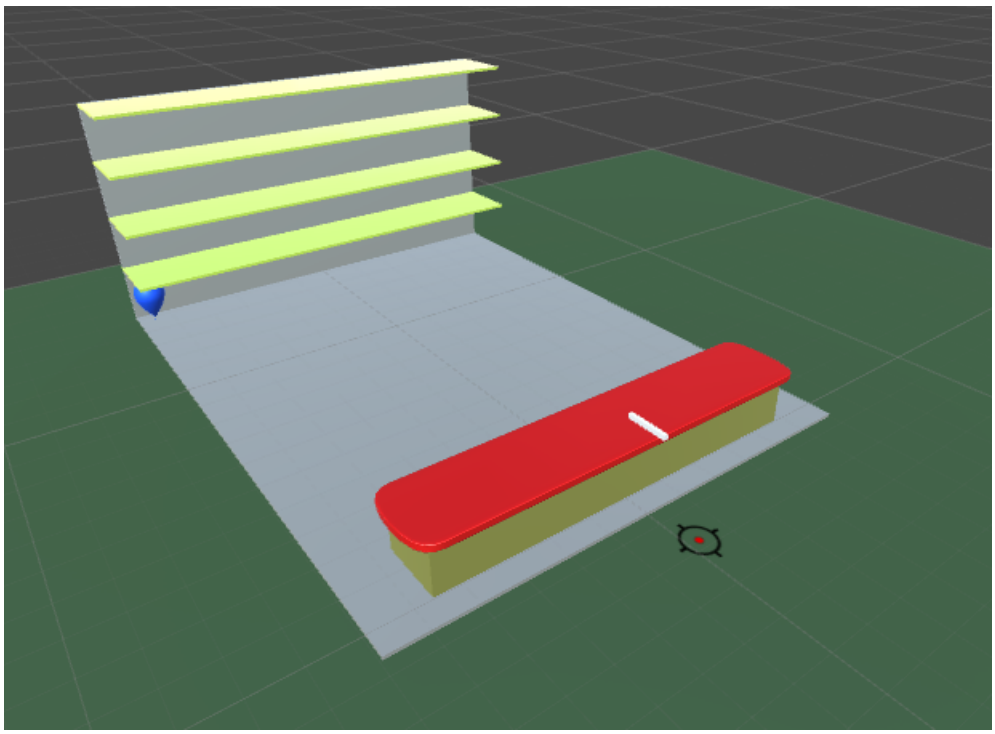


Figure 7.3: Overview Shooting Gallery

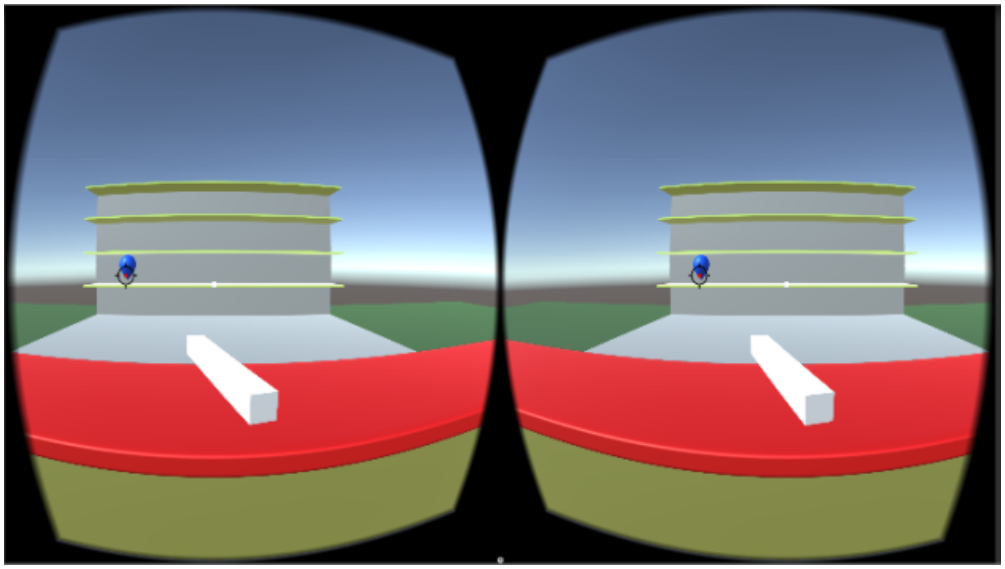


Figure 7.4: Showing the Shooting Gallery in the VR mode

Following elements are used to realize the interaction:

- Simple touch - If the user touches the screen, the arrow flows in the direction of the crosshair.
- Touch down gesture - To reset the arrow to the start position. It also set the velocity and acceleration to zero.
- Smartphone x-Axis - Calculated with the help of the gravity sensor and manipulates the arrow in the y direction (up and down).
- Smartphone y-Axis - To manipulate the y-Rotation (left to the right and vice versa), also calculated with the gravity vector.
- Touch right gesture - To enter the Shooting Gallery.
- Touch left gesture - To leave the area and start the normal movement again.

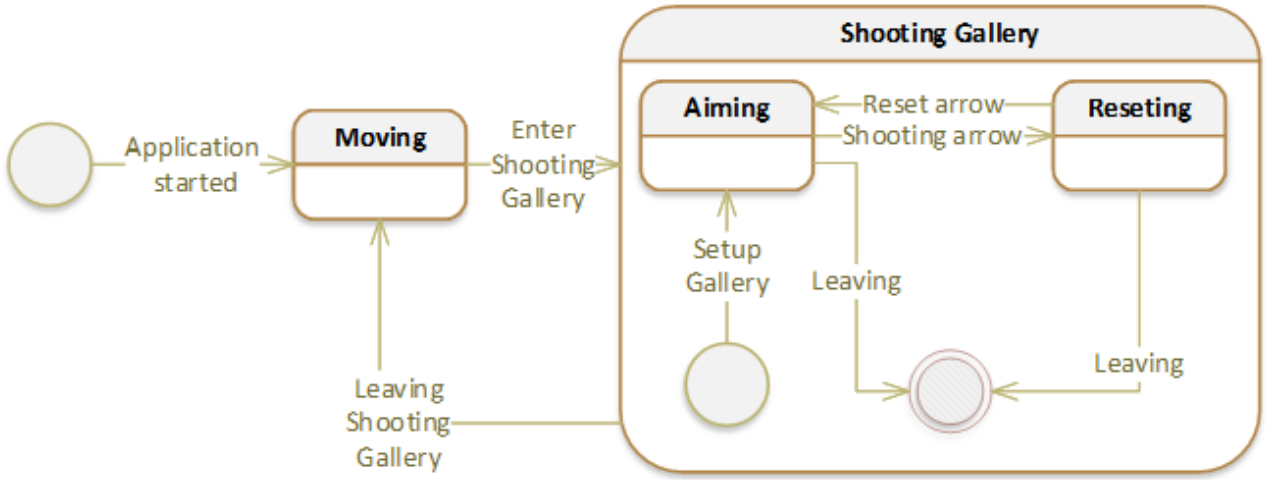


Figure 7.5: States with regard to the Shooting Gallery

7.3.3 The Circle Spin

The second attraction realized in the VR is called Circle Spin and doesn't introduce new sensors. It also uses the gravity sensor to calculate the rotation angle of the Smartphone device. In the Shooting Gallery, it has been used to align the arrow to the target object. For this attraction, the aim is to simulate a car wheel. One possibility in the future can be a simulation, where the user drives a car in a VR application. By rotating the Smartphone around the z axis (the one which goes out from the display), an object react to this changes and makes the same rotation like the user has done it with the device. To test the behaviour, a labyrinth in form of a circle has been created. In it, a white ball is placed in the middle. The aim is to bring the ball from the middle to the outermost ring. For that, the user interacts with the labyrinth by rotation the Smartphone. If the way is free, the ball falls down to the next level.

The ball can be only reset to the start position by leaving the Circle Spin. To do that, it works like the same way as by the Shooting Gallery. The user must Looking at the opposite direction of the wall and carry out the touch gesture "leave". If he decides to enter it later, the ball is in the middle again.

The Circle Spin has following elements:

- Background - To make it easier to show the labyrinth and the ball inside.
- The "Circle" labyrinth There exist only one way from the middle to the outside. By rotation the device, the labyrinth rotates too with the same angle given by the device.
- A ball - The ball will not be directly affected by the user. If it's possible, the ball goes down to the next level. A gravity force tries to pull the ball down, if no obstacle blocks the way.

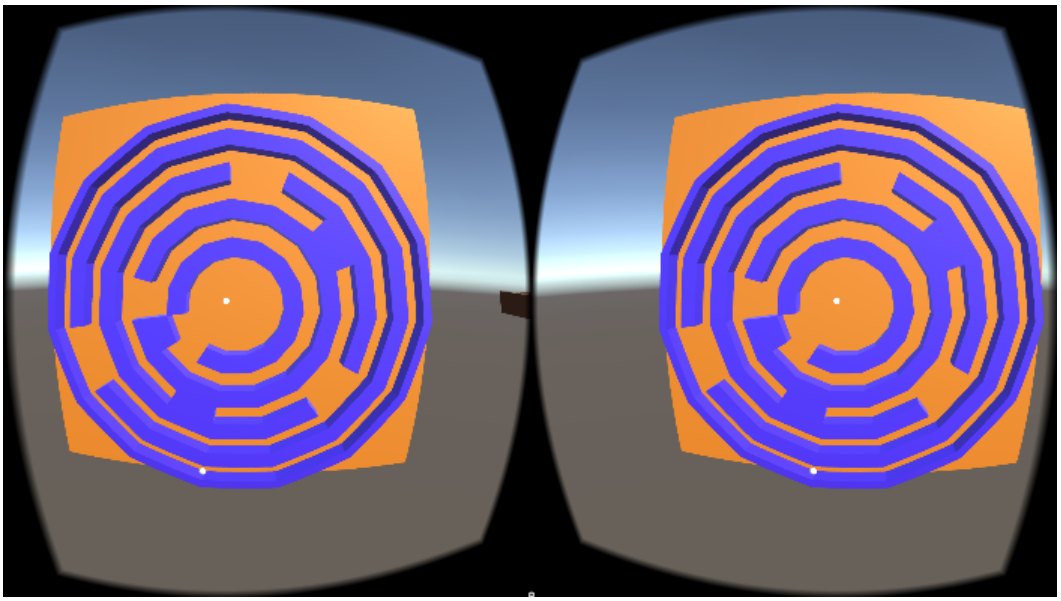


Figure 7.6: Circle Spin concept - ball has reached the outside

- Crosshair - To visualize if the user can enter the attraction.
- Invisible collider - It is placed in the opposite direction of the background wall. If the user looks to it, he can leave this area by carrying out a certain touch gesture.

Interaction realisation for this part:

- Smartphone z-Axis - The angle which has the device will be translated to the same rotation angel for the circle labyrinth.
- Touch right gesture - To enter the Shooting Gallery.
- Touch left gesture - To leave the area and start the normal movement again.

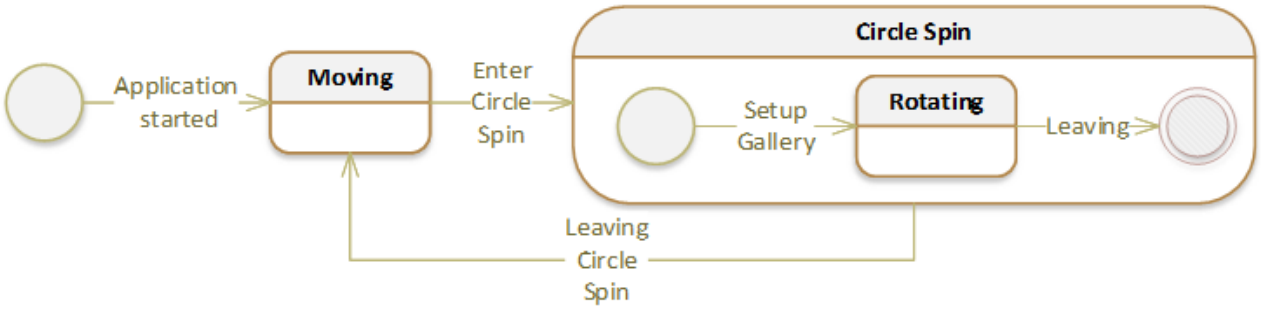


Figure 7.7: States of the Circle Spin attraction

7.3.4 (Mini-)Golf

The golf area has the aim to test the functionalities of the gesture recognition system. To do this, some templates have been recorded, prepared and features set. In the world, there exist three different layouts, where the golf ball must reach the hole. A challenge is to find a way to set the power of the shot and the direction. Other things which

are important are the lever handler. This object is responsible for set the next level if the ball has reached the aim. Another task is to reset the ball to the last position if the ball flows out of the area. The camera follows the ball always in a certain distance to it and can rotate around the ball. This depends on the direction in which the user is looking with his headset (Google Cardboard for example).

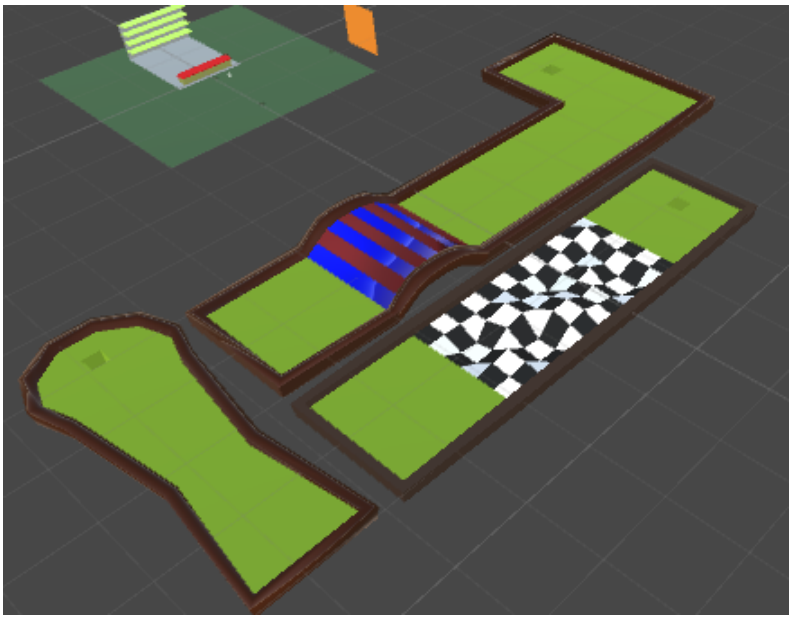


Figure 7.8: Overview of the (Mini-) Golf area

Elements of the golf area are:

- Three levels - To test the functionality, three levels are modelled. Each level has different obstacle which must be overcome to get to the next stage. If the last one has finished, the user gets to the entry point of this attraction.
- Arrow - To show the direction depending on the headset orientation.
- Text - To show the current power this will have the golf club.
- Golf ball - The object which must reach the goal.
- Golf club - A cuboid which hits the ball by doing a certain gesture.
- Invisible colliders Are necessary to leave the golf area. These are located on the opposite direction of the level up in the sky.

For interaction, theses device data are used:

- Look direction of the headset - The headset can set the direction in which the ball should be shot.
- UpDown gesture - To change the state, if the direction and/or power was set.
- Golf gesture - Emulate a golf movement with the phone. If the gesture was recognized correctly, the ball would be hit by the golf club.
- Touch up gesture - The level of power will increase and visualized on the screen.
- Touch down gesture - Decreasing the power of the golf club.
- Touch right gesture - To enter the Shooting Gallery.

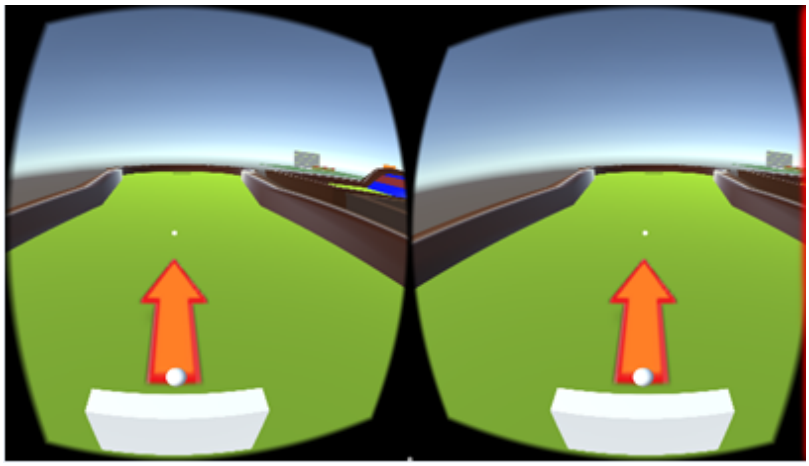


Figure 7.9: How it looks in the headset: The phases where the direction will be set

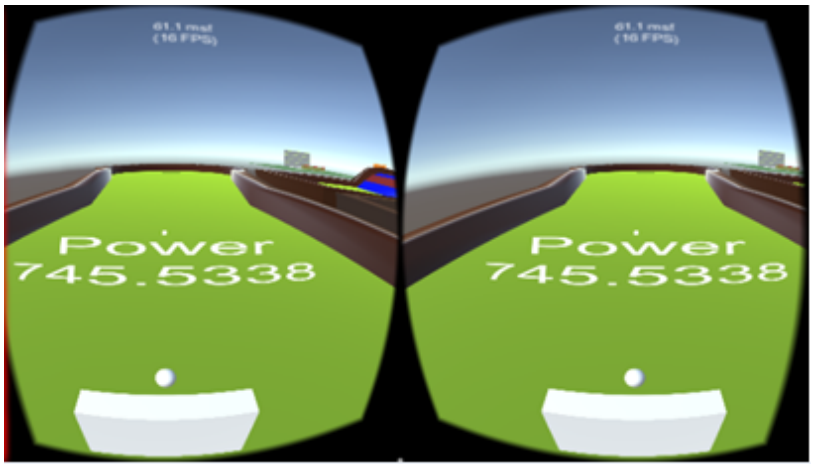


Figure 7.10: Set and visualize the power

- Touch left gesture - To leave the area and start the normal movement again.

Figure 7.11 represents the different states which can occur during the golf interaction. It has a certain flow and begins with the setting of the direction. As mentioned before, with the help of the UpDown gesture, the user can get to the next state. In this case, the power setting part becomes the focus where the strength of the hit will be determined. Now it's time to carry out the golf gesture and with little luck and can, the ball falls into the hole and the user gets to the next level or it will be finished. An important property is that it's always possible and every state to leave the golf area.

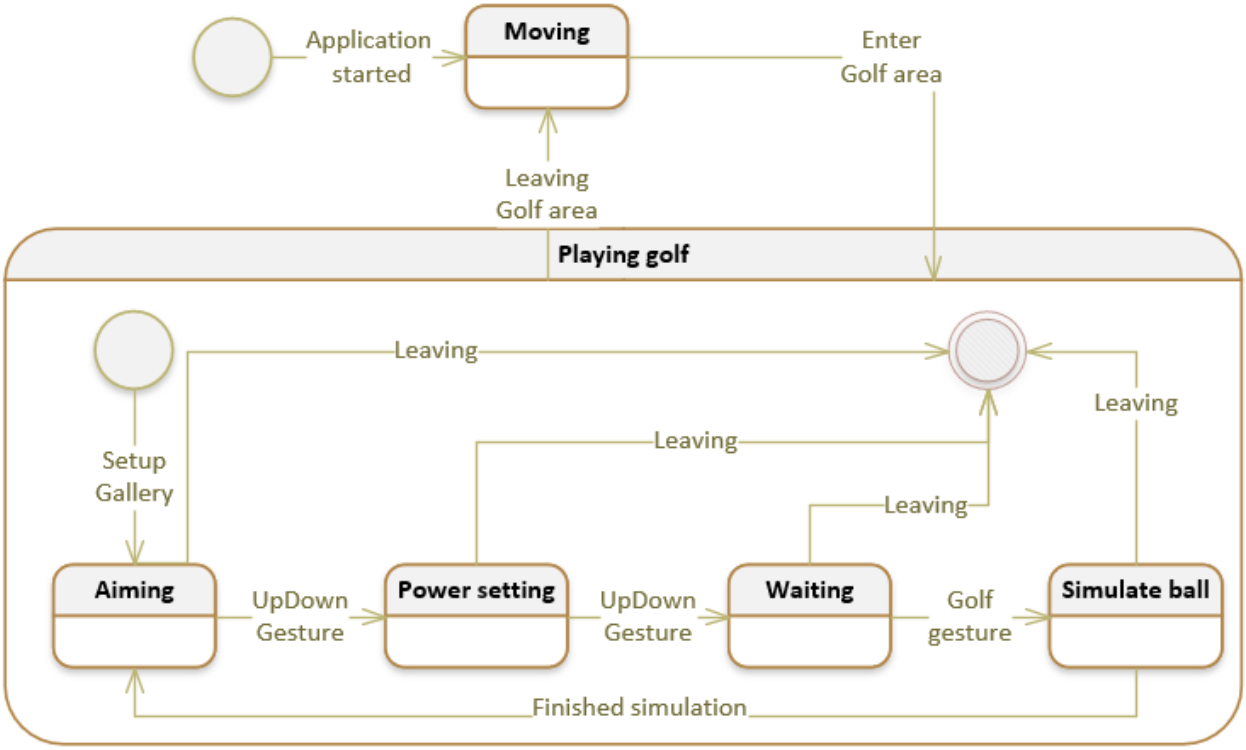


Figure 7.11: The state chart of the golf interaction and how to change it (which gesture it's necessary)

7.3.5 Implemented gestures

To do something in the VR-world, the showcases need some gesture input to interact with the VR. The following part of these chapters will explain which gestures have been implemented and the related settings. The default framework has realized more as it was necessary for the showcases. The developer have access to all of these gesture (settings, template) and can alter it as he needs. A reason for this decision was to evaluate the recognition system on two facets. The first question is if it's efficient enough to handle more than one gesture at the same time. Another point, and the most important, is how often a gesture was correctly recognized. To become an overview of the implemented gestures, the following list shows the names of them (name is important for accessing of the InputData concept):

- UpDown
- Left
- LeftDown
- Right
- RightDown

- Tennis
- Golf

These gestures build the basis for the analysis in the result chapter 8. The recorded data of every gesture and which setting has been done can be also found in the Appendix C. It represents the data which has been used to determine the thresholds for the feature settings. To do this, an important question is which sensors are useful and how the value range should be choose. During the implementation, following rules has been worked up:

- Which form has the data? Includes it certain patterns which are always occurring in the data?
  - To check if two signals are the same, one requirement is that the signal has the same sequence of peaks, sign changes and have near the same value. It doesn't matter if the current data is a little bit slower or faster as the template. With DTW, it can be compensated a little bit.
- What is the range which the sensor can detect?
  - The high of a value depends on the physical behaviour of the measured data, so the sensor covers a certain area which is possible. For values which are strict limited (has a small range of values), the min and max feature can run into some problems. The reason is how exact the sensor works, everyone has certain accuracy and in small values, the mistakes are bigger. For data which a high range, min and max should be used.
- Are all values small or are there high peaks in the data?
  - If all values are near the same, it is recommended do not allow bigger values like that. For this case, min not lower and max not upper can be used. Otherwise if peaks occur in the data, min and max are suggested.
- Does the sign of the data changes?
  - One of the simplest questions in this area is to look if all data point has the same sign. If this is the case, the same sign feature is the best one. The special case zero doesn't affect the result.

You can find all settings and features in Appendix C. Also the raw data of each record gesture can be found there. With the help of this visualization, the type of sensors has been selected and determine which thresholds are suitable to have a reliable detection rate.

A feature of the framework is, that the thresholds can be set automatic by the system. It depends on the values of the template which will be multiplied with a certain factor. The developer can select the factor in the settings. For the showcase, all thresholds have been set manual, because to this time, the feature wasn't fully implemented and tested.

7.4 The world of the Microsoft Kinect

With the Kinect it's possible to generate more precise input data and has lots of potential for more realistic interactions. You can do many things which are also possible with Smartphones like gesture detection, but one of the greatest strengths is the body detection. With this option, it's possible to translate the user skeleton into the VR and interact with the world as in real.

The only thing is that the effort to create a functioning application is higher than by Smartphones. It exists two parts which must be considered, the programming part and the environment setup. The first different is, that the user doesn't need to hold a device in his hands. The Kinect must be placed anywhere in the room where it can record the environment with its different sensors. A requirement is to have enough space to make full use of the potential. On the programming side, a problem occurs what has to do with the property of a VR scene. A question is how to look around in the world and a concept must be found to realize a moving character.

Finding a way to walk through a room is the aim of the showcase for the Microsoft Kinect. Another scenario which should be covered is to draw a simple skeleton which represents the body data, collected by the Kinect. In future work, this skeleton can be used to catch object and throw it anywhere through the VR. As well, other interaction like shake hands with other persons or control a machine (Wheel, buttons) can be realized with this data.

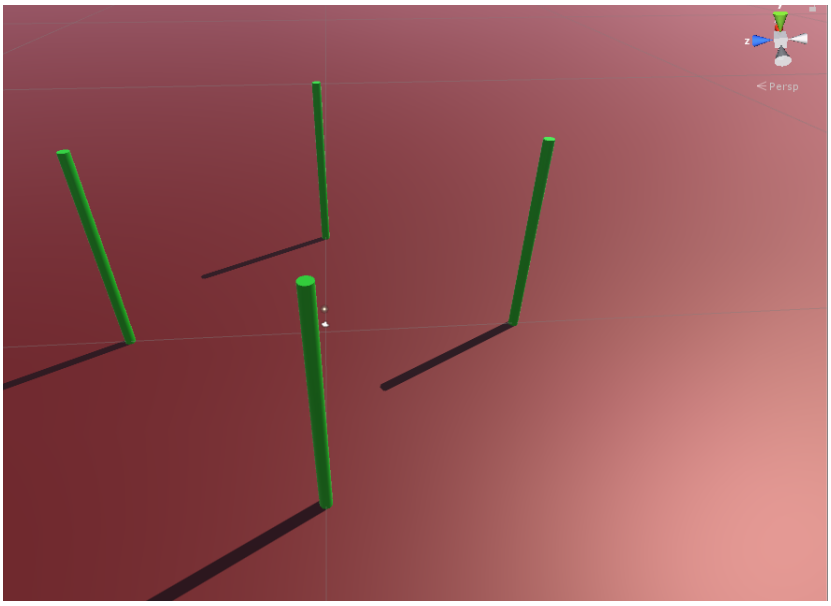


Figure 7.12: Simple test scene for the Microsoft Kinect (in the middle will be placed the avatar/skeleton)

7.4.1 Movement

Movement is a necessary thing for many interactions in the VR, if the user should be able to explore every stone and corner in the world. A VR headset works a lot with looking around without any restriction of angels, as long as no specification was set by the designer. The 360 view is the reason why the problem occurs much more as by Smartphones. The Kinect devices can handle only a certain angle within the user must stand. For example, if the user turns around and the back will be scanned, the data will be false. The left side becomes the right in the world and vice versa. Or if the side is detected, not all Joints can be determined by the Kinect and the behaviour is not the same as expected by the developer.

Moving in the VR is a complex problem which includes different states and it's necessary to keep it as simple as possible. For the Microsoft Kinect, moving doesn't only meant walking through the virtual world, also a method must be create for rotating the camera. The challenge is that it should be done without moving the head. Implementing gestures can help to solve this task. It's designed in a way that for the moving part, there exits two different gestures. Both will be performed with the hands. These gestures must be robust enough to avoid errors. Errors can be if natural movements activate the functionality, but were not intended at this time.

To trigger the beginning, both hands must be over the shoulders and closed. To determine if the user intends to rotate or walking, the position of the hands are important. The reference points in this casse are the elbows. If the closed hands are closer as the elbows to the head, the rotating phase will be activate, otherwise the user can walk through the VR world. To see the effect of the rotation and moving, a simple scene has been created where the user starts in the middle of the scene, surrounded by four pillars (figure 7.12. These pillars are placed in every cardinal direction. Now if the user makes some action, he can recognize them easy.

If rotation is activated, the user can put one hand away from the body (the elbow is the point which is important), to rotate the camera. The right hand rotates the camera in the right direction, the left in left direction (right handed coordinate system). It's getting faster if the distance between hand and elbow gets larger. In the case of both hands are away from the body, the rotation velocity can become zero.

To walk in the world, the gesture for activating the movement must be active. If it was done by the user, he can push on hand in front of the body. The z axis (the direction in which the camera is looking) is important for the speed. If the right hand is further away than the left hand, the virtual body is moving back, otherwise he walks straight forward. For both, rotating and walking, after activating, the hands can be anywhere and they must not be over the shoulder anymore. To stop moving, the user can open one hand and it will end instantly.

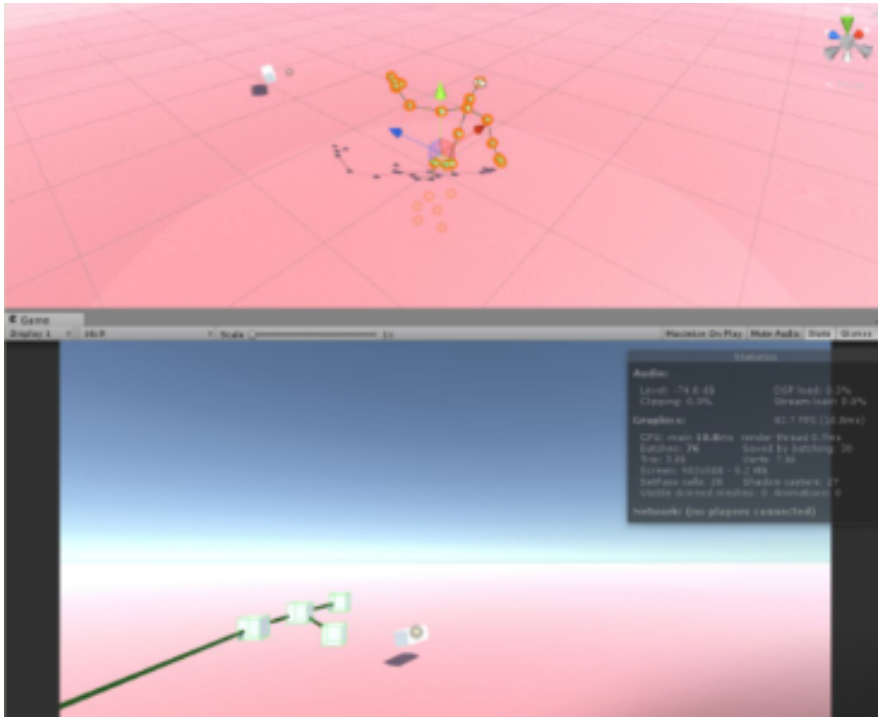


Figure 7.13: First example of the skeleton

The last part of the showcase is to draw a skeleton. The body Joints will be visualized as simple cubes and the goal is to transfer the data from the PC, where the Kinect is connected, to the VR headset device. This data will be used to set the position of all elements. The developer has the possibility to specify which part of the body should be used. There exist three parts:

- Main Body: This part includes the data of the head, neck and body center data.
- Arm data: Left and right arm are part of the arm data. Also a variable for checking the hand state is available (hand closed or open for both sides).
- Leg data: Data for the legs will be drawn if this part is checked.s

If a part is activated, it will be drawn in the VR. Also al necessary elements for interaction will be instantiated at the start which can be useful for further work. For example colliders to determine if the hands are near an object. If this is the case, it could be possible to catch them by closing the hand.

Chapter 8

Discussion and results

This section analyses the usage of the framework. With handling is meant to test the gesture detection system and how the developer can add new devices or sensor data to an existing implementation. To show this part, the Smartphone device will be extended and a new, abstract device will be introduced. This can be everything as long as it's possible to implement them in Unity or C#.

Another part is a short discussion how the realized interaction works, where the limits are and what can be possible for the future. The basis for the discussion builds the showcases from the last chapter (7).

8.1 Results - Gesture Detection

One large part of this project was to implement a system which recognizes reliable different gestures. Reliable is important because in some cases, there isn't allowed for wrong input. An example can be any application, where fast reaction is needed. An incorrect input can cause undesired behaviour or leads to frustration at the user. For example, the user has a menu where he can select different options. If a wrong input happens, he must undo the last step or in the worst case he must do all the settings again. A second requirement is that the recognition of certain inputs doesn't take too long time.

As mentioned in the showcase part, the framework contains seven, ready implemented, gestures. Implementation for these includes a ready template which can be used and all features are set. Also the framework has a menu and the possibilities to record new data or see what gesture has been detected (explanation can be found in the section "the menu system 6.7"). The list helps to evaluate how good the detection works.

For the test, some information about different test users is needed. The evaluation distinguish between the first group (in this case a single person), who has recorded the template data. The second group don't have ever come into contact with the system. At first, have a look to the person, who has made the gestures (the developer). It's an individual person with the knowledge how the gesture has been recorded. This is a great advantage over the others, in other words, he is perfectly familiar with it. He has the information at what speed and direction he must swing the device. For the evaluation, he must carry out every gesture 20 times in a row. The focus is on the number of correct recognized gestures. A second important factor is the investigation how often a false positive detection has occurs, whereby the counter has considered the whole test time span. With this test settings, following results can be seen in figure 8.1.

How often a gesture was correct detected is only one important property of the system. Also it must consider the false positive detection. Figure 8.2 represents the data about the whole test series about the individual user. As it can be seen, the concept works well where many features avoid true detection where it isn't one. Only by the LeftDown gestures occurs an error.

One reason can be that the left and the left down are to similar and can't be sufficiently distinguished from each other. Well, right and right down must be the same effect, but there has not occurred a wrong detection during the individual test. Another reason for the behaviour can be the fact, which hand is the dominant one of the user. In this case, the test person was right handed. He can't create the same feeling like by the other hand. This also shows

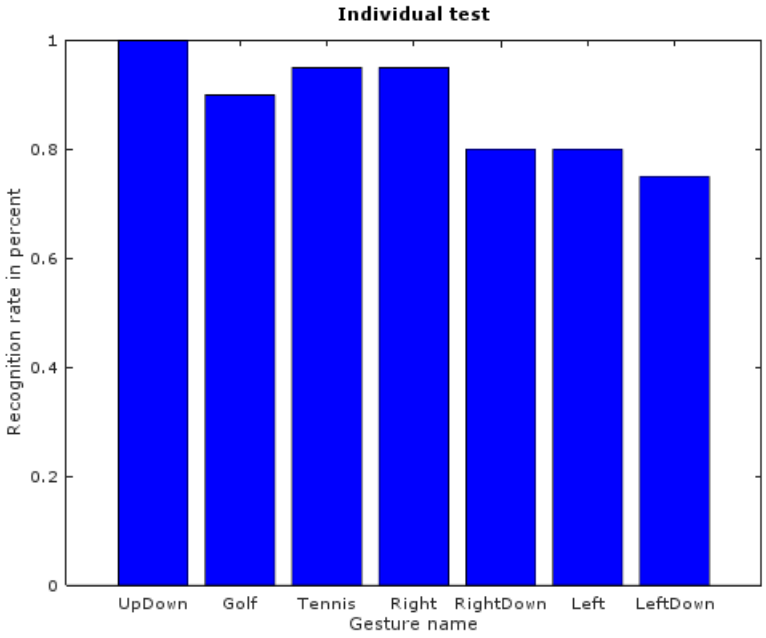


Figure 8.1: Recognition rate from the person, who has recoded the template data

the recorded data in the Appendix C section, where the right side data, the maximum is higher as by the left hand for the same gesture (absolute value 2 vs 4).

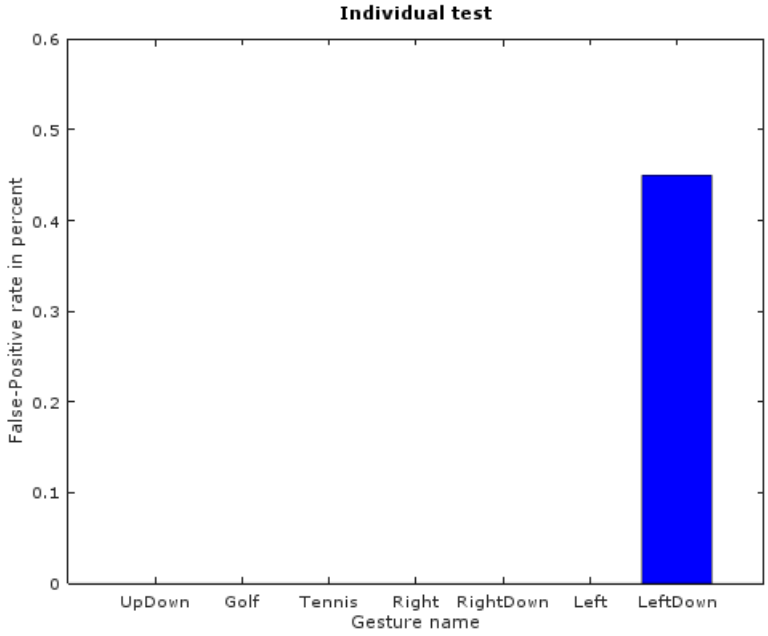


Figure 8.2: False-Positive detections during the whole test

The next test has been carried out by general user which didn't have any information about how the gestures were recorded. To have a certain number of valid test values, the user have carry out each gesture 155 times. Recognition rate and false positive results has been noticed the same way like by the individual tests. Before the test, he has 15 minutes time to become familiar with the system. The following graphs shows how the system has worked for that user:

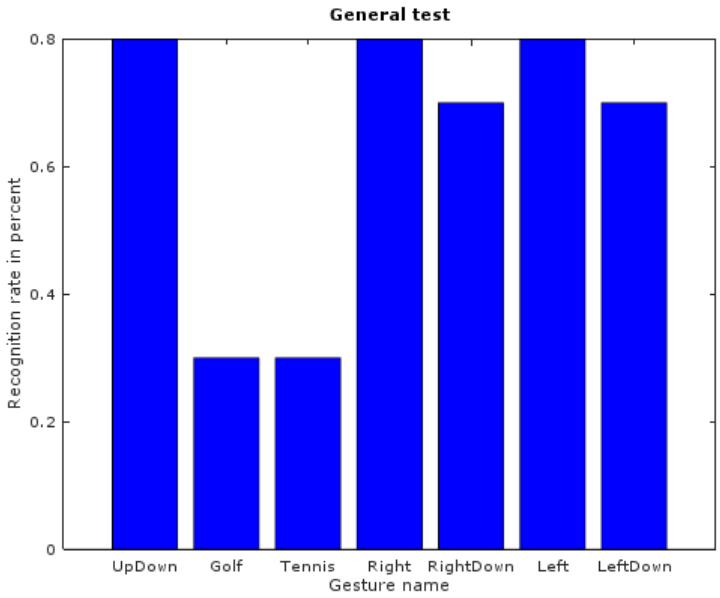


Figure 8.3: Recognition rate for a general user

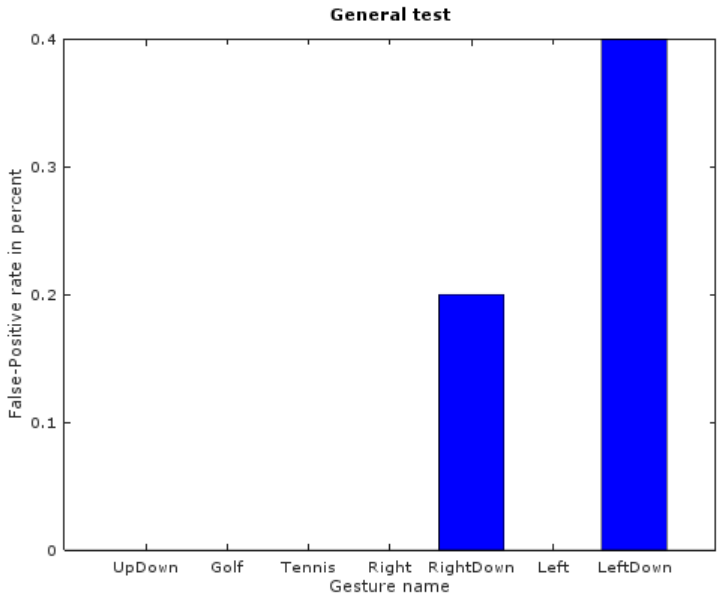


Figure 8.4: Recognition rate for a general user

As we can see, for the most gesture, the recognition rate is high. For two gestures, the result is not the best. That applies to golf and tennis. Note, that these gestures are more complex than the others. To improve the performance, it is advice to record more templates for them, ideally of several users. Also only for the gesture rightDown and leftDown occurs a false positive detection. As mentioned before the reason is that they are near similar to left and right.

This test shows where the limitations are about similarity between different gestures. It's always possible to use more than one template, but if they can't distinguish enough from each other, the system will recognize it one of them always as false positive. For complex gestures, it's necessary to have more templates to increase the recognition rate, otherwise it doesn't work reliable for every user.

Also better recognition can be achieved if a user often use the application. After certain time he knows in which direction and velocity he must swing the device. As in every area, training is important. If the developer observes the rules, it should be possible to write simple application to test some ideas.

8.2 Extendibility of the Framework

Another part of this thesis was to create a framework with which is possible to add new devices and sensor data. This section briefly presents how it can be done based on an abstract gadget. This abstract device can be a Smartwatch, another Smartphone or other useful alternative.

The first step is to create a script to setup the device. As discussed before, the InputData can be used to retrieve the desired device (Smartphone, Kinect, Smartwatch, CustomizeDevice1-4). The call looks like follow: InputData.Instance.SmartwatchDevice. This reference handles all data like gestures and sensor data, but it also contains information about current states and calculations. After the developer has retrieved the part he wants to extend, the next step is to tell the system about the structure of the information. As discussed in the implementation part, there are four categories available, namely:

- Sensor Data
- GestureData
- CustomizeData
- CalculatedData

Every of the four section can be added like the developer needs them. Note that for Smartphones it makes only sense for the CustomizeData area and for the Kinect everything except SensorData. The references can be changed by all classes which are derived from DeviceData. All of the four categories have getter and setter methods to change them. If they are untouched by the developer, they have a null pointer reference. Therefore, all information about how the frameworks structure looks like can be found in the implementation chapter (6).

If the variables for the data have been specified, the section can be used without any further settings for the Bluetooth communication. For the network communication, it's necessary to write the data manual to an output or read them from an input stream. The reference will be offered by the related abstract method, which must be implemented for all DeviceData instances. Also it's allowed to keep them empty, but note that it doesn't transfer any data to a device.

Related to the Smartphone, there exists a collector for diverse sensor data which can be used to save them, but also for gesture detection. Well, the framework cannot handle all kinds of data which offers the Smartphone or what in the future will be available. For that, the Window class has four sections for new data which has a time dependency (for example acceleration). To make it available, the developer can to following steps:

1. Create an empty game object in a Unity Scene.
2. Create a script which derives from the abstract class "AbstractCustomizeSensorCollector".
3. Implement the abstract method.
4. Retrieve the data from the device and add them to the correct section (currentData.customizeData¡1-4¿).

5. In the GestureDetection object, there is a child-object called "SmartphoneCollector". This object has a script attached called "RecordSmartphoneSensorData" where the Framework collects the data. As a developer, it is possible to define his own collector. For that there is a free place to add a new reference. To make the new data available, drag the new object to this place.

6. Now he can use them in the feature setting (choose the correct enumeration CustomizeData1-4).

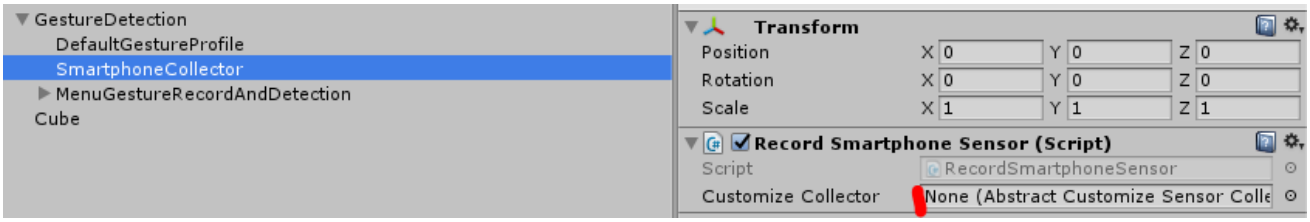


Figure 8.5: Place where the developer can set the object with his own collection data logic

8.3 VR-Interaction limits

The showcase covers many possibilities for interaction possibilities in the VR, but far from everything. With simple devices, it is possible to create applications for testing new algorithm, ideas and is a good alternative for beginners, who don't want to buy expensive solutions. Well, the developer must think about which devices he wants to use and after that decision he can think about how the application should look like. Every device has some limitation and is not suitable for every area of usage.

The showcases for testing application use two different devices which differ significantly. One use sensor with measure the physical conditions of the environment and motion, the other one determine a body to simulate them in the VR. Not only the available sensor in a gadget is important, but also how the user can server it.

8.3.1 VR-Interaction - Smartphone

At first, have a look to the Smartphone. It has built-in lots of sensors which are suitable to use them for gesture recognition. For this case, it works very good and reliable, but considers that the accuracy is not as high as by real, professional sensors. This is a point which must be always considered and has the consequence that the developer must avoid interactions, where the position of the hands is required. The error is accumulate over time and comes farther away than the actual position.

A second point depends also on the accuracy of diverse sensors. Every of them have a limited value range which can be detected. This limitation affects also the gesture detection system. For example it's possible to recognize a golf gesture, but it's hard to use more information. Sometimes the player hits the ball with less power, then again with more power. If we take a look of the recorded data, a normal strike has a high value and is determined near the maximum. There is not much room for detection the power. This was the decision for the showcase that the force must be set per touchscreen. But for that, also a solution exists. The developer can provide different templates about the same gesture, but with different forces. If a fast template has been detected as true, the power is higher and vice versa.

Also other problems can occur during the detection. If the user has a Smartphone in his hands, he can rotate them like he wants, but it makes it more difficult to handle the sensor data. The axis will retrieve other values, although it is the same gesture, but with other orientation. There exist two solutions for this problem. One is two consider the orientation of the Smartphone and transform the data in a reference space, the other is to record more than one template to improve the detection rate.

Here an overview of the limits affecting the sensors:

- **Sensor accuracy:** Use acceleration and co. only for interaction where no information about the device position is needed.

- **Value range:** Pay attention to the values of the different gestures, but also the maximum value which can detect the sensor. In most cases, it's not possible to use the information to determine the power.
- **Orientation:** Design gestures in a way so there are not many possibilities for the user to hold the devices in his hands. For example use touchscreen to provide the correct orientation.

8.3.2 VR-Interaction - Microsoft Kinect

The other way to create an interaction application is to use diverse sensors which can't be recognized by the user like infrared or camera sensors. They are working in a different way and allow the developer to realize more accurate manipulation.

It can be distinguish between two kinds of interactions. One of the two possibilities is to manipulate the world with the help of gestures and an avatar like as they are currently used in games. In other words the developer can to the same interactions as he uses a Smartphone. The different sensors allow him to use them in different environment conditions (lighting, etc.). Also it's possible to determine the correct hand position. This feature uses the infrared and depth sensor and already implemented algorithm retrieves certain body points as Joints. It leads to the second area of usage, namely to transfer the body data to the world and simulate the users skeleton. Basically all interaction can be done what the user can do in real life.

- Also making gestures
- Take objects and threw them
- For more advanced interaction: Virtual room with different people to communicate with language and hand gestures. With the help of a camera and image processing it's possible to map a texture of the user to the avatar. All these features generate a feeling of a natural human communication

The potential of the Microsoft Kinect is high, but has also some disadvantages and limits. For example, to use this kind of devices, the user needs some space in his environment. Without the place, it's not possible to detect the whole body and not all features can be used. It's not much tragic, because gesture detection with feet is not the common way. Also simulate steps can be difficult if the user doesn't move for- or backwards.

As mentioned before in the Microsoft Kinect section, the user must always look in the direction where the device is placed in the room. If the body rotates, the sensors can't detect all body Joints reliable.

In a virtual world, the room isn't empty at all. Without any objects, interaction is useless. You can touch objects and use them to solve certain tasks, but the limitation is how the user feels the interaction process. He can see that there is an obstacle or other kind of physical elements, but in the real world, there is only air. This makes the feeling not as real as in the real world.

A summary of disadvantage and limits can be taken from the following list:

- Much space is needed to use all features which are provided by the Microsoft Kinect.
- The user is restricted in the opportunities about the look direction.
- In the real world, there isn't any object which simulates resistance, temperature or other physically properties.

Chapter 9

Conclusion

Alternative devices has been studied which has some potential to use them as replacement for controllers and other hardware from VR companies. A reason and a big motivation is that not every headset has many possibilities to create input for manipulating the VR. Every device can be used as long as following conditions are met:

- It has the possibility to connect and send the data to the device where the VR app is running.
- The data can be collected and processed in real-time.
- Creating input must be intuitive and easy to carry out; otherwise it's not suitable for VR applications.

The result of the implementation is the finished framework which handles the basic logic and routines. Networking and basic gesture detection is the main task of it. A great feature of the framework is that the developer can extends it in a way as he needs it. The only thing is the device/platform where the VR program is running must be supported by Unity. For the client is necessary that the language is C#. A reason for it is that the serialization matches with the data structure of the server. This is only guaranteed if both have the same programming language.

For exchange data between two or more devices, the frameworks offer two different kind of realization. To have a possibility to choose a communication has the advantage that more devices can be supported by the framework. The types are Bluetooth and IP over network. Both are reliable and fast enough to create a good feeling in the VR. If too much data must be transferred, continuous data are allowed to lose one frame, but it not that often. Discrete data can be transferred more than once to guarantee it has been reached successfully the aim.

The extension works in different ways. It's possible to add new devices to the existing environment. This can be necessary if a gadget is more useful for a certain interaction technique or a new concept should be tested. Only one requirement is as mentioned before. A second type of extension is to collect new data. The framework can't know which sensor data a device offers and the development doesn't stop to build new sensors. Last but not least an important area is the gesture detection. Interfaces provide certain flexibility to record and analyse new data and have no restriction of numbers which can be detected at same time. It depends on the performance of the device itself.

The gesture detection system has a reliable recognition rate. But it depends on the person who uses the application. The one who has recorded the data has potential higher recognition rate as other users, who use it the first time. Two possibilities are exists to improve the performance. One is to use it often to get practise and find out in which velocity and direction the result gets better. The second solution is to record the same gesture more than once to have more templates. An ideal case is if the gesture is recorded from different person. Also it can be done by one person, but the minimum requirement is that the templates distinguish enough from each other.

Every gesture has its own properties. That leads to analyse the gesture for optimal feature setting. It is important to look which sensors are useful for which feature. Sometimes there exist only peaks which are ideal for min/max setting, for other a pattern which always occurs has a tendency to use DTW. Also an interface exists to define own feature if the framework doesn't offer the correct one. A second point is that every feature can be chosen as oft it's needed and the developer can combine them with no restrictions.

With the help of the framework, two showcases has been developed which has the aim to show how good it works. The used devices are an Android Smartphone and the Microsoft Kinect. A reason why these devices have

been tested was because they are widespread. Near everyone has a Smartphone and with the XBOX, the user often has a Kinect too. The developer isn't restricted to use an Android phone, it's also allowed to create an IOS application.

With Smartphone, the developer can collect input data which can be used to manipulation and interaction in the VR. It offers a touchscreen and diverse sensors. The sensor data can be used for gesture detection or to activate a certain access by moving in a certain direction. Acceleration values become the focus und the false positive detection can be minimized if more data like gravity and rotation rate will be used. Note that the accuracy isn't high, they only gives the developer a rough value. If the sensors are used to determine the hands position, than an error occurs which becomes bigger and bigger.

The touchscreen has also great potential for creating interactions. It can be used for menu input or select items, but also for gesture detection. Other possibility is to change states during a process, for example to trigger the next event as the golf area of the showcase has shown.

The showcase for the Smartphone part has realized different interaction with the help of diverse attraction, scattered in the whole world. A movement system has been realized with the help of the touchscreen and for testing the gesture detection system, a golf area has been designed. It has shown that the reliable of true detection is high enough to avoid many frustrating moments.

For the Microsoft Kinect, a second showcase has been created. It works namely with optical sensors like infrared, depth and camera. With them it is possible to detect gestures with hands or other body parts of the user. Also the developer has the possibilities to determine the correct positions of all body Joints which a single phone can't. But a big disadvantage is that the user is restricted to be in a room with a computer which must enough power. Also the place between the Kinect and user must be free. By using Smartphone and Google Cardboard as headset, the user can be everywhere he wants.

The showcase shows how it's possible to interact with the environment as simple as possible. It can be used for future work to do more simulations like walking through the world. But the realization is important, because the user can explore the world with no scripted moving path. The second task which solves the showcase is to draw a skeleton which represent the user's body. It's hard to notice a delay between VR and real world (real-time).

In the future the framework can be used and extend for many areas as the developer just needs it. It is reliable enough for communication and gesture recognition tasks. But it can be getting even better. For example to use other methods for gesture detection which has a better rates for users, who didn't have carry out the recorded data. This also minimizes the number of templates and computing power and speed up the development process.

The showcase for the Microsoft Kinect is kept very simple and shows only basic interaction tasks (moving, skeleton). If the physic is good enough it's possible to test catching object or to other natural movements like in real world. Another problem which should be addressed is what happens when the user touches an object. The feeling isn't great to touch an object in empty air. A possible solution is something to bind a device the hands around which returns feedback like vibration, etc. A second point is the skeleton which is drawn. It can be replaced by an avatar.

Bibliography

[1] VR application: https://books.google.at/books?hl=de&lr=&id=sLgtDAAAQBAJ&oi=fnd&pg=PR1&dq=virtual+reality+in+medicine&ots=EjaO1dvMg4&sig=TVAdr8w4Cfh0rRIbS-OoGKYUKOg#v=onepage&q=virtual%20reality%20in%20medicine&f=false

[2] Top 10 best Virtual Reality Headsets 2017 (Date: June 02, 2017): http://heavy.com/tech/2015/07/best-vr-virtual-reality-headset-glasses-goggles-oculus-rift-specs-review/

[3] Official website for Oculus Rift (Data: June 02, 2017): https://www.oculus.com/rift/

[4] Oculus Touch implementation description (Date: June 02, 2017): https://developer.oculus.com/documentation/unity/0.8/concepts/unity-ovrinput/

[5] HTCs Vive developer site (Date: June 03, 207): https://developer.viveport.com/eu/develop_portal/

[6] SDK Oculus Rift and Samsung Gear VR SDK (Date: June 03, 2017): https://resources.samsungdevelopers.com/Gear_VR

[7] Oculus Rift Spec and test (Data: June 03, 2017): https://www.wareable.com/oculus-rift/how-oculus-rift-works

[8] HTCs Vive Spec and test (Data: June 03, 2017): https://www.wareable.com/vr/htc-vive-vr-headset-release-date-price-specs-7929

[9] PlayStation VR Spec and test (Data: June 03, 2017): https://www.wareable.com/project-morpheus/sony-project-morpheus-release-date-price-games

[10] Samsung Galaxy Spec and test (Date: June 04, 2017): https://www.wareable.com/samsung/new-samsung-gear-vr-2016-review

[11] Get a Cardboard or build a headset (Date: June 04, 2017): https://vr.google.com/cardboard/get-cardboard/

[12] Google DayDream Spec (Date: June 04, 2017): http://www.androidauthority.com/daydream-vr-ready-phones-specs-727780/

[13] DayDream ready phones (Date: June 04, 2017): ttps://vr.google.com/daydream/smartphonevr/phones/

[14] Small, C. E.: Immersive Virtual Reality. No. SAND2011-1515C. Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States), 2011

[15] da Silva, P.R.D.C.: Smartphone Gesture Learning: Universidade do porto, faculdade de engenharia, Master Thesis, June 17, 2013

[16] Klingmann, M.: Accelerometer-Based Gesture Recognition with the iPhone: MSc in Cognitive Computing, Goldsmiths University of London, September 2009

[17] Henze, N., Boll, S. CJ: Gesture Recognition with a Wii Controller: ResearchGate, Article January 2008, DOI: 10.1145/1347390.1347395

[18] Akl, A., Valaee, S.: Accelerometer-based gesture recognition via dynamic-time warping, affinity propagation, & compressive sensing; Department of Electrical and Computer Engineering, University of Toronto (2010)

[19] Wang, H., Li, Z.: Accelerometer-based gesture recognition using dynamic time warping and sparse representation; Multimed Tools Appl, DOI 10.007/s11042-015-2775-2, Springer Science+Business Media New York (2015)

[20] Types of sensors (Stand: 29.05.2017): https://books.google.at/books?hl=de&lr=&id=dZjo-254FucC&oi=fnd&pg=PR27&dq=android+sensor+testing&ots=m3iRHBkAak&sig=DBL_c4DWRkMDXqFSz3aazd6JwXw&redir_esc=y#v=onepage&q=android%20sensor%20testing&f=falsehttps://developer.android.com/guide/topics/sensors/sensors_overview.html

[21] Official Android Developer side about supported sensors (Stand: 29.05.2017): https://developer.android.com/guide/topics/sensors/sensors_overview.html

[22] Kinect sensor (Stand: 29.05.2017): http://www.hongkiat.com/blog/innovative-uses-kinect/

[23] Zeng, W.: Microsoft Kinect Sensor and Its Effect, Multimedia at Work; University of Missouri, 1070-986X/12, IEEE 2012

[24] MIler M.: Information Retrieval for Music and Motion; Chapter 4, Springer (2007), ISBN: 978-3-540-74047-6

[25] C# Framework to calculate DTW with different methods and visualization examples (Date: 28.06.2017): https://github.com/doblak/ndtw

[26] Microsoft Kinect SDK for Unity (Date: 12.06.2017): https://developer.microsoft.com/en-us/windows/kinect/tools

[27] Skeleton with the Joints (Date: 21.06.2017): https://msdn.microsoft.com/en-us/library/microsoft.kinect.jointtype.aspx

Appendix A

How to start using the Framework

This section covers how the developer can use the framework. Basically, following must be installed on the device before the developer can start with developing:

- Unity 3D To create the application (game, simulation, testing, ...)
- Matlab/Octave To analyse recorded gestures and decision making about which features and thresholds are meaningful

Not only programs for developing are needed, also the user needs some devices to test a VR application. As well the framework handles different input gadgets, so an input possibility must be available. In other words, the following is necessary:

- VR Headset To run the VR application and in the case of the framework, it's optimized for Google Cardboard, but general it can be also used for other technologies which are suitable and has a support for Unity.
- Input Devices Smartphone, Smartwatch, Microsoft Kinect, ...

In addition to the above mentioned tools, following can be also useful, but they are not necessary:

- Blender To create 3D models for a VR world (scene)
- Gimp For creating textures and GUI elements
- Audacity To manipulate simple audio files or convert them into different formats

If the developer has all the requirements for developing a application, he can start with his project.

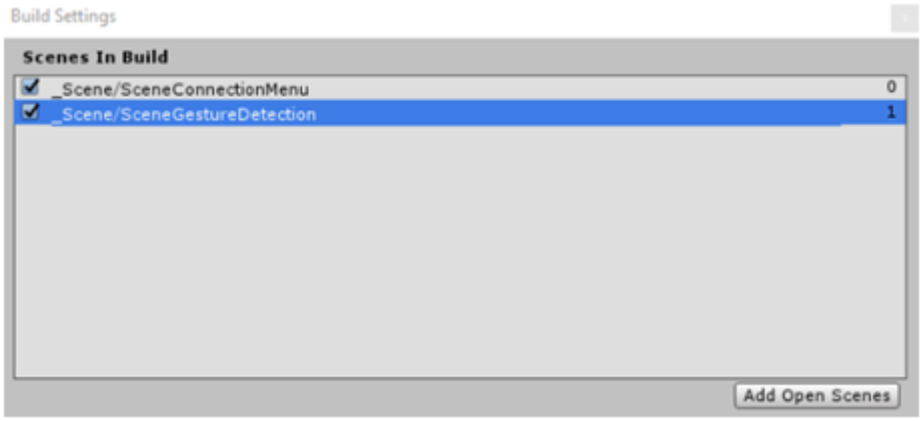
The first step is to create a new Unity project (give the project a name) which is empty at the start. In Unity, the developer has different windows, called inspectors. Look for the window called "Project" and click in there with the right mouse. There it should popup a menu where he can select the option "Import Package -¿ Custom Package". Now the developer gets asks where the package is located which should be imported into his project. Select the path where the package is located on the disk (standard name is **"VR-interaction_1_x"**) and check all assets to import them. After the importing process, the project has following sections/files:

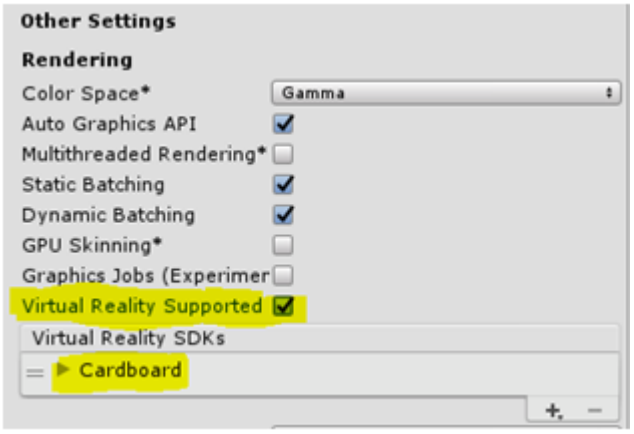
- VRinteractionPrefab (_Prefab):
  - In this folder, there are pre-defined Unity GameObjects which solve certain problems for the project. There exist objects for handling the network communication (client and server), to collect the body information from the Kinect, but also it offers diverse menu systems for all purpose (setup connection). In detail, the framework offers following objects:
    - BluetoothClient
    - BluetoothServer
    - NetClient
    - NetServer
    - GestureDetection
    - MenuRecordSensor
    - ConnectionMenu
- VRinteractionScene (_Scene):
  - Similar to the prefabs there exists a folder with pre-defined scenes which handles diverse tasks. In detail, it solves problems which occur again and again in a project like network setup and for basis gesture detection system. Only few settings must be done like tell the system which port should be listen or the UUID for identification of Bluetooth communication. For gestures the developer can use the defined templates with their thresholds, but he can also define own gestures and features for his porject. The scenes which are available from the beginning are:
    - SceneConnectionMenu:
    - SceneGestureDetection:
- ConnectionAPI:
  - ConnectionAPI contains all scripts which are necessary to build a connection between devices and for controlling the data flow. Both, network and Bluetooth are included.
- GestureDetection
  - The GestureDetection folder contains all scrips and information which are needed by the gesture detection system. It has included classes for data collecting, feature extraction and a decision system to tell the developer if the desired gesture, which he has defined, was performed by the user or not.
- Kinect:
  - As the folder names says, all necessary classes, scripts and logical flows for the Microsoft Kinect are available there. Two big parts are the data collection and drawing the skeleton in the VR in real-time.

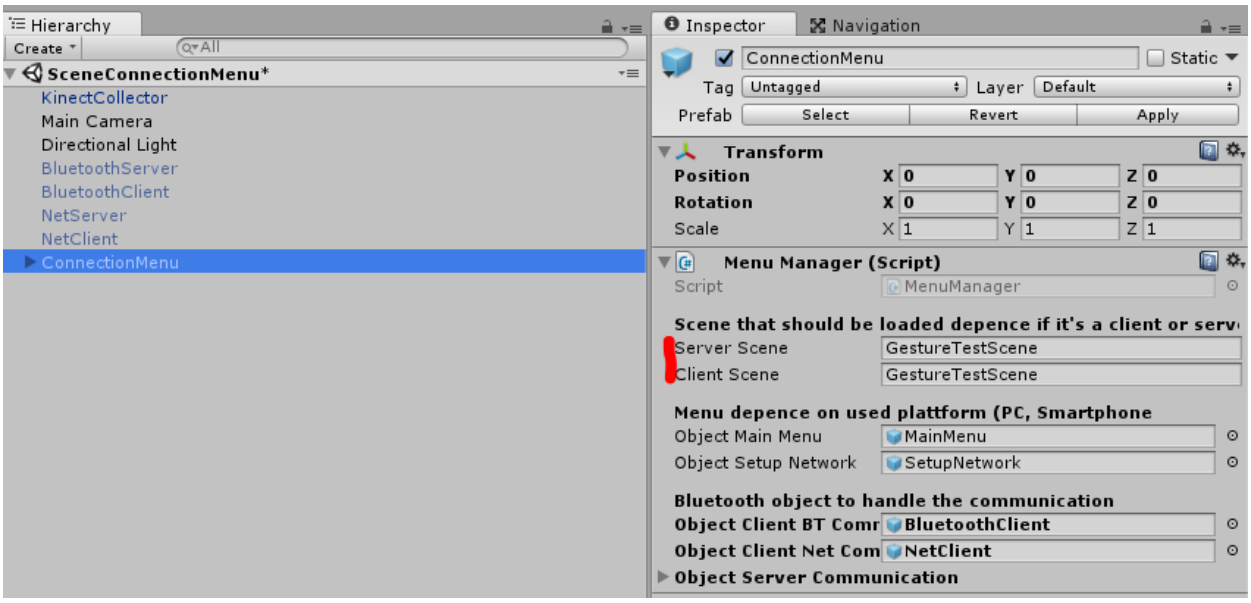
- GoogleVR:
  - This part of this project contains the GoogleVR framework which allows the developer to create an application for Google Cardboard. The used version is 1.5, but it is possible to change the bundle with a newer version.
- Resource/Plugin
  - This part has information about diverse plugins for GoogleVR and Bluetooth. As well there are located the data about all templates which can be read by the application during runtime.

If the developer imports the framework into his project, he can start immediately to code the logic. The first step is to add the two finished scene to his build settings (it's important that the menu scene about the connection setting will be start as first build number 0). If he did this step, he can run the application and will see the menu mentioned it in this master thesis writing. The same applies for the gesture detection part. You can remove the pre-defined gestures with its feature settings, so the developer can use them or add his own defined templates.

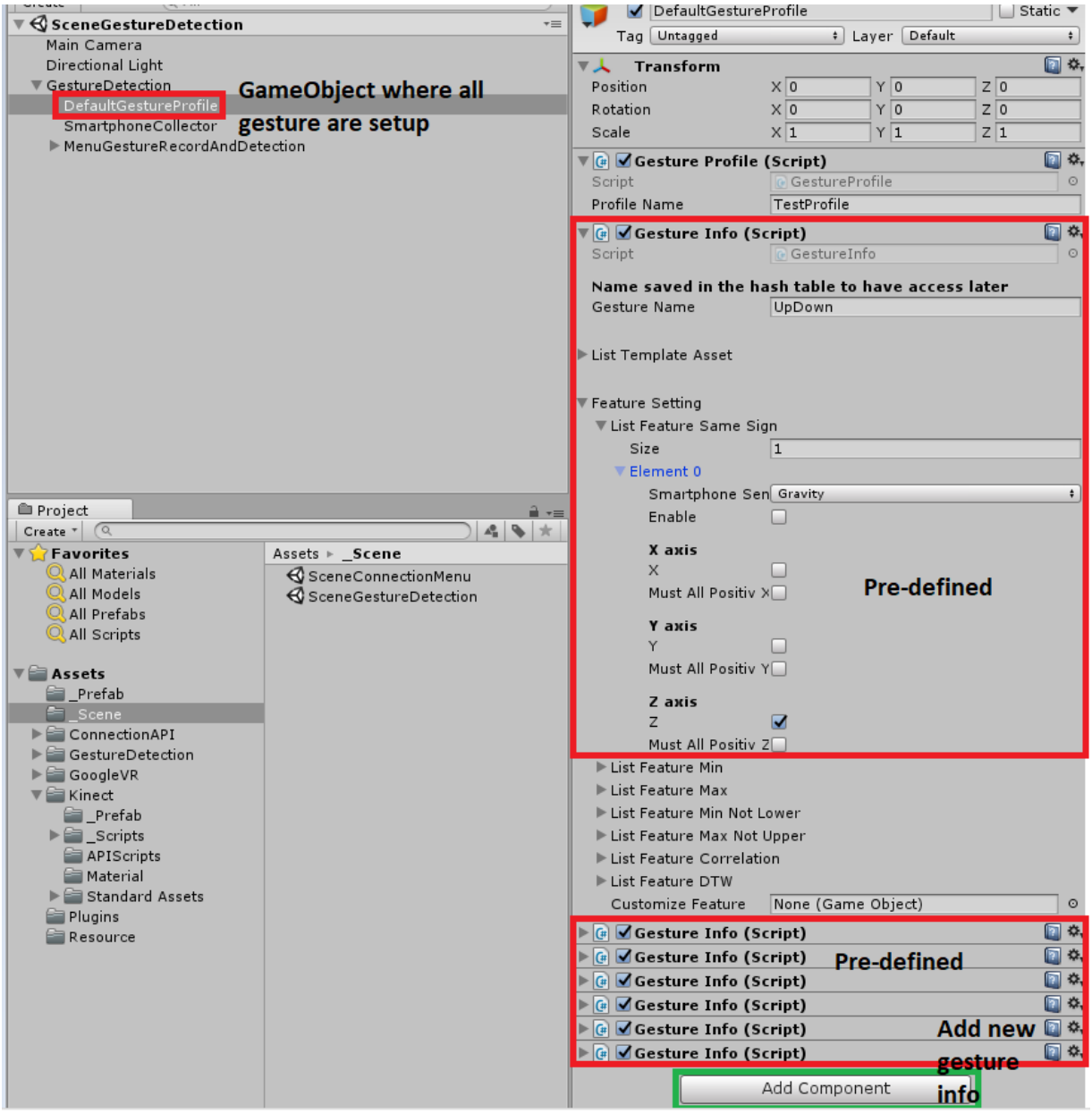
Depending on the player settings, the VR mode will be enabled or disabled and influence the menu which will be shown at the start-up of the application. If VR supported is checked, no menu will appear during runtime and the world will show up immediately. Uncheck the option to tell the framework that the application is a client and needs information about the server. This data must be entered by the user (kind of communication, IP-address, etc.). To control the scene which should be loaded, the SceneConnectionMenu has a GameObject where the developer can tell the system the name of the desired scene. Don't forget to add all scenes which should be loaded in the Scene Build, otherwise the application will crash.







The server scene is the part where the developer can implement all his ideas. This scene is running on the headset and with one or more clients, the manipulation and interaction of the VR could be possible, if it's supported by the developer. The client scene is the scene which depends on the used device for input creation. For the Smartphone, a pre-defined setup has been done with different gestures. The developer can decide to use them, but if it's not useful for his application, he can remove all templates and feature settings and define his own possibilities. To add a gesture, a new GestureInfo script must be attached to the object and the setup for it can be done. It will be automatic add to the hash table in the InputData instance and will be shown in the gesture detection menu.



Another part of the framework is to analyse the recorded gestures. With it, the developer can view the form of the gesture and make decision which features he needs. As well the thresholds for a certain sensor can be defined if he looks after peaks and other decisive properties.

If the developer has recorded some data, it's necessary to analyse them. For that purpose, the framework contains some Matlab function, where he can do with few steps draw the data and save a template file for further processing. Currently, only five data series are supported which are produced by the Smartphone. The data which are drawn at this moment are (look at Appendix C to see some example):

- Acceleration The acceleration of the device with gravity information
- User Acceleration The acceleration exclusive gravity information
- Gravity A vector which shows the direction of gravity
- Rotation Rate The speed of the rotation which was made by the user.

- Rotation Rate Unbiased More accurate measurement for the rotation rate

But with less effort the developer can fit them in the right form. General only the GestureAnalyse Matlab script must be adapted. At the beginning, the two variables filename and templateFile (line 5 and 6) must be assigned. The filename points to a file where the raw data of the gesture can be found. The templateFile variable is to name the template and save it to a certain location.

GestureAnalysis.m

```
1  %This program can be used to analyse the measured data of different sensors with
2  %regard to smartphones.
3
4  %filename = 'filename.txt';
5  filename = 'UpDown/UpDown.txt';
6  templateFile = 'LeftNew/tmp.txt';
```

If the program has the necessary information for the file handling, the next step is to read the data and save them temporary in an array. There is an option, which depends on the set boundary. The boundaries define the window which should be viewed. There exists a min (minBoundary) value and a max (maxBoundary) value. If one of them is smaller than zero, all data will be read into the system and visualized. A threshold can be also defined which influence the data. If the absolute value is smaller than the given value, it will be set to zero, otherwise it will be untouched. The manipulation only affects the drawing part, for saving the template, the original data will be used. If the data has read into the system and drawn, the data will be automatic saved.

```
41 LeftDown
42 minBoundary = 349;
43 maxBoundary = 390;
44 threshold = 0.05;
45
46 %The first three lines in the file are the acceleration data
47 [accX, accY, accZ] = ReadGestureDataRecording(filename, minBoundary, maxBoundary,0);
48
49 %The next three lines in the file are the acceleration data
50 [useraccX, useraccY, useraccZ] = ReadGestureDataRecording(filename, minBoundary, maxBoundary,3);
51
52 %Followed by the gravity data, as well splitted up in three lines
53 [gX, gY, gZ] = ReadGestureDataRecording(filename, minBoundary, maxBoundary,6);
54
55 %Next to last the rotation rate data
56 [rrX, rrY, rrZ] = ReadGestureDataRecording(filename, minBoundary, maxBoundary,9);
57
58 %Last but not least the rotationRate data, which can be biased or unbiased (depends
59 %on what the user has choosen).
60 [rruX, rruY, rruZ] = ReadGestureDataRecording(filename, minBoundary, maxBoundary,12);
61
62 %Save the template for future processing in the framework
63 SaveTemplate(templateFile, accX, accY, accZ, useraccX, useraccY, useraccZ, gX, gY, gZ, rrX, rrY, rrZ, rruX, rruY, rruZ);
```

To set the optimal boundary of the window, the information can be drawn in different ways. It depends on the set min and max value as mentioned before. Negative values cause everything to be drawn. Here the developer can see all the data of the current window and choose other values to get closer to the data of a single gesture.

```
65 %Drawing all data with given threshold (if abs(value) < threshold, set it to zero
66 DrawSensorData("Acceleration", threshold, accX, accY, accZ);
67 DrawSensorData("User acceleration", threshold, useraccX, useraccY, useraccZ);
68 DrawSensorData("Gravity", threshold, gX, gY, gZ);
69 DrawSensorData("Rotation rate", threshold, rrX, rrY, rrZ);
70 DrawSensorData("Rotation rate unbiased", threshold, rruX, rruY, rruZ);
```

The next image represents the data format which is used by the framework. This structure is needed for saving the raw data, for reading in a external analysing tool and saving a template.

Acceleration data - xAxis
Acceleration data - yAxis
Acceleration data - zAxis
User Acceleration data - xAxis
User Acceleration data - yAxis
User Acceleration data - zAxis
Gravity data - xAxis
Gravity data - yAxis
Gravity data - zAxis
Rotation rate data - xAxis
Rotation rate data - yAxis
Rotation rate data - zAxis
Rotation rate unbiased data - xAxis
Rotation rate unbiased data - yAxis
Rotation rate unbiased data - zAxis
If needed: Own data by the developer

Appendix B

All scripts and there location

The following section presents which scripts are available in this framework and where the developer can find them. Bold scripts and folders are important and often used (from the framework itself and for the showcase). The first list is about the framework itself, after that a list about the Matlab/Octave function is listed:

- DisableVR
- EnableVR
- MenuManager
- /ConnectionAPI/Script
  - DontDestroyHelper
  - EmptyDeviceData
  - **InputData**
  - SetupSmartphoneDevice
  - /Bluetooth
    - AbstractBTComponent
    - Client
    - ConnectionState
    - ServerForAndroidStudio
    - ServerScript
    - **/BluetoothObject**
      - BluetoothClient
      - BluetoothServer
      - BluetoothServerSetting
      - IMapServerData
    - /ExchangeData
      - AbstractDeviceDataComponent
      - **DeviceData**
      - GestureDetectionInfo
      - GestureDetectionInfoContainer
      - SetupSmartphoneData
      - SmartphoneCalculation
      - SmartphoneSensor
      - SmartphoneTouch

            - /Serializeable
            - SerializableQuaternion
            - SerilizableVector2
            - SerializableVector3
        * /Menu
            - DeactivateOnStart
            - HandleExit
            - LoadLevel
        * /Notification
            - IStateChangeNotification
            - NotificationRegistration
            - NotificationTextChange
        * /Setup
            - ConnectionSetting
            - HandlerButtonUUID
    - /Network
        * DeviceDataContainer
        * NetClient
        * NetServer
    - /Smartphone
        * CollectCalculateGravityAngular
        * CollectSmartphoneSensor
        * CollectTouchScreenData

- /GestureDetection/Scripts
    - FileHandler
    - GestureDetectListHandler
    - **/DetectGesture**
        * GestureFeatureSetting
        * GestureInfo
        * GestureProfile
        * GestureProfileDefault (not used -¿ empty)
        * TemplateData
    - **/Features**
        * AbstractFeature
        * FeatureCorreleation (deprecated, should not be used)
        * FeatureDTW
        * FeatureMax
        * FeatureMaxNotUpper
        * FeatureMin
        * FeatureMinNotLower
        * FeatureSameSign
        * FeatureZCR (Zero Crossing Rate -¿ not implemented!)
        * SpecificFeature
    - **/SmartphoneDataCollection**

        * CurrentSmartphoneSensorData
        * ISmartphoneCallback
        * RecordSmartphoneSensor
        * SaveSmartphoneSensorHandler
        * WindowSmartphoneData

- /Kinect/$Scripts$
    - CameraControl
    - CameraControlMovement
    - CollectBodyData
    - DrawSkeleton
    - KinectSensorData
    - SetupKinect
    - **/BodyData**
        * LeftArmData
        * RightArmData
        * LeftLegData
        * RightLegData
        * MainBodyData

For Matlab/Octave, following resources are available (note there are also recorded data saved which has been used by the showcase):

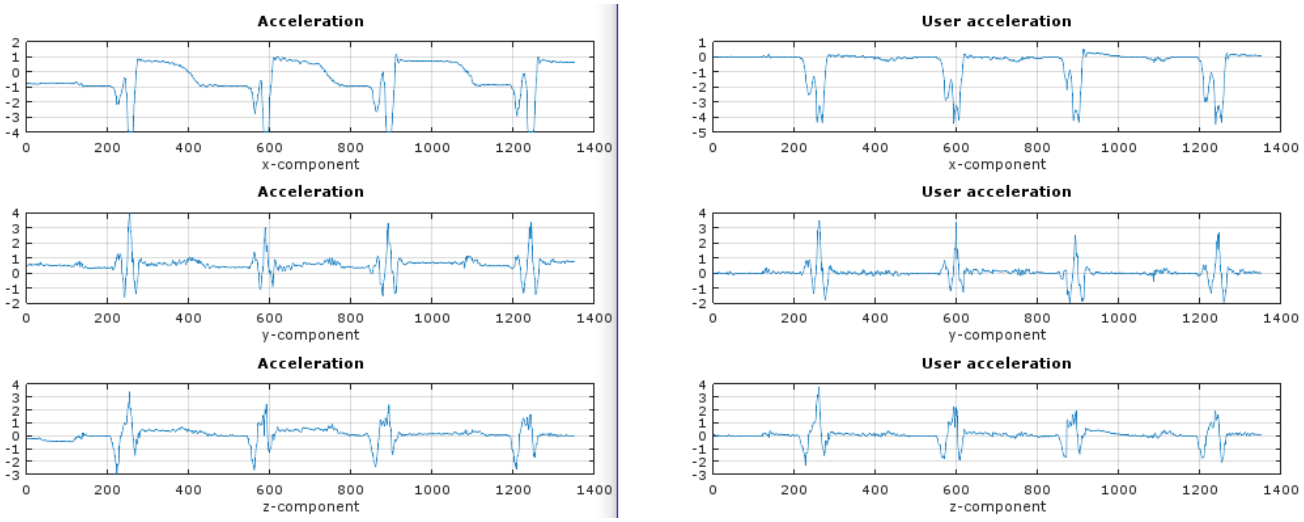
- /GestureDetection
    - DrawSensorData
    - GestureAnalysis
    - ReadGestureDataRecording
    - SaveTemplateData
    - /Golf
    - /Left
    - /LeftDown
    - /Right
    - /RightDown
    - /Tennis
    - /Updown

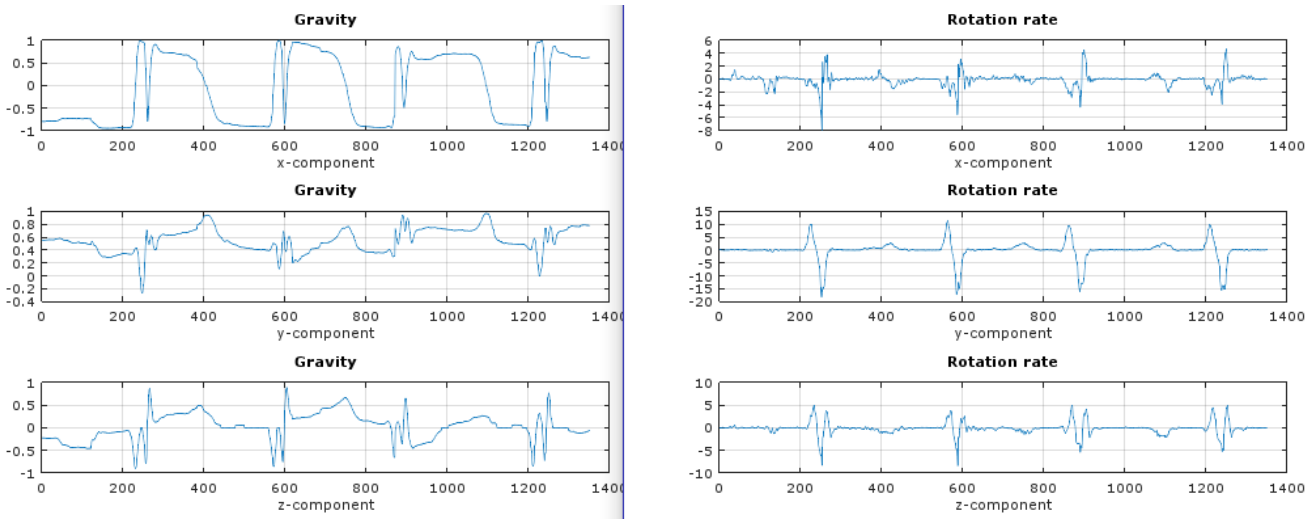
Appendix C

Recorded signals of used gestures

This section will briefly presents the recorded data from the different gestures which have been used for the showcase. Notice that the data includes a sequence of a certain gesture. It simplifies the template selection, because the developer can choose the one which has a strong meaning. The used device for the recording process was the LG G3. Rotation rate and Unbiased rotation rate has always delivered the same value for all time points. Also a table to each gesture is available where it can bee sees which features and thresholds has been used.

Golf:

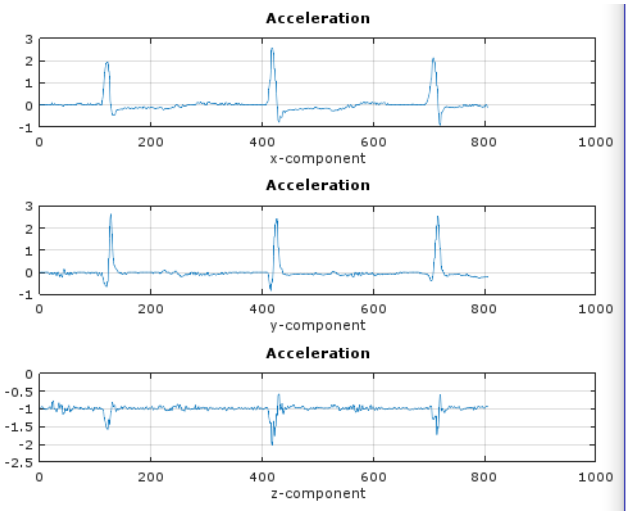


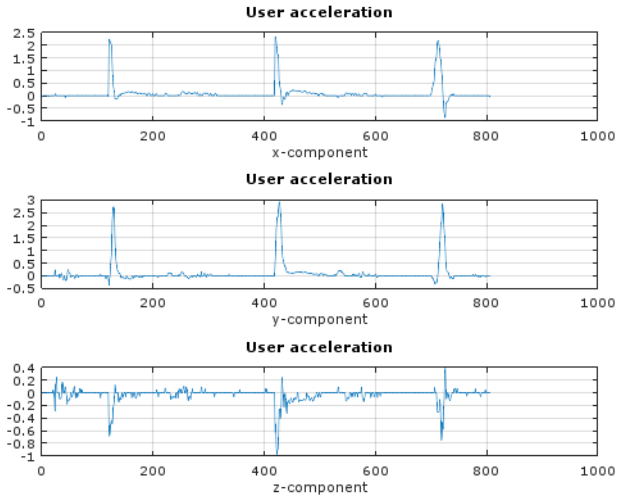


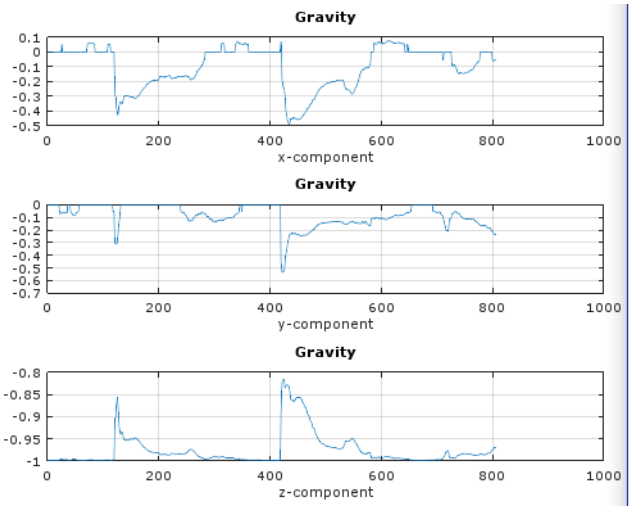
Gesture name:	Golf:
Same sign	Not used
Min	• Size:2 • Data1: – Enable - YES – Smartphone sensor data - Gravity – Automatic calculation - No – X axis - Yes ∗ X Threshold: -0.8 • Data2: – Enable - YES – Smartphone sensor data - Rotation rate unbiased – Automatic calculation - No – X axis - Yes ∗ X Threshold: -10

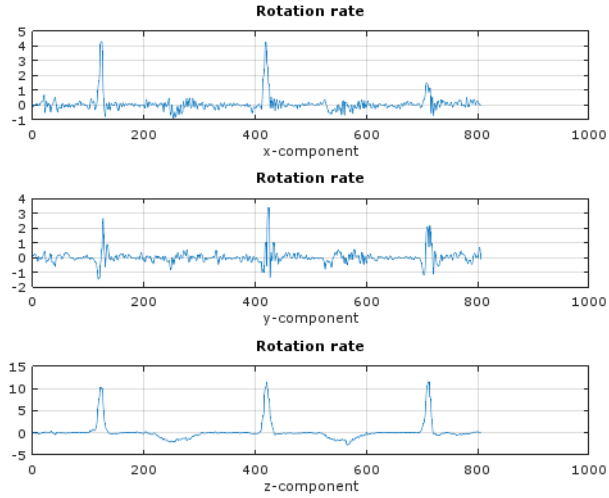
Max	• Size:2 • Data1: – Enable - YES – Smartphone sensor data - Gravity – Automatic calculation - No – X axis - Yes ∗ X Threshold: 0.8 • Data2: – Enable - YES – Smartphone sensor data - Rotation rate unbiased – Automatic calculation - No – X axis - Yes ∗ X Threshold: 6
Min not lower	Not used
Max not upper	Not used
DTW	• Enable - YES • Cutoff value: 0.1 • Smartphone sensor data - User Acceleration • Distance measure - Euclidean • Boundary constraint start - Yes • Boundary constraint end - Yes • X axis - Yes – X Threshold: 35 • Y axis - Yes – Y Threshold: 35 • Z axis - Yes – Z Threshold: 35
Customize feature	No

Left:







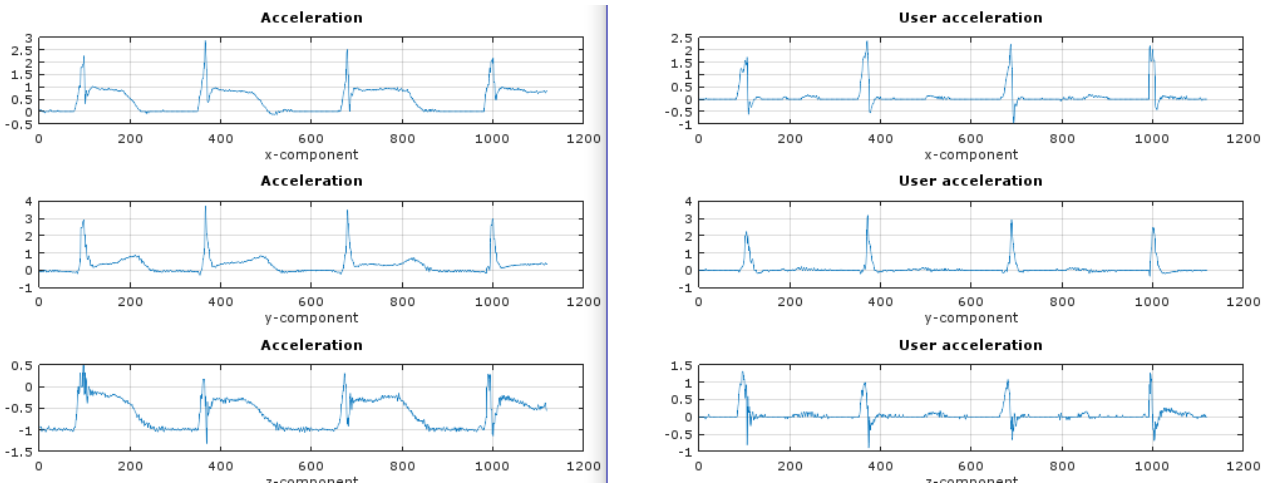


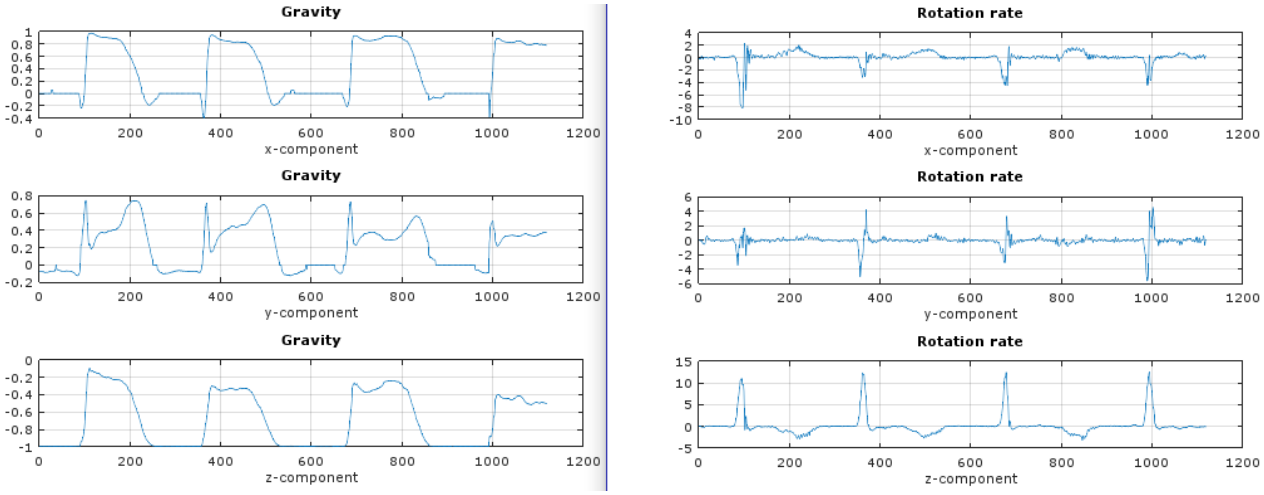
Gesture name:	Left:
Same sign	• Enable - YES • Smartphone sensor data - Gravity • Z axis -Yes – Must all positive z - No
Min	not used

Max	• Enable - YES • Smartphone sensor data - Rotation rate unbiased • Automatic calculation - No • Z axis - Yes – Z Threshold: 9
Min not lower	• Size:2 • Data1: – Enable - YES – Smartphone sensor data - Gravity – Automatic calculation - No – X axis - Yes ∗ X Threshold: -0.3 – Y axis - Yes ∗ Y Threshold: -0.3 • Data2: – Enable - YES – Smartphone sensor data - Rotation rate unbiased – Automatic calculation - No – Z axis - Yes ∗ Z Threshold: -3
Max not upper	• Enable - YES • Smartphone sensor data - Gravity • Automatic calculation - No • X axis - Yes – X Threshold: 0.3 • Y axis - Yes – Y Threshold: 0.3

DTW	• Enable - YES • Cutoff value: 0.1 • Smartphone sensor data - User Acceleration • Distance measure - Euclidean • Boundary constraint start - Yes • Boundary constraint end - Yes • X axis - Yes – X Threshold: 35 • Y axis - Yes – Y Threshold: 35 • Z axis - Yes – Z Threshold: 35
Customize feature	No

LeftDown:

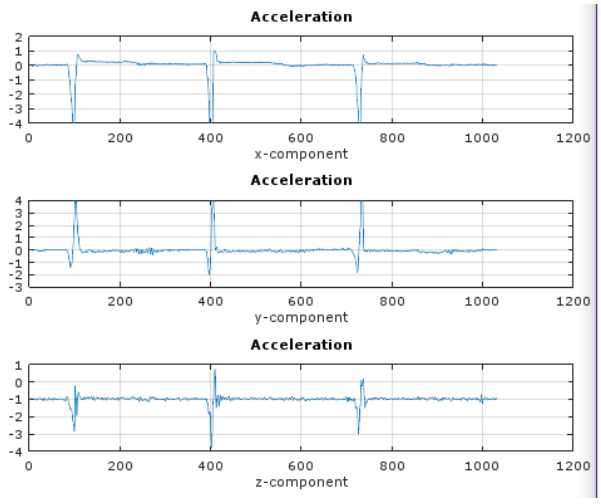


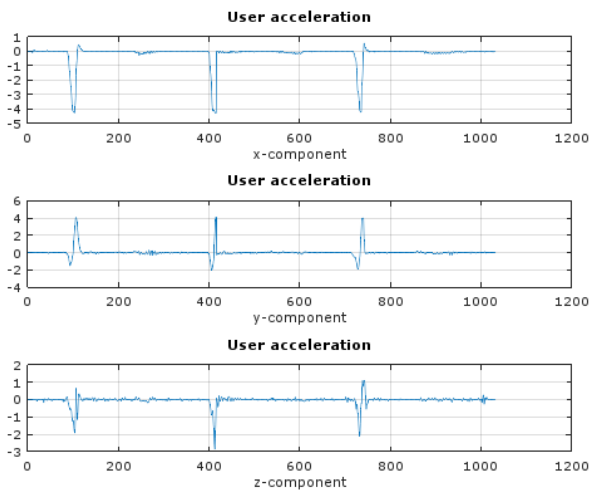


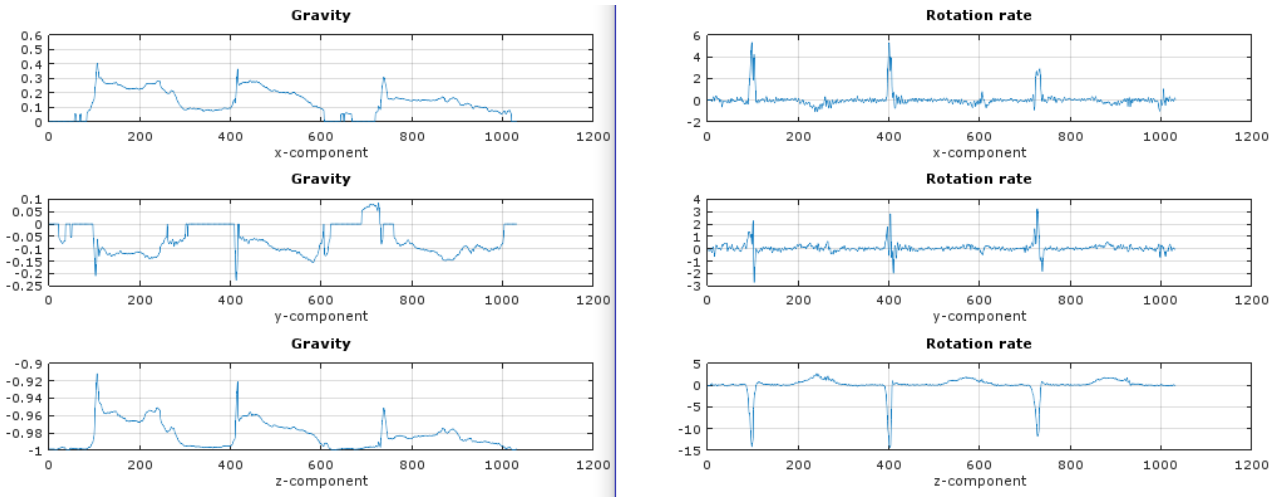
Gesture name:	LeftDown:
Same sign	Not used
Min	• Enable - YES • Smartphone sensor data - Gravity • Automatic calculation - No • Y axis - Yes – Y Threshold: -0.5
Max	• Size:2 • Data1: – Enable - YES – Smartphone sensor data - Gravity – Automatic calculation - No – X axis - Yes ∗ X Threshold: 0.7 • Data2: – Enable - YES – Smartphone sensor data - Rotation rate unbiased – Automatic calculation - No – X axis - Yes ∗ X Threshold: 0.7 – Z axis - Yes ∗ Z Threshold: 10

Min not lower	Not used
Max not upper	Not used
DTW	• Enable - YES • Cutoff value: 0.05 • Smartphone sensor data - User Acceleration • Distance measure - Euclidean • Boundary constraint start - Yes • Boundary constraint end - Yes • X axis - Yes – X Threshold: 35 • Y axis - Yes – Y Threshold: 35 • Z axis - Yes – Z Threshold: 35
Customize feature	No

Right:



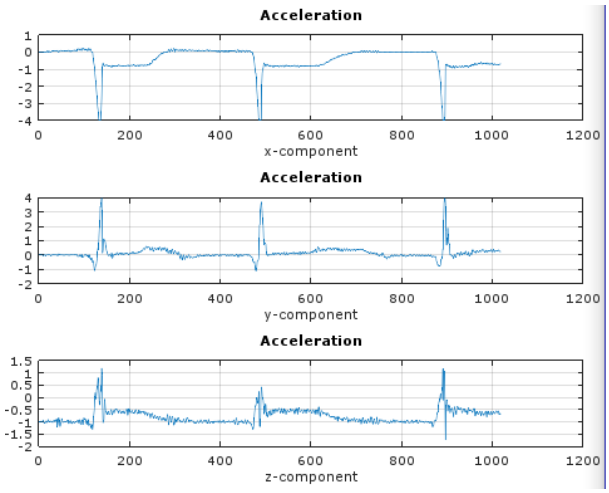


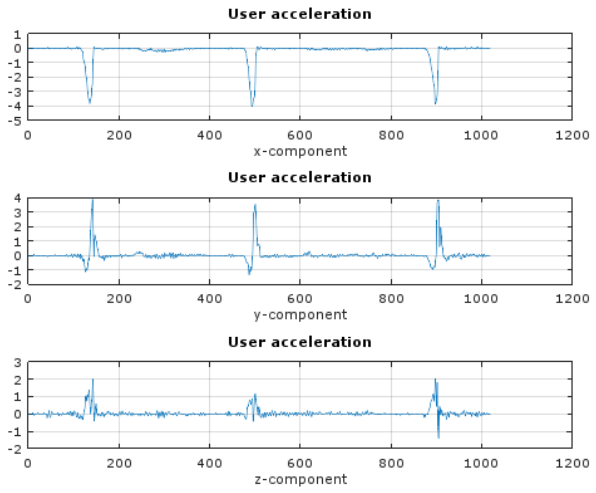


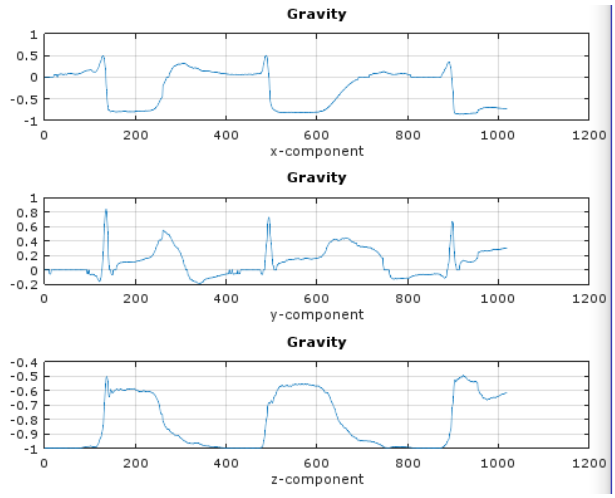
Gesture name:	Right:
Same sign	• Enable - YES • Smartphone sensor data - Gravity • Z axis -Yes – Must all positive z - No
Min	• Enable - YES • Smartphone sensor data - Rotation rate unbiased • Automatic calculation - No • Z axis - Yes – Z Threshold: -9
Max	Not used
Min not lower	• Enable - YES • Smartphone sensor data - Gravity • Automatic calculation - No • X axis - Yes – X Threshold: -0.3 • Y axis - Yes – Y Threshold: -0.3

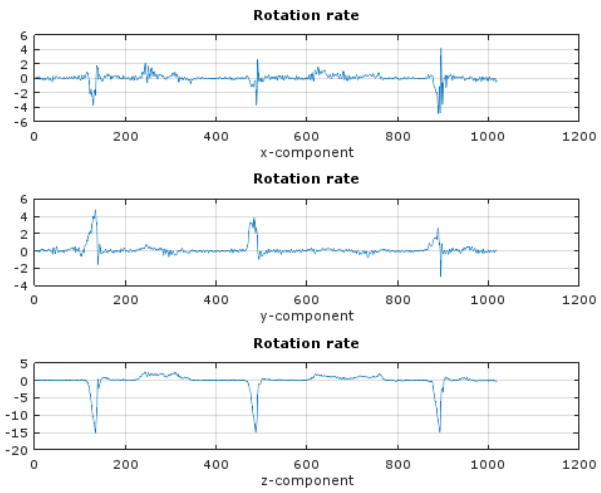
Max not upper	• Size:2 • Data1: – Enable - YES – Smartphone sensor data - Gravity – Automatic calculation - No – X axis - Yes ∗ X Threshold: 0.3 – Y axis - Yes ∗ Y Threshold: 0.3 • Data2: – Enable - YES – Smartphone sensor data - Rotation rate unbiased – Automatic calculation - No – Z axis - Yes ∗ Z Threshold: 3
DTW	• Enable - YES • Cutoff value: 0.1 • Smartphone sensor data - User Acceleration • Distance measure - Euclidean • Boundary constraint start - Yes • Boundary constraint end - Yes • X axis - Yes – X Threshold: 35 • Y axis - Yes – Y Threshold: 35 • Z axis - Yes – Z Threshold: 35
Customize feature	No

RightDown:







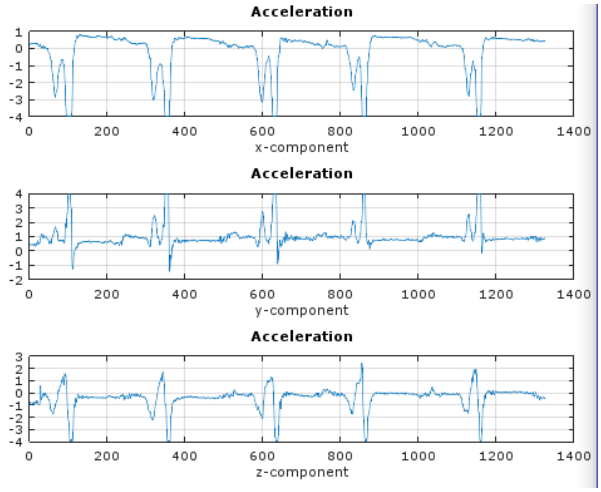


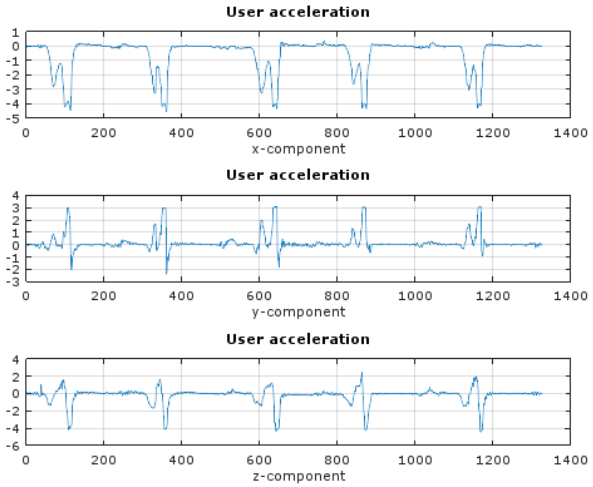
Gesture name:	RightDown:
Same sign	Not used

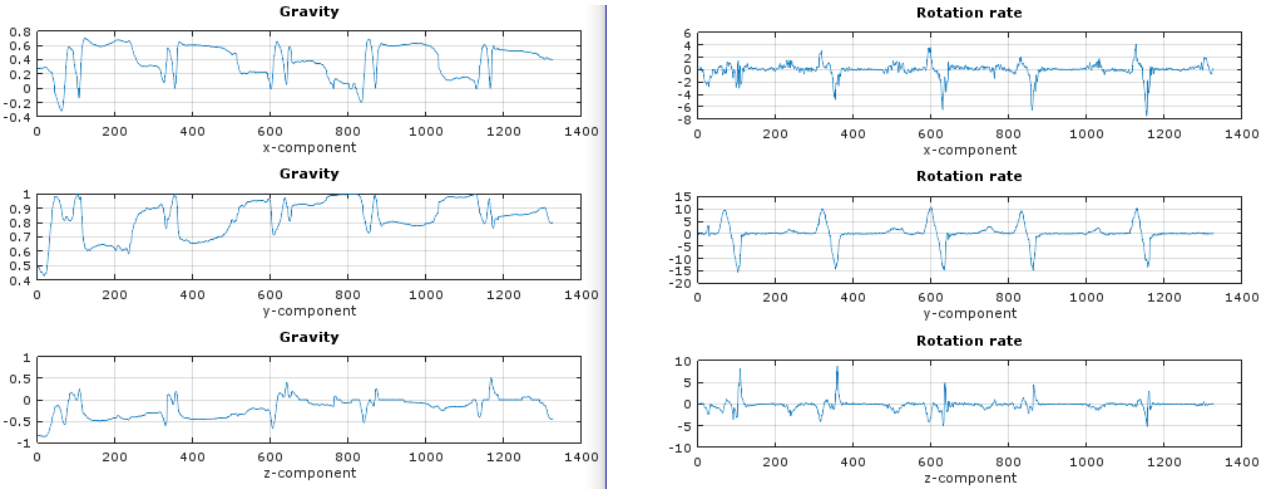
Min	• Size:2 • Data1: – Enable - YES – Smartphone sensor data - Gravity – Automatic calculation - No – X axis - Yes ∗ X Threshold: -0.7 • Data2: – Enable - YES – Smartphone sensor data - Rotation rate unbiased – Automatic calculation - No – X axis - Yes ∗ X Threshold: -0.7 – Z axis - Yes ∗ Z Threshold: -10
Max	• Enable - YES • Smartphone sensor data - Gravity • Automatic calculation - No • Y axis - Yes – Y Threshold: 0.5
Min not lower	Not used
Max not upper	Not used

DTW	• Enable - YES • Cutoff value: 0.05 • Smartphone sensor data - User Acceleration • Distance measure - Euclidean • Boundary constraint start - Yes • Boundary constraint end - Yes • X axis - Yes – X Threshold: 35 • Y axis - Yes – Y Threshold: 35 • Z axis - Yes – Z Threshold: 35
Customize feature	No

Tennis:



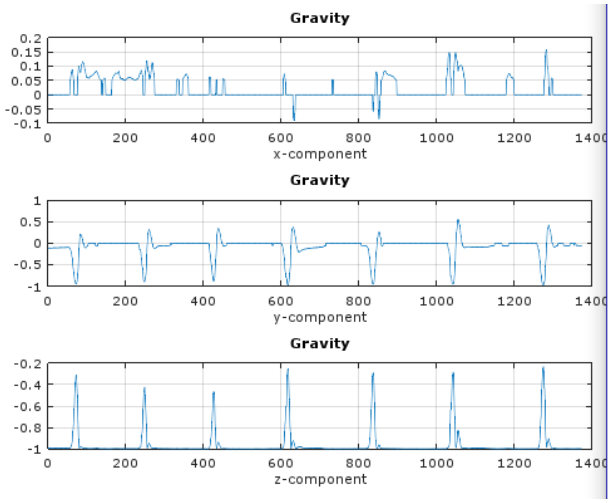


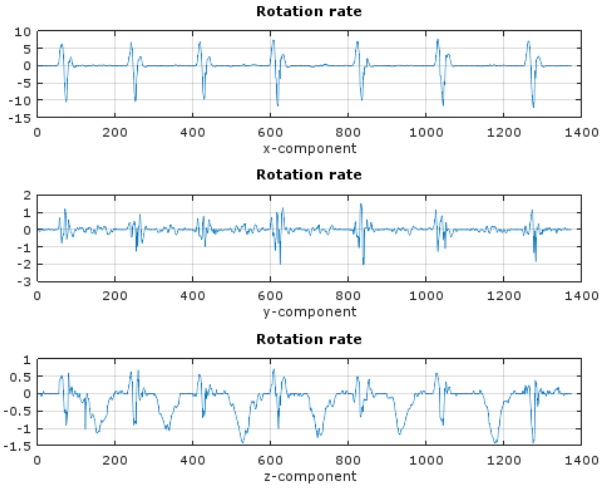


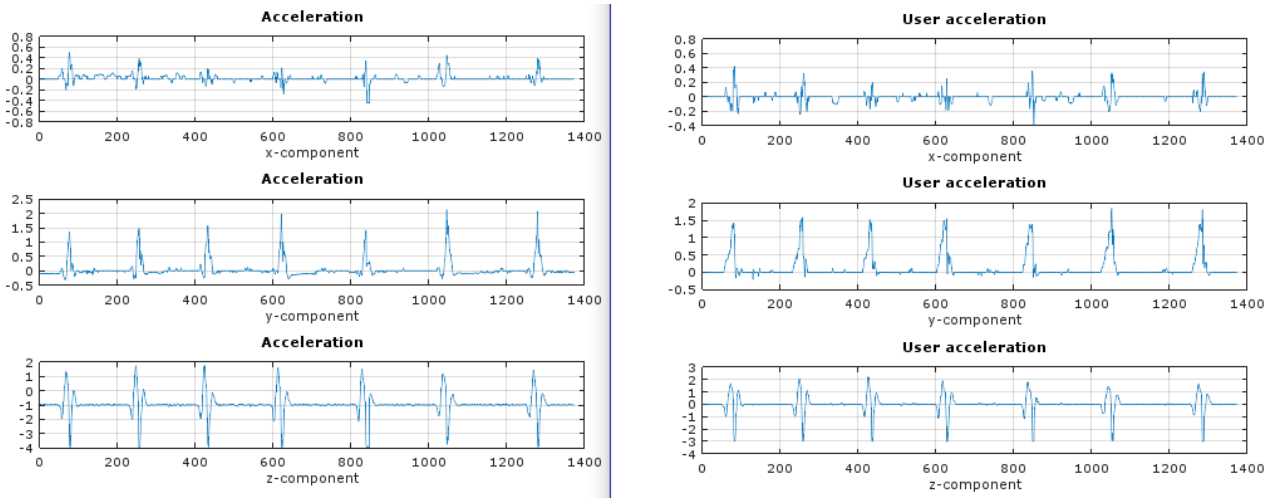
Gesture name:	Tennis:
Same sign	• Enable - YES • Smartphone sensor data - Rotation rate unbiased • Y Axis - Yes – Must all positive - Yes
Min	• Enable - YES • Smartphone sensor data - Rotation rate unbiased • Automatic calculation - No • Y axis - Yes – Y Threshold: -10
Max	• Enable - YES • Smartphone sensor data - Rotation rate unbiased • Automatic calculation - No • Y axis - Yes – Y Threshold: 5
Min not lower	Not used
Max not upper	Not used

DTW	• Enable - YES • Cutoff value: 0.2 • Smartphone sensor data - User Acceleration • Distance measure - Euclidean • Boundary constraint start - Yes • Boundary constraint end - Yes • X axis - Yes – X Threshold: 35 • Y axis - Yes – Y Threshold: 35 • Z axis - Yes – Z Threshold: 35
Customize feature	No

UpDown:







Gesture name:	UpDown:
Same sign	Not used
Min	• Enable - YES • Smartphone sensor data - Rotation rate unbiased • Automatic calculation - No • X axis - Yes – X Threshold: -6
Max	• Enable - YES • Smartphone sensor data - Rotation rate unbiased • Automatic calculation - No • X axis - Yes – X Threshold: -4

Min not lower	• Enable - YES • Smartphone sensor data - User acceleration • Automatic calculation - No • X axis - Yes – X Threshold: -0.6 • Y axis - Yes – Y Threshold: -0.5
Max not upper	• Enable - YES • Smartphone sensor data - User acceleration • Automatic calculation - No • X axis - Yes – X Threshold: 0.8
DTW	• Size:2 • Data1: – Enable - YES – Cutoff value: 0.1 – Smartphone sensor data - User Acceleration – Distance measure - Euclidean – Boundary constraint start - Yes – Boundary constraint end - Yes – Z axis - Yes ∗ Z Threshold: 25 • Data2: – Enable - YES – Cutoff value: 0.1 – Smartphone sensor data - Gravity – Distance measure - Euclidean – Boundary constraint start - Yes – Boundary constraint end - Yes – Y axis - Yes ∗ Y Threshold: 25

Customize feature	No

Appendix D

Abstract - English and German

D.1 Abstract

Virtual Reality (VR) is a research field which becomes more and more important. In the last two years, immersive reality has reached the next level to leave the laboratory and is now available for normal user. After releasing some headsets for the VR, the expectation was high, but currently it seems that the big hype is over. One reason is, that solutions like Oculus Rift or HTC Vive are not cheap (around 500$ and more) and a PC with high performance is needed. In opposite of expensive solutions, Google Cardboard is a cheap alternative. For this headset, the user only needs a Smartphone and a "Cardboard Headset", developed by Google. The target group are beginners, which are looking for to collect first experience in the VR, and those who do not want spend too much money for the equipment. From the idea to make it accessible for everyone, this work concentrates on how interaction in the VR-world can be done with devices, which users maybe have at home. Professional solutions often have own developed controller for this purpose to make the feeling of interaction more realistic, but for Google Cardboard, there do not exist gadgets like these. Currently it is possible to use a normal controller (the same as it is used for a PC or a gaming console). However, the user is limited to create a natural interaction (no motion/gesture). The output of this research is a framework to handle different devices, which are suitable for interaction with a standard controller. As well, some alternatives are better than other. So the second goal is to investigate advantages and disadvantages of each device and compare them. The focus is on the Microsoft Kinect and a Android-Smartphones.

D.2 Zusammenfassung

Virtual Reality ist ein Forschungsfeld, welches immer mehr an Bedeutung bekommt. In den letzten zwei Jahren, VR ist nicht nur mehr in diversen Labors zu finden, sondern es gibt bereits viele Technologien, welche der User auch ohne Problem zuhause verwenden kann. Das Problem zurzeit ist aber, dass viele Headsets, wie zum Beispiel Oculus Rift oder HTC Vive, nicht gerade billig sind (um die 500$). Auerdem wird zumeist ein Hochleistungs-PC benötigt. Eine Alternative zu diesen Headsets stellt Google Cardboard dar, wo lediglich ein Smartphone und die Cardboard Brille benötigt wird. Die Zielgruppe für diese Umsetzung sind Beginner bzw. Einsteiger, die ersten Versuche in die VR starten möchten. Das Hauptaugenmerk sind die Kosten, die hier nur gering ausfallen. Damit der User auch mit der VR Welt interagieren kann, werden verschiedene Eingabegeräte benötigt. Diese Arbeit konzentriert sich darauf, geeignete Alternativ-Geräte zu finden und zu untersuchen, welche Geräte sich dafür am besten eignen. Für den PC z.B. kann man einen normalen Controller verwenden, aber diese Interaktion mit der VR wirken keinesfalls natürlich (keine Bewegung/Gesten möglich). Das Ergebnis der Arbeit stellt ein Framework dar, mit dem es möglich sein soll, verschiedene Geräte einzubinden, mit denen Interaktionen in der VR umgesetzt werden können. Manche eignen sich dafür besser als andere. Somit ist das zweite Ziel der Thesis eine Gegenüberstellung von verschiedene Geräten durchzuführen, um Vor- und Nachteile für diverse Interaktionen in der VR aufzuzeigen. In dieser Arbeit werden hauptsächlich zwei verschiedene Typen von Geräten untersucht, nämlich die Microsoft Kinect und ein Android Smartphone.