Technical University of Crete, Greece

School of Electrical and Computer Engineering

An input architecture for hand interaction in virtual reality applications based on mobile phone platforms

Methimakis Michalis

Chania, October 2020

Abstract

The aim of this thesis is to take advantage of modern technologies related to the development of virtual reality applications as well as the special features and capabilities of modern mobile phones, to develop a software that allows the user to interact in virtual environments based on mobile phone platforms, using his hands.

In order to make this interaction possible, the user needs one or two additional smartphones, which are used as controllers. Concerning the software, two applications were developed. The first application needs to be installed on the controllers and it is programmed to utilize some of their technological features. More specifically, it "reads" the required data and then transmits them to the second application which handles them accordingly. Using this smartphone based connection and communication software as a core, a user-friendly interface has been implemented which offers the user some options. The user is able to choose which features of the controller he wants to use, for example a navigation joystick, adjust their position and size on the smartphone screen, as well as can save the profile of the controller he/she has just adjusted.

The application that receives the data from the controllers is used in two demo games for android platforms. The first game challenges the user to transfer some boxes using one controller to control a virtual hand in the VR environment. The motion of the virtual hand depends directly and completely on the motion of the user's real hand which holds the smartphone. The second demo brings the user in a virtual castle where he has the opportunity to practise on the sword and the shield. This makes it necessary to use two smartphones as controllers, one for each hand. Both games were implemented in the context of this thesis and take full advantage of the functions the controllers can offer.

Acknowledgements

- First of all, i owe my deepest gratitude to my supervisor, Assoc. Prof. Mania Katerina for providing me the opportunity to work on this thesis, as well as the guidance needed throughout the process.
- I would like to thank professors Samoladas Vasilis and Deligiannakis Antonis for their time reading and reviewing this thesis.
- Finally, I would like to express my gratitude to my friends and family for their endless encouragement and support in many aspects.

Contents

List of Figures

Chapter 1

Introduction

1.1 Brief Description

Virtual reality environments are created and presented to the user in such a way that they supersede the real world environment, creating suspension of disbelief and helping the user experience the virtual world as real. By enabling the majority of our senses such as vision, hearing, touch, even smell, the computer is transformed into a gatekeeper to this artificial world.

Virtual Reality's basic component is the head-mounted display (HMD). VR headsets are becoming more and more advanced and available on the market for anyone who wants to experience a virtual reality world. Devices such as the HTC Vive Pro Eye, Oculus Quest and Playstation VR are leading the way. There is a second class of Virtual Reality HMDs that is really just a shell with special lenses that pairs with a smartphone to deliver a VR experience. These devices can deliver a scaled down VR experience that still approaches the immersive experiences generated by computer based HMDs. [1]

VR applications require a different approach and design than the standard applications and games for 2D computer monitors, due to the nature of VR itself. More specifically, traditional input methods, such as the keyboard or mouse, are hard to be manipulated when the user is wearing the Head Mounted Display, so developers have to come up with new, more suitable input methods. Computer based HMDs come with special designed controllers so the user is able to interact with the virtual world. Another great approach is that of using a hand-tracking hardware as an input method. Concerning the applications which aim the mobile phone platforms the user can interact with the VR environment by looking at a specific direction, for example a button he wants to press. Combined with the fact that the game camera is also controlled by the head, this constant head movement might be uncomfortable for the user.

The concept of this thesis was to develop a software which allows the user to interact with a mobile platform based virtual environment by using his hands. It is important to point out that the user will not have to buy any hand tracking hardware, for example the Leap Motion device. Nowadays smarthphones are everywhere around us. Almost every family holds two to three smartphones in her possession. Modern smartphones are so advanced that they go beyond the concept of the phone itself. Their use is not only for making phonecalls. They are microcomputers with high computing power as well as a plethora of sensors to work with. The aim of this thesis is to take advantage of the special features and capabilities of modern mobile phones and use them as hand controllers for Virtual Reality applications.

The software developed in the context of this project, consists of two individual android applications. The first application needs to be installed on the controllers (smartphones) and it is programmed to utilize some of their technological features. More specifically, it reads the data which are required and then transmits them to the second application which handles them accordingly. The transmission happens through a Client-Server connection. UnityEngine.Networking library is used for the client's implementation. It contains a class which starts the client called NetworkClient. This is a class used by the networking system of Unity.

After the implementation of a NetworkClient instance the client uses the ip address of the server and the port number he listens to in an attempt of establishing a connection. If the connection is established successfully the controller gets a corresponding feedback. At this point the transmission of the data starts. These are the motion of a virtual joystick on the smartphone screen, the pressing of two virtual buttons and the data of the desired sensors (gyroscope, accelerometer) of the device. The data being transmitted are being refreshed in every frame. For every feature the controller offers there are at least two functions. The first function "pulls" the data from the smartphone and the second one sends them to the server. The transmission functions make use of unique id numbers defined in the server for each type of data. For example, the client's function that sends the gyroscope sensor data is being matched with the server's function that receives the gyroscope data.

Using the above software of the client as a core, a user-friendly interface has been implemented which offers some options. The user chooses which features of the controller he wants to use, for example a navigation joystick, adjusts their position and size on the smartphone screen, as well as can save the profile of the controller he has just created.

The implementation of the server required the use of the same Networking library offered by Unity. The first thing to be defined here was the port on which the server listens. When it starts, must be able to handle remote connections and their messages. In order to do that, a dictionary with unique id numbers for each type of the transmitted data had to be defined. The dictionary makes it possible for the server to "understand" and separate the different types of data he receives. Both the client and the server use a function for each type of data. The first is the transmitter while the second one is the receiver. These two functions are "paired" with a unique id so the server knows what messages he receives.

Using the above software of the server as a core, two demo games based on the android platform were developed. In the first demo the user is placed in an industrial warehouse. His task is to transport a number of packages from a workbench to another. The second demo is a simple form of a combat simulator. The user finds his self in a castle equipped with a sword and a shield ready to defend it. Both of them can be played by using the 2D screen of the android smartphone as well as in a VR mode. The actual goal of the two demos is to test the controllers functionality so they utilize all of their features.

1.2 Thesis Outline

This thesis is divided into six chapters. The first one is an introduction to the thesis. The second one has an introductory and educative content. The purpose of this chapter is to help the reader understand concepts like Virtual Reality and VR related hardware, 3D Graphics, Game Engines as well as informs about the evolution of mobile phones over the years and how they are linked with VR applications. The next three chapters focus on the development process of the applications as well as the platform used in order to achieve the final result. Chapter six includes the conclusions of this thesis and future work suggestions. A detailed explanation of the thesis's structure follows.

Chapter 2 - Backround:
This chapter defines what exactly Virtual Reality is, it's history over the years and most important makes clear that VR applications are not only for entertaining purposes. Another important topic of this section is a closer look to modern mobile phones capabilities and how are they related to Virtual Reality. Basic terminology is being explained here like the Frame Rate (FPS), 3D modeling, 3D rendering and there is also a subsection which explains what a Game Engine is, which are the most popular and which is the most useful for this project.

Chapter 3 - Technological Backround:
Chapter 3 focuses on the platform used during the implementation process. A detailed description of Unity 3D is provided including its architecture, project structure as well as the most important tools in building a project. These are the Scene Window, the Canvas, the Inspector with its components, Scripts, Game Window and Console. Also there is a brief description of GoogleVR software development kit, an essential tool for our project.

Chapter 4 - User's View:
Chapter 4 analyzes the User Interface of the applications. There is a brief description of every feature and virtual button the controllers contain as well as of the virtual environment of the two demo games.

Chapter 5 - Implementation:
This section describes the implementation process. The first subsection defines the Unity Scripting APIs and explains their importance in this project. Then there is an analysis of the structure of the server and the client implemented in this project as well as of the connection between them. The next subsections explain how the controller features are implemented and parts of the programming code are presented. Also there is a description of how the 3D environment of the demo applications are constructed and how the user can interact with them. Finally, all the applications were tested by a number of users and the evaluation results are presented here.

Chapter 6 - Conclusion:
The final chapter contains the main results of this project individually as well as in comparison with computer based HMDs. Also, ideas for potential future work on this project are presented.

Chapter 2

Background

This chapter covers the historical background and the evolution of Virtual Reality and mobile phones as well as the relation between these two. An evolution that makes us able to talk about and work with amazing VR systems and VR applications nowadays. Both of them are topics crucial for the project of this thesis. There is also an analysis of the most popular desktop VR controllers and how they became an influence for this project. In addition, in this chapter basic terminology of the 3D Graphics and the Frame Rate is explained as well as the definition of a Game Engine and its capabilities. All of these topics hold an important part in the implementation of this project.

2.1 Virtual Reality

2.1.1 Introduction

The virtual reality definition emerges from the definitions of both "virtual" and "reality". The definition of "virtual" is near and the one of reality is what we experience as human beings. Thus, the whole term "virtual reality" means more or less "near-reality". This definition refers to a specific type of reality emulation.

People are aware of the world through their senses and perceptional systems. Besides, having five senses was taught from the school years and these are taste, touch, smell, sight and hearing. Although these are only the most apparent sense organs. There are many more senses, such as a sense of balance. The combination of the above ensures the multiple ways of the perception of the environment.

Based on that and in the connection to the virtual reality definition, if someone presents the senses with made-up information, their perception of reality would also change accordingly. Unreal facts that might be presented as real ones, they can be perceived as real too. This is what virtual reality represents. Particularly, it describes a three-dimensional, computer generated environment which can be utilized by people. The user becomes part of this virtual environment. He/She is immersed there and is able to manipulate objects or perform various actions. [2]

2.1.2 History

The exact origins are difficult to be addressed because of how difficult it has been to formulate a definition for the concept of an alternative existence. The most simple example of virtual reality is a three dimensional (3D) movie. An "Experience Theatre" written in the 1950s by Morton Heilig which could encompass all the senses in an effective manner. In 1962 he built a prototype of his vision which was called "Sensorama", along with five short films to be displayed in it using various senses (sight, sound, smell, and touch). In 1968, Ivan Sutherland, having been assisted by Bob Sproull, created the first so-called head-mounted display system for use in immersive simulation applications.

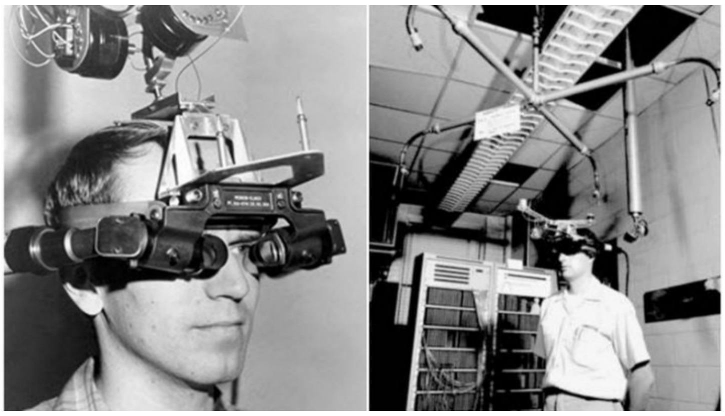


Figure 2.1: Ivan Sutherland's head-mounted 3D display (c. 1968). (Left) The system in use. (Right) The various parts of the three-dimensional display system.

The virtual reality industry is really fundamental as it offered VR devices for medical, flight simulation, automobile industry design, and military training purposes from 1970 to 1990. In 1988, the Cyberspace Project at Autodesk embedded on personal computers in an economical way. In 1990 the project leader Eric Gullichsen found Sense8 Corporation and developed the WorldToolKit virtual reality SDK, which provided the PC, for the first real time in industries and academia, with graphics with Texture mapping.

The 1990s there was the worldwide release of consumer headsets, such as Sega VR headset for arcade games, Nintendo's Virtual Boy. The 2000s was a period of relative public and investment indifference to commercially available VR technologies. Since 2010, VR had a rapid growth and by 2016, there were at least 230 companies with VR-related products, such as Amazon, Apple, Facebook, Google, Microsoft, Sony and Samsung.. Nowadays, there are several Head Mounted Displays like HTC Vive, Oculus rift, PlayStation VR as well as Smartphone-based headsets. [3]

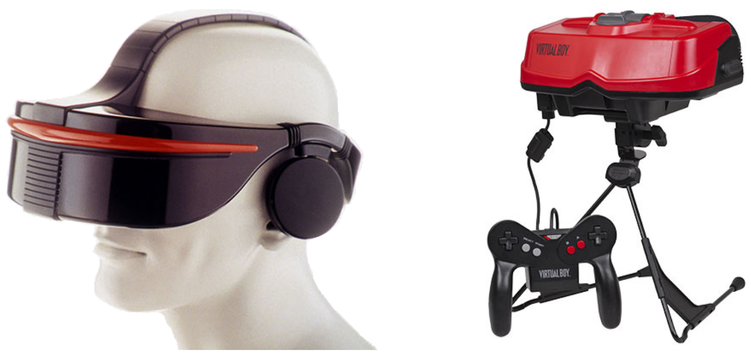


Figure 2.2: (Left) Sega announced the Sega VR headset in 1993. (Right) Nintendo's Virtual Boy released in 1995.

2.1.3 VR Applications

Virtual Reality is directly connected to gaming industry and entertaining objectives. VR applications compose of various contents, such as:

Flight and vehicular applications:
Flight simulators are a form of VR training. The pilot enters to a fully enclosed module or uses a computer with monitors which provide the pilot's point of view. Furthermore, driving simulators train tank drivers on the basics before they are allowed to operate the real vehicle. An other similar case is that of truck driving simulators for specialized vehicles such as fire trucks.

Military:
VR's sound and visual effects can be massive, and contribute to the improvement of the realistic feeling of a combat while preparing and training soldiers but without putting them at risk. Also, Virtual Reality creates a realistic environment without extra cost, by saving ammunition. It has been utilized for combined arms training and teach soldiers when to shoot.

Figure 2.3: Military's virtual training environment

Mental Health:
Virtual reality exposure therapy (VRET) is a form of exposure therapy for the treatment of anxiety disorders, such as post-traumatic stress disorder and phobias. In this way, conducted studies have indicated that the combination of this therapy with the physical one can reduce symptoms to an extent.

Space:
NASA's use of VR technologies is coming from the past which trains astronauts before flights. Particularly, VR simulations provides exposure to zero-gravity work environments and training of the walk in space and the use of tools in any case.

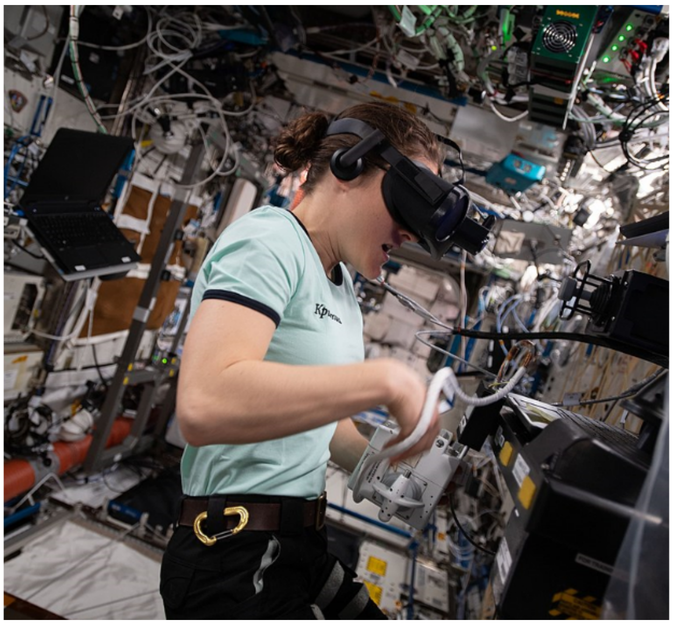


Figure 2.4: Flight Engineer Christina Koch wears a VR headset for the Vection study that is exploring how microgravity affects an astronaut's motion, orientation and distance perception in 2019.

Education:
This includes students' interaction in a 3D way. Virtual field trips to museums, taking tours of the solar system and going back in time to different eras are some of the examples of the education experience through VR. Autistic or with other special needs students can benefit from Virtual reality as they can safely practice social skills for children in general. Technology company, Floreo, has created a variety of virtual reality scenarios such as pointing, making eye contact and building social connections in which parents as well can also follow along and interact with the use of a linked tablet. [4] [5]

2.2 Frame Rate

Frame Rate, commonly measured and referred to as frames-per-second (FPS), is the frequency at which a hardware device is able to draw or capture consecutive images, called frames. The term refers to technical specifications of film and video cameras, computer graphics and motion capture systems. It is also a measure of performance of games and 3d applications. Concerning the video, film computer graphics and even more in Virtual Reality, Frame Rate functions critically, as it decides if the consecutive frames to be perceived by the brain as separate images or not, making so the illusion of physical motion.

Although the process of the human visual system can theoretically comes to 1000 different images a second, untrained eyes can hardly tell any difference above 60fps or 100fps based on studies, depending on the variants of the display device and usage. For example Virtual Reality devices usually require a higher than usual fps rate to create the pleasant illusion of a smooth motion, a fact which is difficult to be accomplished, as achieving high frame rates requires hardware with substantial processing power and thoughtful application programming. Especially in the case of smartphone-based applications, like the project of this thesis, this task is even harder to be achieved because of smartphones' lower performance compared to desktop computers.

2.3 Modern Mobile Phones

2.3.1 What is a smartphone?

In this day and age, classic mobile phones have been replaced by smartphones. The latter is a mobile device that combines both cellular and mobile computing functions into one unit. They are characterized with their strong hardware capabilities and extensive mobile operating systems, promoting the function of wider software, internet (including web browsing over mobile broadband), and multimedia functionality (including music, video, cameras, gaming), alongside core phone functions, such as voice calls and text messaging much easier. [6]

2.3.2 Capabilities

The development of the smartphone emerged by the several key technological advances. The exponential scaling and miniaturization transistors down to sub-micron levels during the 1990s-2000s not only contributed to the creation of portable smart devices, such as smartphones, but also it opened the way to the transition from analog to faster digital wireless mobile networks. Other important enabling factors include the lithium-ion battery, an energy source for long battery life, invented in the 1980s and the development of more mature software platforms. Modern smartphones carry a variety of special traits. Hence, they have the great ability to provide people with plenty of services and sometimes things that in another case, that would be impossible to be done through them. Some of their special features, are shown below:

- **CPUs with incredible computing power**
- **Large amount of RAM memory**
- **They can run graphically demanding games**
- **Plenty of memory to store data**
- **Strong connection to the internet**
- **Rechargeable, long-lasting batteries**
- **High resolution widescreens**
- **Fast, flexible software systems**
- **Various sensors that can be leveraged by their software (such as a magnetometer, proximity sensors, barometer, gyroscope, or accelerometer)**

As a result, they can be a very useful tool for our daily life but also for cases in which they were not intended to be used. One of these cases is the use of a smartphone as a controller in VR applications, as explained in the concept of this project. [6]

2.3.3 Smartphones and Virtual Reality

With the great and worldwide usage of the smartphones, VR can be easily be adopted from the people. There are many wearable VR displays on the market which consist of a head mounted case with focusing lenses, into which the user inserts a smartphone. The phone plays the role of the computer running the application as well as of the display, showing a stereo pair of images. How successfully your brain can be tricked into believing you are actually in another virtual world depends on the quality of the phone, concerning the extent of it to let the VR make the proper impression of your presence in that world. The following features of a display are essential when the goal is a realistic state of presence: a large field of view; low image "persistence" and high screen refresh rate in order to avoid latency effects; high screen resolution. All these are designed in this way so as to take advantage of the tricks the mind uses to perceive depth of field and focus and for true 3D vision. More specifically:

Image persistence and the screen refresh rate:
Image persistence and the screen refresh rate are closely associated, and in both cases, faster is better when refers to the real world replication. Persistence is the term used to identify the time it takes for a new image to replace the current one. The image appears sharper as persistence decreases. In the case of a high persistence, the image will seem blurry.

The screen refresh rate, is the number of times per second a display screen can update its showing image. Image persistence and refresh rate are inversely proportional. High screen refresh rate means low image persistence. OLED displays can offer more than a 1000x faster response rate than LCD displays so they are an excellent choice for a smartphone VR. Every millisecond plays a crucial role for the achievement of a realistic status of presence in the virtual world. Simulator sickness is also attributed to motion blur and jitter, which both are eliminated with OLED screens. In addition, VR-ready smartphones use OLED screens to reach the highest image quality and power efficiency (e.g., cooler and longer operation), and smaller form factors.

Another major trait which accomplishes truly realistic virtual reality is latency. Latency is defined as the time from when you move your head to when you actually see the correctly rendered view. Low latency is crucial in constructing a believable virtual space. In virtual reality there is a delay between the movement of your head and the one of the image in the VR headset. If the delay is too long, the VR immersion will seem unnatural and most importantly, the disparity with your brain's understanding of normal movement can lead to nausea or dizziness.

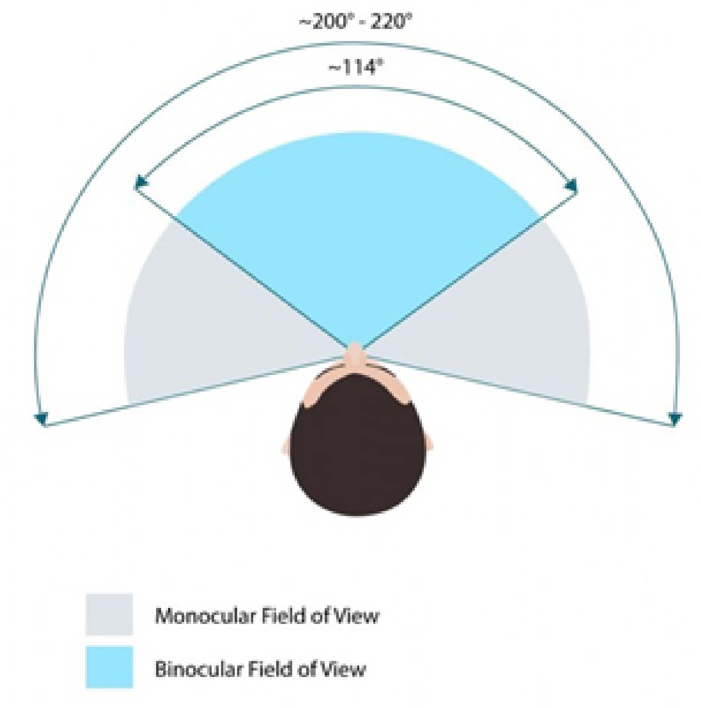


Figure 2.5: Human eyes Field Of View.

Field of View:
Field of view is the extent of the observable environment at a given moment. It is one of the most fundamental aspects of VR as the wider the FOV, the more likely the user will feel as he/she lives the experience like in reality. Depth of focus and 3D vision are fulfilled with the combination of monocular and binocular vision of the FOV. It is achieved in smartphone-based VR headsets by placing the phone on a specific and appropriate distance from the headset's lenses, and also that the lenses are large enough. If you want to

reach a better FOV, there is either the way for you to get closer to the lenses or increase their size.

Resolution:
Screen resolution as measured by pixels per inch (ppi) is another essential feature for smartphone-based VR systems. Because smartphone displays are placed fairly close to the eye in VR headsets and are magnified by the headset's lenses, pixels are likely to be seen. Higher-density OLED screens eliminate this problem. [7]

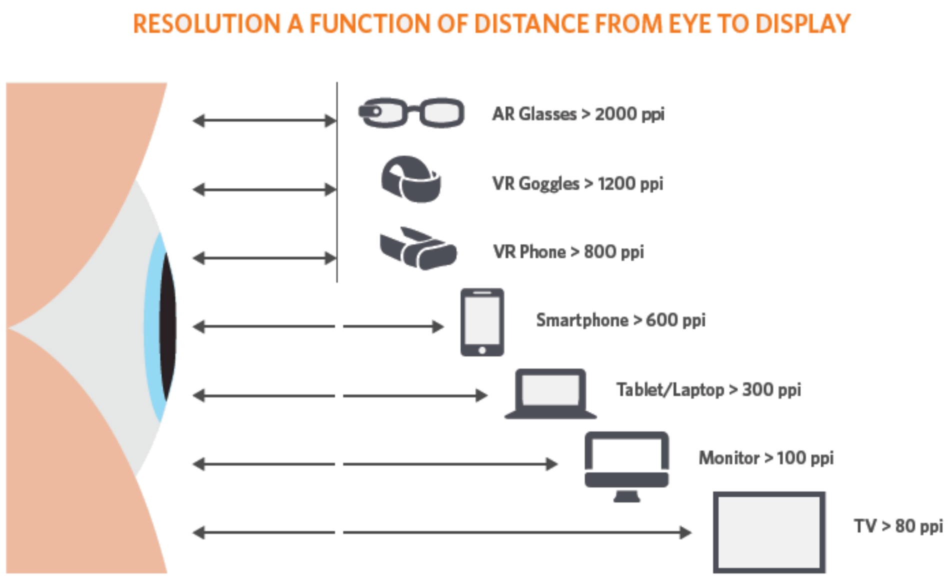


Figure 2.6: Resolution, a function of distance from eye to display.

2.3.4 Interaction in Smartphone based VR Applications

Interaction amidst the human and the virtual world is the key in virtual reality immersion as it is indisputable to feel your presence and express your power in it. Does it worth when being somewhere that the world doesn't react to you and you can't interact? A world that ignores your presence will not contribute to make you feel part of him.

So far, there was a great attempt made for the users to utilize their hands while playing. One way of interaction is the case of putting an interface cartridge printed with a conductive pattern into the mobile HMD. Concerning this interface cartridge, a controller with particular contents must be used in order to achieve high immersive interactive VR content. For example, to realize a fishing game, the device can use rotation actions as the input gesture, to realize a throwing interaction, it can use a swiping hand movement as the input gesture. As a result, the device needs to be manually adjusted according to the application as well as this is an outdated technique according to our age.

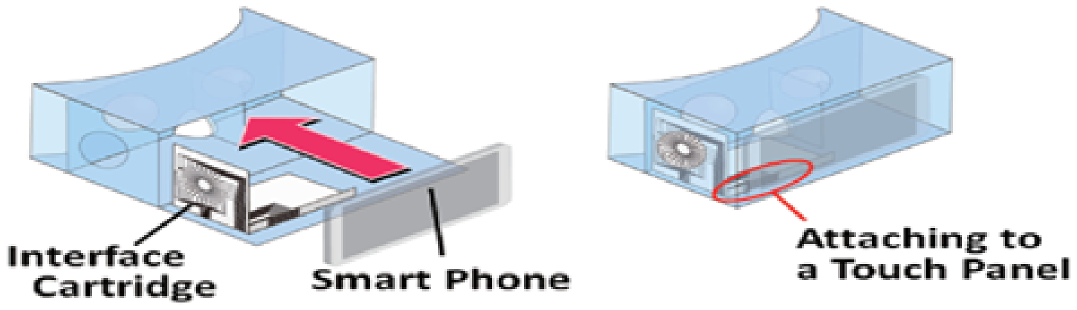


Figure 2.7: Interface Cartridge printed with a conductive pattern into the HMd.

On the other hand, an interesting and innovative approach is that of the hand gestures. In January 2017, an algorithm for gesture recognition with First Person View was presented, with a four swipe model (Left, Right, Up and Down) for smartphones through single monocular camera vision. Another common way to interact into VR contents is to use a touch sensitive surface, as an input device. The attachment of this surface can be done either on the VR case or the user's hand. [8] [9] [10]

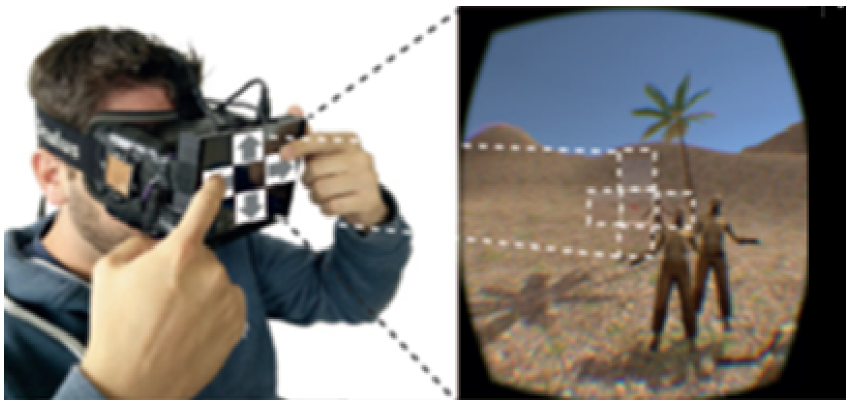


Figure 2.8: Touch sensitive surface attached to the HMD.

2.4 Desktop VR Controllers

The concept of this thesis came after personal experience with computer based virtual reality applications. As mentioned above, the two most popular Head Mounted Displays for such applications are the Oculus Rift and the HTC Vive. Both of them come with a hardware equipment which includes controllers. The user can interact with the virtual environment using his hands, making so the VR experience more enjoyable. Thus, a need arises for the creation of controllers for smartphone based VR applications and this is the purpose of this thesis. During the development of the software that implements the controllers, a study was made of the external layout of Oculus Touch and HTC Vive hand controllers.

As shown below, each of the Oculus Rift controllers offers the user plenty of interaction elements. More specifically, it contains a 2-dimensional motion joystick which can also be used as a button. In addition, it has two buttons the use of which depends on the game, a start button and a touch sensitive area, so that the controller is able to sense user's finger. There the user can rest his thumb without the fear of accidentally pressing a button. Finally, it offers two triggers, one at the back of the controller and one at the side. [11]

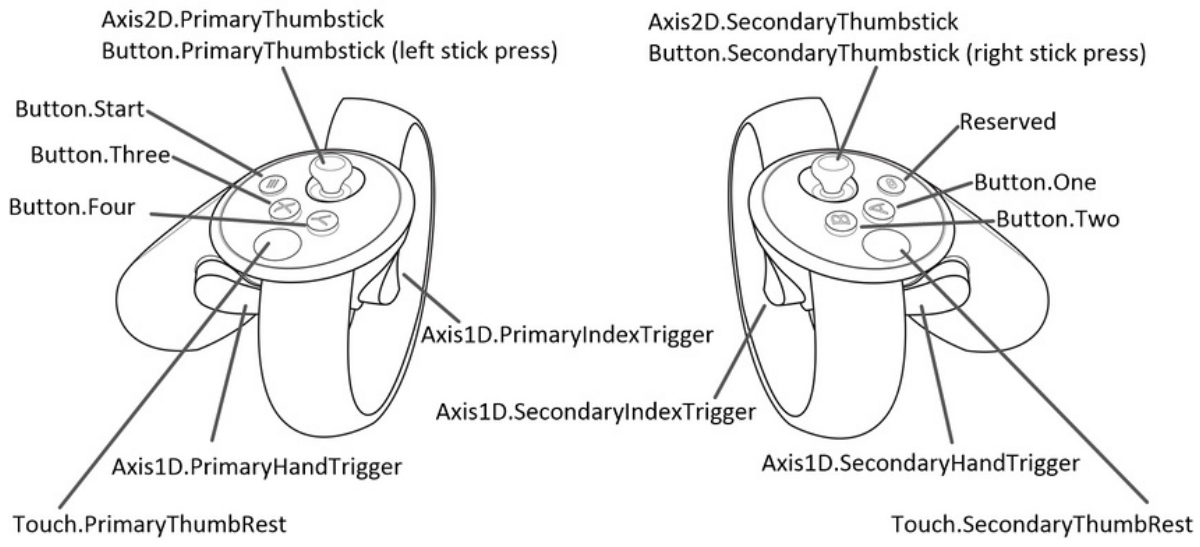

Axis2D.PrimaryThumbstick
Button.PrimaryThumbstick (left stick press)
Button.Start
Button.Three
Button.Four
Axis1D.PrimaryIndexTrigger
Axis1D.PrimaryHandTrigger
Touch.PrimaryThumbRest
Axis2D.SecondaryThumbstick
Button.SecondaryThumbstick (right stick press)
Reserved
Button.One
Button.Two
Axis1D.SecondaryIndexTrigger
Axis1D.SecondaryHandTrigger
Touch.SecondaryThumbRest

Figure 2.9: Oculus Touch Controllers preview.

Looking at the image below, it's obvious that the HTC Vive offers a controller with limited features compared to Oculucs Touch. More specifically, it offers a trigger on the back, a grip button on the side and the user's thumb controls a trackpad which is located on the front of the controller and works like a joystick. Finally, there are 2 buttons to control the system and display any menu. [12]

Studying the above VR controllers, an attemt was made to create the controllers of this thesis in such a way as to offer a plethora of features. However, this project remained at a programming level so trigger-type buttons could not be implemented as they require physical construction. The buttons that the controllers can support are virtual and located at the screen of the smartphone. Adding plenty of buttons was a feasible and easy process but the smartphone screen is a flat surface, thus creating a problem. The user can not feel which button is touching, nor see with his eyes as he is in a VR environment. So the controllers were limited to the menu control buttons and two basic buttons with which the user can interact with the game.

In addition, a virtual joystick is added which can serve functions similar to the Oculus

Touch Thumbstick and HTC Vive TrackPad. Finally, the user is offered the opportunity to interact with the virtual environment by using the rotating motion of the smartphone. In order to do this, the software developed in this thesis utilizes data from the gyroscope and the accelerometer contained in modern mobile phones. A feature that is also present in the controllers of the Oculus Rift and HTC Vive.

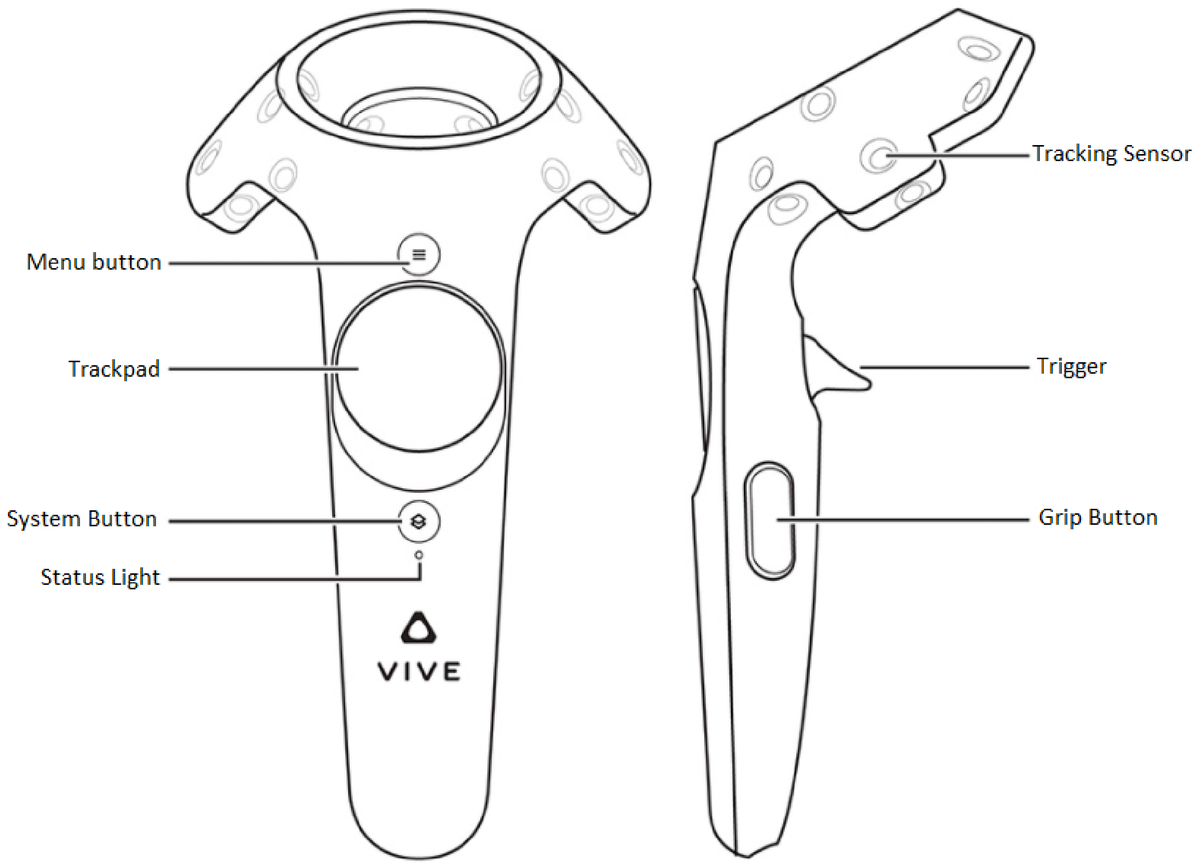


Figure 2.10: HTC Vive Controllers preview.

2.5 3D Graphics

Computer graphics is a field focusing on generating and displaying three-dimensional geometric data in a two-dimensional space (e.g., computer monitor, screen). The difference between a single point in a two dimensional graphic and a 3D point is that the former has the properties of position, color, and brightness, whereas the latter shows also a point on an imaginary Z-axis, giving a depth property. The modeling process with the layout and interactivity of the modeled objects in a scene is the start of 3D graphics. Then the 3D rendering follows, which refers to the computer calculations required to display the graphic on a screen.

2.5.1 3D Modeling

3D modeling is the 3D surface formation of an object. The product of this process, the 3D model, is created with a variety of geometric shapes such as lines, triangles, and curved surfaces which link certain points in a 3D space. This creation is either attributed manually to a modeling tool, algorithmically or it can be scanned into a computer from real world objects.

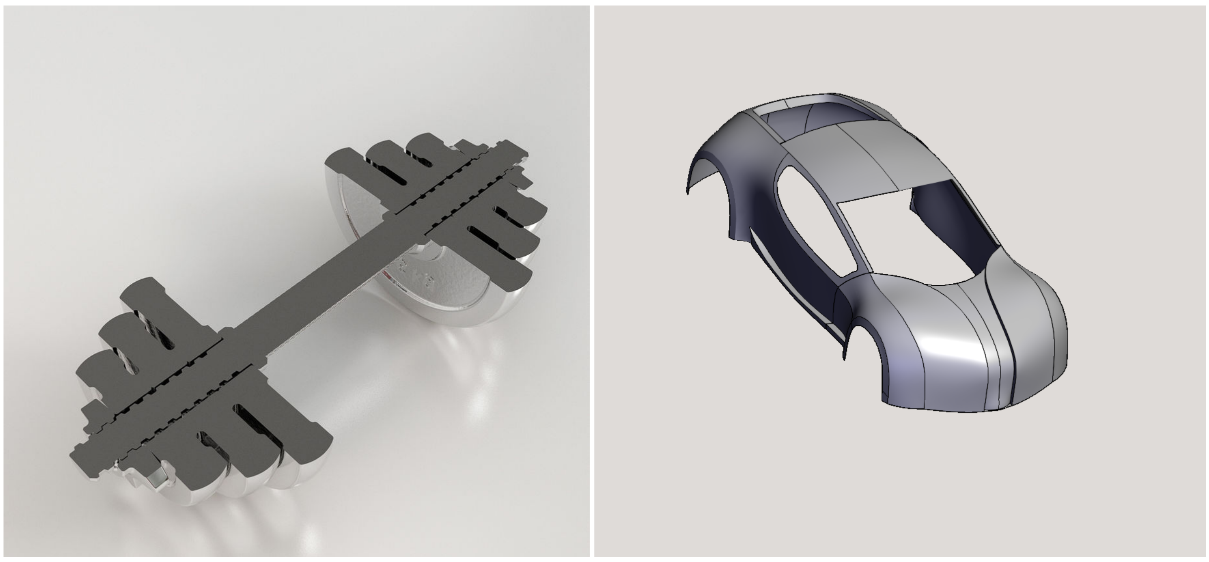


Figure 2.11: (Left) A solid 3d model. (Right) A Shell/Boundary 3d model

The 3D models are devided into two categories. The first one is the solid models, which define the volume of the object they represent. These are more realistic, and it is required more effort to build. Solid models are mostly used for non visual simulations, such as medical and engineering simulations. The second is the shell or boundary models which represent the surface, the boundary of the object, not its volume. These are easier to work with than the solid models. Boundary models are the ones that are used mostly in computer graphics as to make the appearance of an object is directly connected to the exterior of the object, so shell models are the ones used mostly in entertainment industry.

Vertices that define the shape and form polygons compose a 3D model. A polygon is a shape consisted from at least three vertices, a triangle, or four vertices, called a quad. A line which links two vertices is called edge. There are various modeling methods. The most commonly used ones are:

Polygonal modeling: is a modeling process with the main focus on the polygons of the object on the surface. The vertices are connected by lines in order to form a polygonal mesh. The today 3D modeling is greatly associated to textured polygonal models as they are characterized with flexibility and quick representation. However, many polygons can reach curved surfaces as polygons by themselves are planar.

Curve modeling: Surfaces are defined by curves, which are determined by weighted control points. The curve follows but does not necessarily interpolate the points. The higher the weight of a point is, the more that point will pull the curve closer to it.

Digital sculpting: Still a fairly new method of modeling, 3D sculpting has become very popular in the last years. In this case, a software is used that offers tools able to pull, push, smooth, and manipulate in many ways a digital object as if it was made of of a real-life substance such as clay. It can introduce details to surfaces that would otherwise have been difficult or impossible to create using traditional 3D modeling techniques. The downside is that in order to achieve detail with sculpting the models must have a high number of polygons. [13] [14]

2.5.2 3D Rendering

Converting information about 3D objects into a graphics image that can be displayed is known as rendering. It usually requires considerable memory and processing power.

Rendering adds realism to computer graphics by adding three-dimensional qualities such as light, shadows and variations in color and shade. This process is usually performed using 3D computer graphics software. There are many rendering methods that have been developed, each one appropriate for specific applications. There is the non photorealistic rendering, which gives the effect of painting, drawing or cartoons, and the rendering methods aiming to achieve high photorealism.Another categorization is suitability for real time rendering and non real time rendering. Non real time rendering is implemented in non interactive media such as films and video. The rendering process of this kind of content can be very time consuming. That is because non real time rendering has the advantage of very high quality even with limited processing power due to the absence of real time response, which makes the time for the rendering process not considerable. A method suitable for non real time rendering is ray tracing, which simulates the path of a single light ray as it would be absorbed or rejected by various objects in the scene. Real time renderings implemented in interactive media such as games and simulations. The calculations and the display are happening in real time. The primary goal is to achieve an as high as possible degree of photorealism at an acceptable rendering speed. This is 24 frames per second, as that is the minimum the human eye needs to see to successfully create the illusion of movement.

2.6 Game Engines

2.6.1 What is a Game Engine?

A game engine is a software frame-work which is designed for the development of video games. They are used in order to create games for consoles, mobile devices and personal computers. The main features of a game engine include a rendering engine (renderer) for 2D or 3D graphics, a physics engine, a collisiondetection (and collision response) system, scripting, sound, animation, artificial intelligence, memory management, networking, streaming, threading, localization support, scene graph, and may includevideo support for cinematics. The process of game development is often economized, in large part, by reusing the same game engine to create different types of games. Also one game engine can be able to build applications for multiple platforms.

The beauty and power of game engines, is that they speed-up the development process, by providing a suite of visual development tools, reusable software components and simplification of frequently used tools, elements and processes. Game Engines are usually built upon one or multiple rendering application programming interfaces (APIs), such as Direct3D or OpenGL which provide a software abstraction of the graphics processing unit (GPU). These APIs are commonly used to interact and communicate with the GPU,to achieve hardware-accelerated rendering.

Modern game engines are some of the most complex applications written, which is the result of years and years of improvements, experience and development. Nowadays they often feature dozens of finely tuned systems interacting to ensure a precisely controlled user experience. The evolution of game engines has separated concepts like rendering, scripting, artwork, and level design. Nowadays it is common for a typical game development team to have as many artists as actual programmers.

Furthermore, due to the constant growth of the smartphone application market and increasing competition, popular high-end Game Enginesare proving to be a precious tool for developers worldwide, to bring theirideas and games to life, in as many platforms as possible. [15]

2.6.2 Popular Game Engines

In this section, we will take a brief look at 4 of the most popular free gameengines currently available, and explain which is more suitable for this projectand why.

Unity3D:
Unity 3D, initially released on 2005, is a flexible and powerful development platform for creating high quality 2D and3D games. Emphasizing on portability, Unity currently supports over 20 platforms, including PCs, consoles, mobile devices (iOS and Android)and websites. Additionally, many settings can be configured for each platform. As a result,Unity can detect the best variant of graphic settings for the hardware or platform the game is running, thus optimizing performance and sacrificing visual quality if

necessary. Apart from its next-generation graphical capabilities, Unity also comes with an integrated physics engine(nVidias PhysX). Much like Unreal Engine, Unity offers developers an Asset Store to buy re-usable content and assets for use in their project. To sum up, due to its ability to efficiently target multiple platform at once and user-friendly environment, this game engine is an ideal choice for a large portion of developers. [16]

CryEngine:
CryEngine is developed by game developer Crytek and has been used in all of their titles. It is capable of producing stunning, eye-catching graphics and visuals, featuring advanced shader and lightning systems. Because of this, CryEngine clearly targets only powerful PCs and high-end consoles. It comes with VR support and a large amount of advanced visual features, tools, audio/-physics systems and character and animation systems.CryEngine can be downloaded and used for free. [17]

Unreal Engine:
Unreal Engine(UE),initially released on 1998, is a complete suite of game development tools, powering hundreds of games, simulations and visualizations. It is one of the most advanced engines to date, delivering top quality visuals while providing users with a large variety of tools to work with everything they need. Due to its capabilities, efficient design and ease of use it is well-appreciated engine from hobbyists to development studios. It is also available for free. Developers can also port their projects to mobile devices, both iOS and Android. Unreal Engine also works with Virtual Reality. Finally, UE also gives access to its users with to a marketplace, to buy re-usable content and add to their project, speeding the development process. [18]

Amazon Lumberyard:
Amazon Lumberyard is a free game engine developed by Amazon and based on the architecture of CryEngine. Lumberyard has similar capabilities to CryEngine and can be used for production of high

quality games targeting high-end platforms. It is remarkable that the
entire source code can be viewed and changed by the developers to suit their needs. This engine focuses on a fee-based managed system for cloud building and hosting, intended to allow developers to easily develop games that attract "large and vibrant communities of fans, as stated by the company. [19]

2.6.3 Comparison/Choosing the Right Engine

Unreal Engine and Unity are currently ahead of the competition as the two most popular game engines available to the public. They are both capable of providing high-end graphics, a large variety of usable tools and support for multiple platforms without compromising usability and efficiency. It is important to note that these 2 engines offer a large community support, which is also something that has to be considered when choosing an engine. CryEngine and Lumberyard provide us with lots of capabilities as well, however their complicated structure and smaller community excluded them from our consideration.

In conclusion, taking into account the advantages and disadvantages of each engine, Unity proved to be the ideal choice for this project, mainly due to its efficiency and ease of use.

Summary:
In this chapter we provided an insightful review of technologies used in 3D Graphics, from the creation of 3D models, to the rendering of photorealistic scenes. Moreover, we reviewed and compared today's most powerful free game engines available to the public.

Chapter 3

Technological Backround

3.1 Unity3D

3.1.1 Brief Description

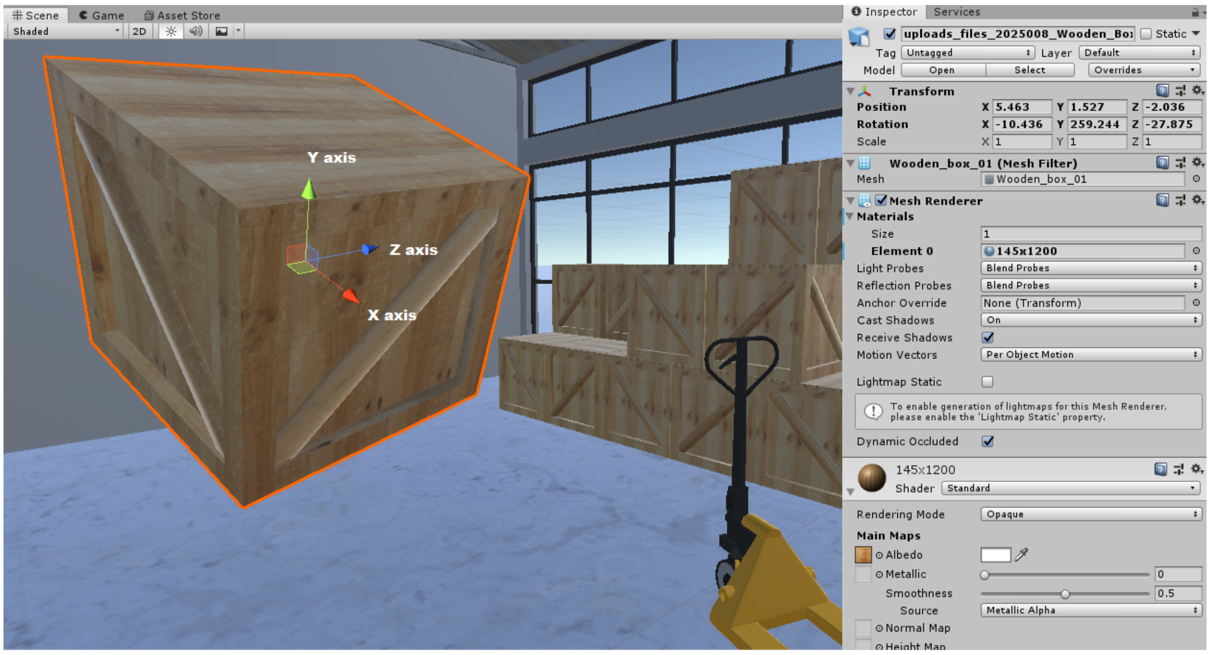
The Android applications developed in this thesis are implemented entirely by using the Unity 3D game engine. Unity 3D is a powerful cross-platform 3D game engine with a user friendly development environment. Unity 3D gives developers the capability to create applications for mobile, desktop, the web, and consoles. Its editor is very handy and suitable for creating user interface menus, doing animations, writing scripts, and organizing projects. Unity is a game engine and its main purpose may be the development of 3D video games, however, it is also suitable to create other kinds of interactive content, such as animations, simulations or 3D visualizations.Unity is a fully integrated development engine that provides functionality to create interactive 3D content. With Unity the developer can assemble assets into scenes and environments, add lighting, audio, special effects, physics and animation, simultaneously play, test and edit the application, and when ready, publish them to a variety of platforms, such as Mac, Windows PC and Linux desktop computers, the Web, Android, iOS, Blackberry 10, Windows Phone 8, Wii U, Sony PS and Xbox. Unitys complete toolset, workspace and rapid productive workows help users to make interactive content faster and with less effort.

3.1.2 Project Structure

Unity is defined by its component based architecture. Its workflow builds around the structure of components. Each component has its own specific job, and can generally accomplish its task or purpose without the help of any outside sources.Each game or application created in Unity is called a project. Each project consists of one or more scenes. Scenes contain the objects of the game. Every scene is considered as a unique level. In each scene, the user can position the 3D models,construct the environment and essentially design most of the functionality. Every object placed in a scene is considered a GameObject. GameObject consist of one or more Components. Components are Unity's fundamental elements, which are used to define properties, behavior and characteristics of a GameObject. The user can add a wide variety of components in a GameObject to achieve the desired functionality.

3.1.3 Scene Window

The Scene window is where the developer constructs the 3D/2D environment of his/her application. It can be used to select, adjust, scale and position the characters of a game, cameras, buildings, lights, and all other types of Game Objects. The scene Gizmo is located at the upper-right corner of the scene. It displays the current orientation the Scene-View Camera has, and makes the user able to modify the viewing angle and projection mode with ease. In order to adjust the position, size or the orientation of a GameObject, the user can use the four Transform tools in the toolbar. Each of them has a corresponding Gizmo that takes place around the selected object in the scene. To alter the Transform component of the GameObject, the user can use the mouse as an input method to manipulate the Gizmo axis, or type the appropriate values directly into the number input-fields of the Transform component in the Inspector. This project consists of five scenes. The first scene concerns the functionality of the controllers and their User's Interface which is a 2D implementation. The rest of the scenes build the 3D environments of the two demos. Each demo has two scenes, the first implements the game in normal mode for 2D smartphone screens while the second one concerns the VR mode of the games.



```
uploads_files_2025008_Wooden_Bo  Static
Tag  Untagged        Layer  Default
Model  Open  Select  Overrides
Transform
Position   X 5.463    Y 1.527    Z -2.036
Rotation   X -10.436  Y 259.244  Z -27.875
Scale      X 1        Y 1        Z 1
Wooden_box_01 (Mesh Filter)
Mesh                Wooden_box_01
Mesh Renderer
Materials
  Size              1
  Element 0         145x1200
Light Probes        Blend Probes
Reflection Probes   Blend Probes
Anchor Override     None (Transform)
Cast Shadows        On
Receive Shadows     ✓
Motion Vectors      Per Object Motion

Lightmap Static     ☐
  To enable generation of lightmaps for this Mesh Renderer,
  please enable the 'Lightmap Static' property.
Dynamic Occluded    ✓
145x1200
Shader  Standard
Rendering Mode      Opaque
Main Maps
Albedo
Metallic                           0
  Smoothness                       0.5
  Source            Metallic Alpha
Normal Map
Height Map
```

Figure 3.1: Scene Window preview.

3.1.4 Canvas

The Canvas is the area that all UI elements are inside. Particularly, it is a Game Object which contains a Canvas component and all UI elements must be children of such a Canvas. This GameObject is required in order to create UI elements, for example images or texts. Creating a new UI element automatically creates a Canvas, if there isn't any already, and this element is a child to this Canvas. Canvas has a rectangle 2D area in the Scene View, a fact that makes it easy to position UI elements without needing to have the Game View visible at all times. The User's Interface of the controllers is built entirely by using the Canvas of the scene. [20]

3.1.5 Inspector

Projects in the Unity Editor are using a lot of GameObjects which contain scripts, sounds, meshes, and other graphical elements such as lights. The Inspector window is located at the right side of the editor and provides us with detailed information about the selected

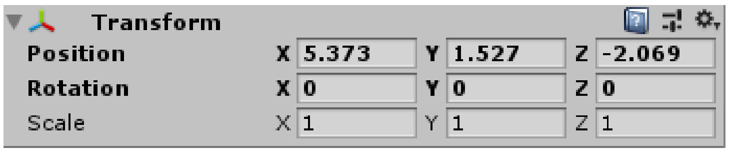
GameObject, including all attached components and their properties, and allows the developer to modify its functionality in the Scene. The Inspector is used in order to view and edit the properties of almost every element of the project, including physical game items such as GameObjects, assets, and materials, as well as in-editor settings and preferences. When a GameObject is selected in either the Hierarchy or Scene view, the Inspector shows the properties of all components andmaterials of that GameObject. Actually, the Inspector can be used to edit the settings of these components and materials. In this project the Inspector was an essential tool as most of the application components had to be modified appropriately. [21]

3.1.6 Components

Some of the most commonly used components in Unity are presented next:

Transform:

Every GameObject contains a Transform component which is created automatically when the object is created. It is an essential component and it can not be removed. It's obvious that it is one of the most important components and most frequently accessed Component. It defines a GameObjects position, rotation, and scale in the game world based on the x, y, z coordinate system. These parameters are initialized by hand and/or can modified in run-time by script to make objects move,rotate and more. It is important to note that when scripting functionality such as movement, Unity considers the Z axis as forward/backwards, Y axis as up/down and X axis as left/right. During the implementation of this thesis a lot of adjustments were needed concerning the position, size or rotation of the game components. Most of them were managed through the Transform component. [22]

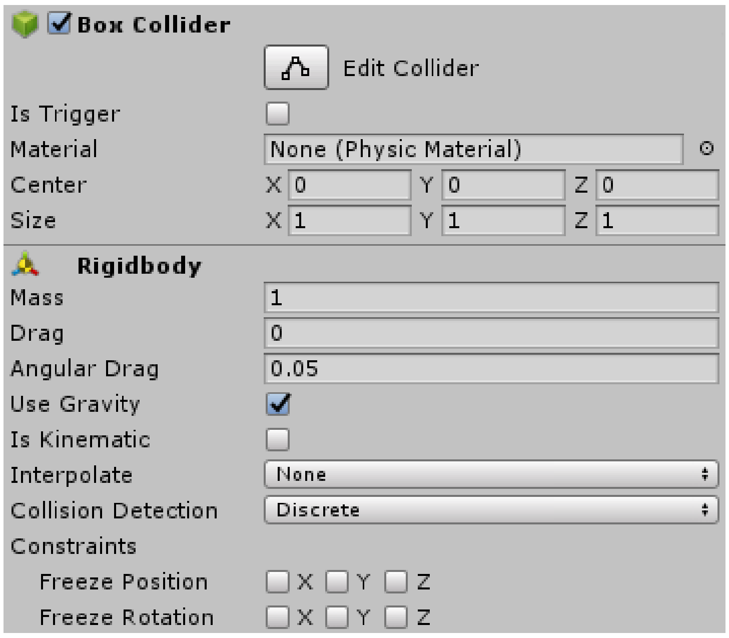


Transform

Position	X	5.373	Y	1.527	Z	-2.069
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

Physics:

Physics components allow the developer to give objects a realistic motion and reaction to collisions by simulating physics laws. Unity has NVIDIA PhysX physics engine built-in. A physics engine is a computer software that provides an approximate simulation of physical systems. This allows the game objects to have a realistic behavior. A rigidbody component makes an object able to be affected by gravity, linear and angular forces and collide with other objects. There is also a variety of collider components(mesh, box, sphere, wheel collider) which surround the shape of an object for the purposes of detecting physical collisions. [23]



Box Collider [✓]

Edit Collider

Is Trigger	☐
Material	None (Physic Material)
Center	X 0 Y 0 Z 0
Size	X 1 Y 1 Z 1

Rigidbody

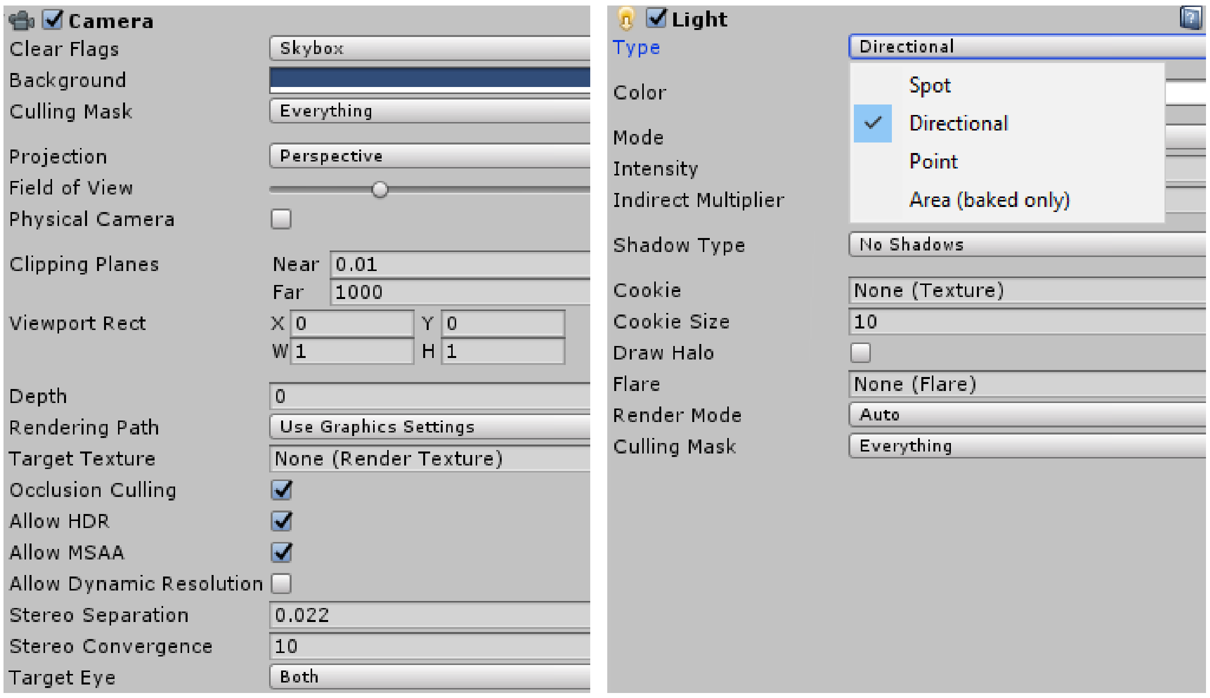
Mass	1
Drag	0
Angular Drag	0.05
Use Gravity	✓
Is Kinematic	☐
Interpolate	None
Collision Detection	Discrete
Constraints	
Freeze Position	☐ X ☐ Y ☐ Z
Freeze Rotation	☐ X ☐ Y ☐ Z

Figure 3.2: Physics components preview.

Mesh:
3D meshes are the main graphic object primitive of Unity. Various components exist in Unity game engine to render meshes. Mesh renderer and mesh filter are used the most. They are both used to display an object on the screen. The mesh filter is responsible of taking a mesh from the project asset folder and then the mesh renderer takes action. It takes the mesh from the mesh filter and renders the object on the screen. Furthermore, the mesh renderer takes the geometry from the mesh filter and renders it at the position defined by the object's transform component. When importing mesh assets, Unity automatically creates a mesh filter along with a mesh renderer. Another component is the

text mesh which generates 3D geometry in order to display text strings. [24]

Rendering:
In this category we can find components which are responsible for rendering in-game and user interface elements, as well as lighting and special effects. The camera component is practically the user's eyes in the game. It is an essential component as it is used to capture and display the virtual world to the player. It can be customized and manipulated to fulfill the requirements of the application. The GUI Texture and GUI Text components are a useful and easy way of creating user interface elements such as buttons, on-screen information panels, decorations as well as displaying text on the screen. An other very important rendering component of a game is the light as it brings a sense of realism. Lights can be used to illuminate the scenes and objects, to simulate the sun, flashlights, or explosions just to name a few.



✓ Camera

Clear Flags	Skybox
Background	🟦
Culling Mask	Everything
Projection	Perspective
Field of View	
Physical Camera	☐
Clipping Planes	Near 0.01
	Far 1000
Viewport Rect	X 0 Y 0
	W 1 H 1
Depth	0
Rendering Path	Use Graphics Settings
Target Texture	None (Render Texture)
Occlusion Culling	✓
Allow HDR	✓
Allow MSAA	✓
Allow Dynamic Resolution	☐
Stereo Separation	0.022
Stereo Convergence	10
Target Eye	Both

✓ Light

Type	Directional
Color	
Mode	
Intensity	
Indirect Multiplier	
Shadow Type	No Shadows
Cookie	None (Texture)
Cookie Size	10
Draw Halo	☐
Flare	None (Flare)
Render Mode	Auto
Culling Mask	Everything

Figure 3.3: Rendering components preview.

Materials and Shaders:
Materials and shaders are crucial components that are categorized in the asset component group. They work together and they both play an essential part in defining how the object is displayed. Materials are used along with mesh renderers and other rendering components. The material properties are determined by the shader the material uses. A shader is a specialized kind of graphical program that determines how texture and lighting information are combined to generate the pixels of the rendered object on screen. In other words, it tells the graphics hardware how to render surfaces.The user can select which shader each material will use. Specifically, a material defines which texture and color to use for rendering, whereas the shader defines the method to render an object. Every object used in the applications of this thesis contains a material. The source of the materials used are either a normal colour or an image downloaded from the internet, so the environment looks more realistic. For example, the material of a wall in the game has as source an image that depicts a wall built from stones. [25] [26]

Audio:
These components are used to implement sound. The most important component here, is the Audio Source component, which as the name suggests, plays a sound file at the location of the game object it is attached to. The developer can set parameters such as sound volume, pitch and change the sound file to be played at any time. These parameters can also be changed by script during run-time. [27]

Script Component:
The script component simply attaches a script onto a game object. Scripts are attached to objects in order to define their behavior during the game-play process and trigger effects upon specified conditions. More about scripts in the following section.

3.1.7 Scripting

In order to make the applications of this thesis and their components functional the use of Scripts was essential. Scripting defines the entire behavior of the application, thus it is an important feature of Unity game engine. Every application must have at least one script in order to respond to user's interaction. They are files that contain lines of code (C# in this project) which aim at the execution of specific operations. Scripts can be used for several reasons such as: to create graphical effects, to control physical behavior of objects or characters, to trigger effects upon specified conditions. The behavior of GameObjects is controlled by the Components that are attached to them. Scripts are also attached on GameObjects as a component and determine their behavior to an extent. Although Unity's built-in Components can be very versatile, the programmer will soon find what he needs to go beyond what they can provide to implement custom game play features.

```
        vignette.blur = (1-health) * 2 + [illegible]
        vignette.blurDistance = (1-health) * 2 + [illegible]
        vignette.chromaticAberration = heatEffect * [illegible]
    }


    void OnTriggerStay(Collider c)
    {
        var fire = c.GetComponent<Fire>();
        if (fire && fire.alive)
        {
            float dist = 1-((transform.position - [illegible]
            NearHeat(dist);
        }


        var smoke = c.GetComponent<[illegible]
        if (smoke && smoke.GetComponent<[illegible]
        {
            float dist = 1-((transform.position - [illegible]
            NearSmoke(dist);
        }
    }

    void OnCollisionEnter(Collision c)
    {
        var healthBox = c.gameObject.[illegible]
        if (healthBox)
        {
```

Figure 3.4: Script preview.

Unity allows the developer to create custom Components using scripts. They can be used in order to trigger game events, modify Component properties over time and respond to user input in any way he could probably want to. Unity supports two programming languages natively, the C#, an object oriented programming language similar to Java or C++ and the UnityScript, a language designed specifically for use with Unity and modelled after JavaScript. The scripts can be written and edited in MonoDevelop, which is an integrated development environment (IDE) within Unity or in any other IDE like Visual Studio.

An IDE combines a text editor with additional features for debugging, auto-complete and other project management tasks. A script makes its connection with the internal workings of Unity by implementing a class which derives from the built-in class called MonoBehavior. The reader can think of a class as a kind of blueprint for creating a new Component type that can be attached to GameObjects. When a script component is added to a GameObject, it creates a new instance of the object defined by the blueprint. The name of the class is taken from the name that the programmers applied when the script file was created. The class name and the script file name must be the same to enable the script component and make it able to be attached to a GameObject.

Scripts are also used to easily access and modify components in order to achieve the desired behavior and functionality during the game-play process of the game. When a script is created, there are two functions automatically declared in it, the Start() function and the Update() function. The Update function contains code that must be executed in every frame such as movement related code, triggering actions, response to user's input and anything that needs to be handled over time during gameplay. Update function might need some data which must be declared when the application starts. The Start function will be called by Unity when a script is enabled and will be called only once. The Start function is the ideal place where initialization occurs. It is used to initialize an object position, state and properties or load other scripts and GameObjects for later use.

The Update functions runs in a loop and is executed in every frame but a script may not contain one. Unity passes control to a script intermittently by calling functions that are declared within it. Once a function has finished its "job", Unity take over the control again. These functions are activated when events are occured during the gameplay. Unity uses a naming scheme to identify which function to call for a particular event. Apart from the Update and the Start function, many more event functions are available in Unity. Some of the most common events are explained below:

Regular Update Events:
These events can make changes to position, state and behavior of objects in the game just before each frame is rendered. Such a code is often written in the Update function. It is called before the frame is rendered as well as before animations are calculated. For

physics update, like adding force to a GameObject, the best option is to place the code in the FixedUpdate function which updates more frequently than the Update function. Sometimes the best place to write code is the LateUpdate function in order to be able to make additional changes at a point after the Update and FixedUpdate functions have finished their "job" and after all animations have been calculated.

Initialization Events:
It is often useful to be able to call initialization code in advance of any updates that occur during game play. The Start function is called before the first frame or physics update on an object. The Awake function is called once when the scene starts but it is called for each object in the scene. Note that all the Awake functions will have finished before the first Start is called.This means that code in a Start function can make use of other initialization previously carried out in the awake phase.

GUI Events:
Unity has a system for rendering GUI controls over the main action in the scene and responding to clicks on these controls. This code is handled some what differently from the normal frame update and so it should be placed in the OnGUI function, which will be called periodically. For example, some OnMouseXXX event functions (e.g OnMouse-Down) are available to allow a script to react when the user actions with the computer mouse. In the case the mouse button is pressed while the pointer is over a particular object then an OnMouseDown function in that object's script will be called if it exists.

Physic Events:
That kind of events were essential for the two demos of this thesis. More specifically, their use concerns the interaction of the user with some objects of the scene as he collides with them. The physics engine will report collisions against an object by calling event functions on that object's script. Some of them are the OnCollisionEnter function, On-CollisionStay and OnCollisionExit. They will be called as contact is made, held and broken. The corresponding OnTriggerEnter, OnTriggerStay and On-TriggerExit functions will be called when the object's collider is configured asa Trigger (a collider that simply detects when something enters it rather than reacting physically). These functions may be called several times in succession if more than one contact is detected during

the physics update process. A parameter is passed to the function giving the required information of the collision happened (position, identity of the incoming object, etc.).

Except for the functions that Unity provides, the developer can create his/her own functions in order to control or determine the behavior of a GameObject, change the properties of a component and generally manages the behavior of the application. These function will be executed after being called inside a Unity event function. The most commonly used functions were presented briefly above, as well as the concept of how they are used. Each component property corresponds to a script variable and the scripts can access not only the components of the GameObjects they are attached to, but also other GameObjects and their components. [28] [29]

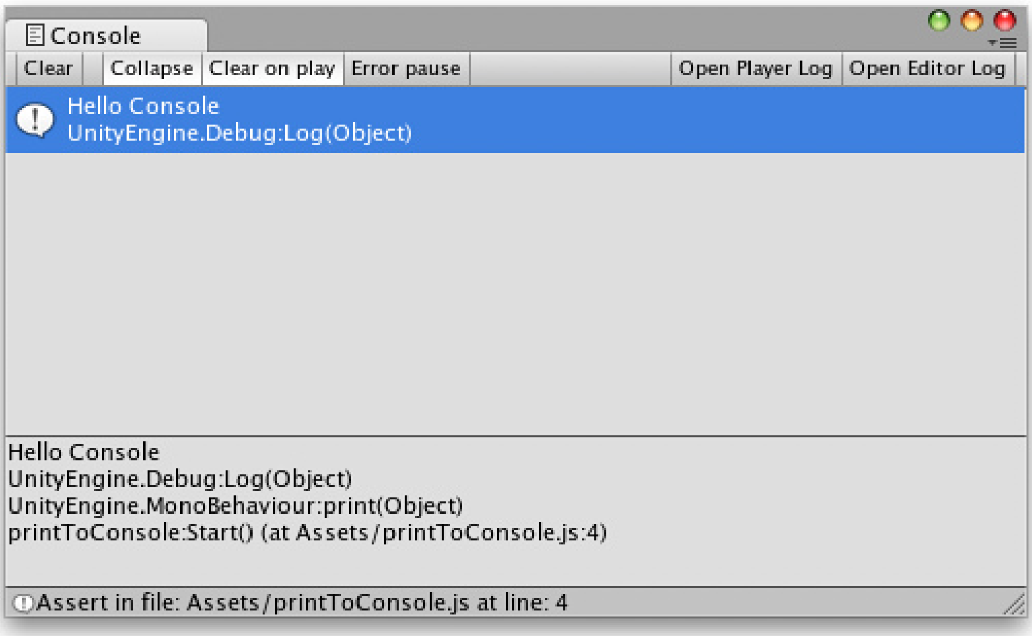
3.1.8 Game Window

The Game window is rendered from the Camera of the scene. It is representative of the final, published game. It is required for the user to use one or more Cameras to control what the player actually sees when they are playing the game. The game window was an essential asset for this project as it helped in testing and debugging the code before the actual applications were built and published.

3.1.9 Console

An other asset ,offered by Unity, which helped in debugging and optimizing the applications of this thesis is the Console window. The Console Window displays errors, warnings and other messages generated by Unity. In order to debug the code, the developer can also display his own messages in the Console using the implemented functions of Unity (Debug.Log, Debug.Log Warning and Debug.Log Error). The toolbar of the Console offers a variety of options that affect how messages are displayed. The Clear button removes any messages generated from user's code but retains compiler errors. The developer can also choose if the console will be cleared automatically whenever he runs the game by enabling the Clear on Play option.

There is also the opportunity to change the way messages are shown and updated in the console. The Collapse option shows only the first instance of an error message that keeps recurring. This can be very useful for runtime errors correction which are sometimes generated identically on every frame update. When Debug.Log.Error is called from a script then playback will be paused. This will happen when the Error Pause option is enabled. Finally, some details are recorded by Unity but may not be shown in the console. The Open Player Log and Open Editor Log items on the console tab menu can access and display these additional details. [30]



```
Hello Console
UnityEngine.Debug:Log(Object)

Hello Console
UnityEngine.Debug:Log(Object)
UnityEngine.MonoBehaviour:print(Object)
printToConsole:Start() (at Assets/printToConsole.js:4)

Assert in file: Assets/printToConsole.js at line: 4
```

Figure 3.5: Unity's Console Window preview.

3.2 Android Platform and GoogleVR for Unity

As mentioned before, Unity is a 3D game engine which can build and publish applications for many software platforms. In order to build this project the Android platform provided by Unity had to be used. This platform doesn't differ a lot with the original but it's use is essential for the building process of the mobile applications. GoogleVR was an other necessary software for this thesis. GoogleVR is a Software Development Kit (SDK) witch holds a set of tools and programs, including libraries, documentation, code samples, processes, and guides used to create VR applications.

Chapter 4

User's View

4.1 Controllers

4.1.1 User Interface (UI)

In the field of human-computer interaction, a user interface is the environment where interactions between humans and machines occur. The goal of this interaction is to allow control of the machine from the user's side. The machine feeds back information that help in the users' decision-making process. [31]

In this thesis, a UI environment is required so the user is able to manipulate the controllers and hence to interact in the VR application. This environment was created by using the Unity UI (Unity User Interface). Unity UI is a toolkit for developing user interfaces for any kind of applications. It is a GameObject based system that uses Components and the Game View in order to adjust (position, size) and style the interfaces. Some of the components that are often used are buttons, texts, images, sliders, input fields etc. So using the client's implementation code as a core, a 2D environment was created around it which allows the user to "communicate" with the application and take advantage of the features it offers. This is achieved by presenting to the user a series of different menus and interface content, each of which has its own purpose. Of course the user can browse these menus freely whenever he wants, back and forth.

More specifically when the application starts, the user is being asked to select the controller (left hand or right hand) which is being implemented by the specific application. As mentioned above, both controllers are implemented by the same application so there is a need to define which controller is being implemented each time. After this definition is made, the appropriate initializations are made within the application so that it executes as the selected controller. The user is then asked to choose between creating a new profile or uploading an existing one in case he wants to play the same game again.

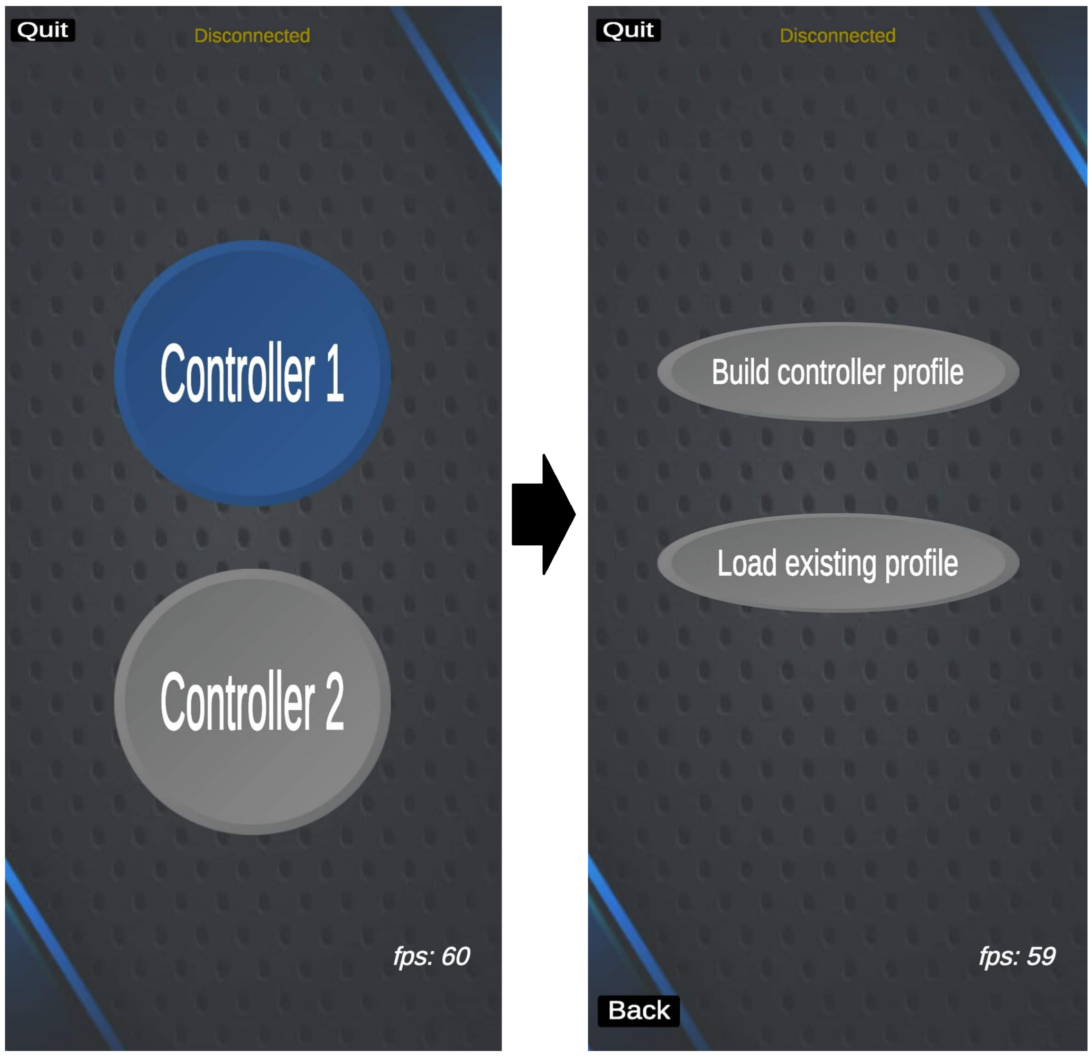


Figure 4.1: Controllers Menu preview.

Depending on the user's choice, in the first case a menu appears in which he chooses which functions of the controller he wants to use while in the second case the application takes the user to a selection menu of the already saved profiles. As shown below the user can save up to three profiles. The options given for a new profile build are the use of gyroscope sensor, accelerometer sensor, navigation joystick and two virtual buttons.

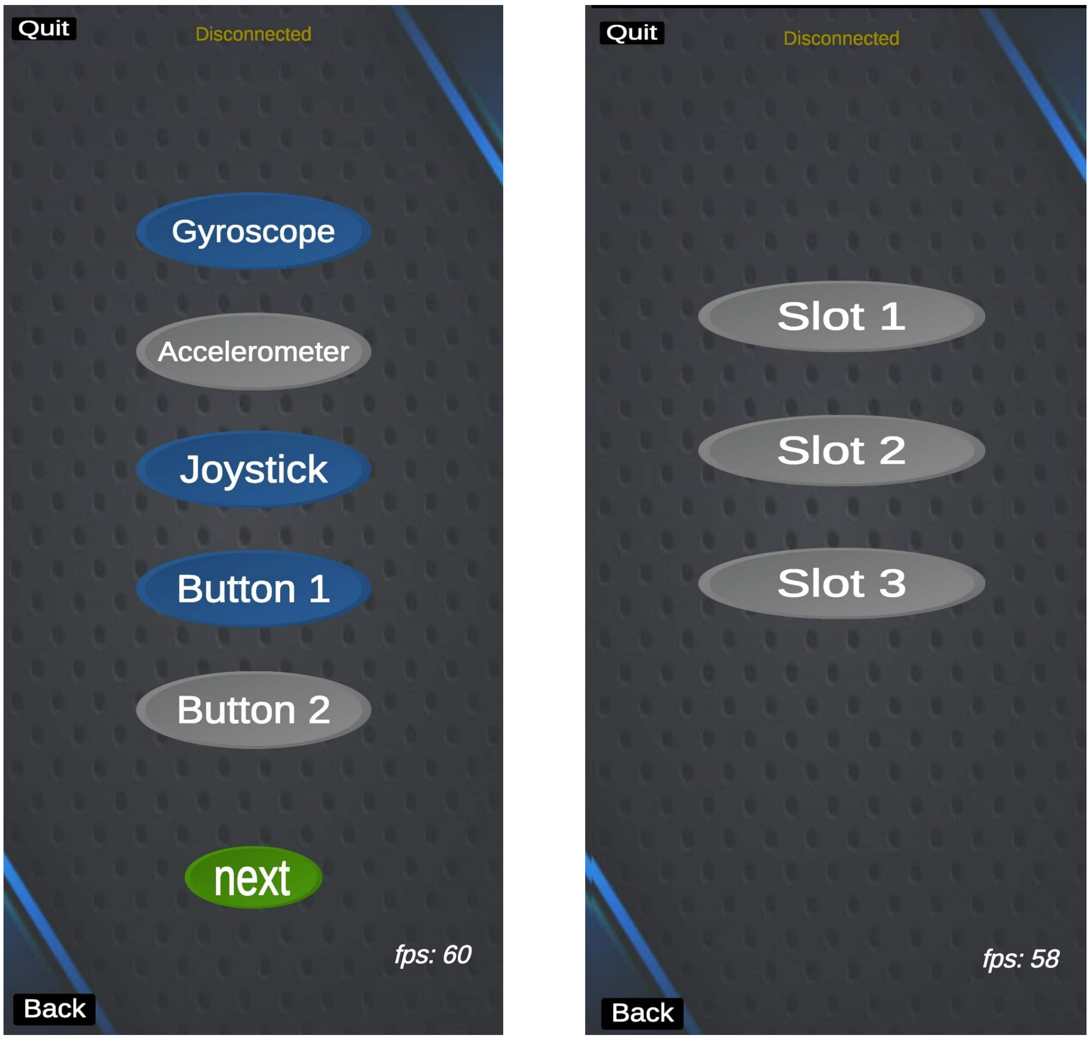


Figure 4.2: (Left) Building new profile. (Rifht) Loading an existing profile.

The next step is to adjust the objects selected by the user, on the screen of the smartphone. The term adjustment refers both to the final position of the objects, for example the navigation joystick, and their size. This is a necessary process because both the smartphones on the market and the hands of random users have different sizes. The user should feel comfortable when using the controller. The customization of the objects therefore comes to alleviate this problem and make the final result of the controller easy to use. When the customization process is complete, the user can save the profile he has created, for future use, or proceed with the connection process to the server. The connection interface presents the user an input field, in which the ip address of the server must be indicated, and a Play button which deactivates the adjustment ability of the objects, activates their functions and calls the function which is responsible for the connection of the controller to the server.

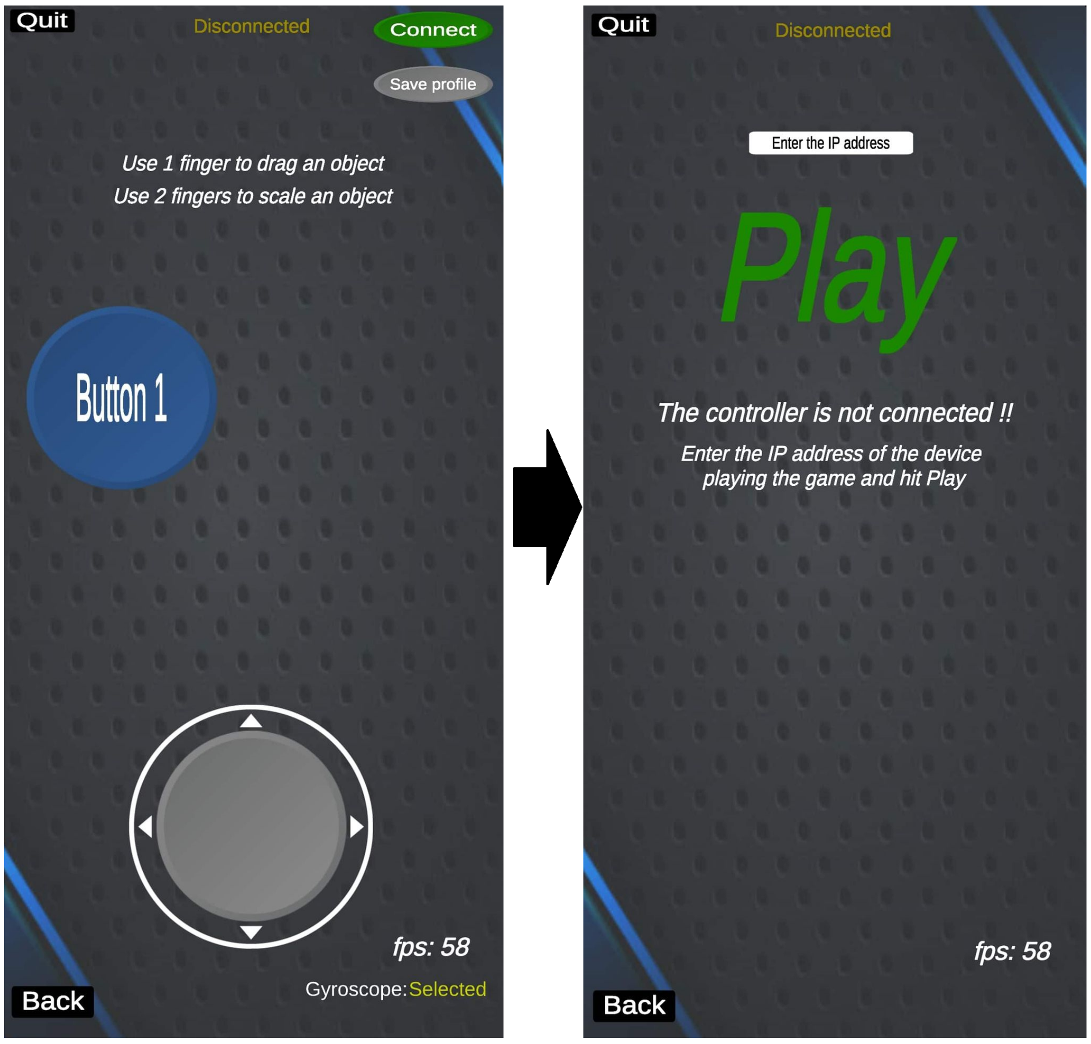

Quit
Disconnected
Connect
Save profile
Use 1 finger to drag an object
Use 2 fingers to scale an object
Button 1
fps: 58
Gyroscope:Selected
Back
Quit
Disconnected
Enter the IP address
Play
The controller is not connected !!
Enter the IP address of the device playing the game and hit Play
fps: 58
Back

Figure 4.3: (Left) Layout customization. (Right) Connect and Play.

During the whole process of adjusting the controller the user can freely go back in case of decision changing. Also useful messages for the user are displayed related with the connection status of the controller as well as whether the sensors have been selected and are supported by the device on which the application is running. There are also tips for the adjustment of the objects on the screen and for the connection process of the controller. As soon as the connection is established, the layout of the controller locks and it can not be modified for functionality reasons. The final state of the application is shown below.

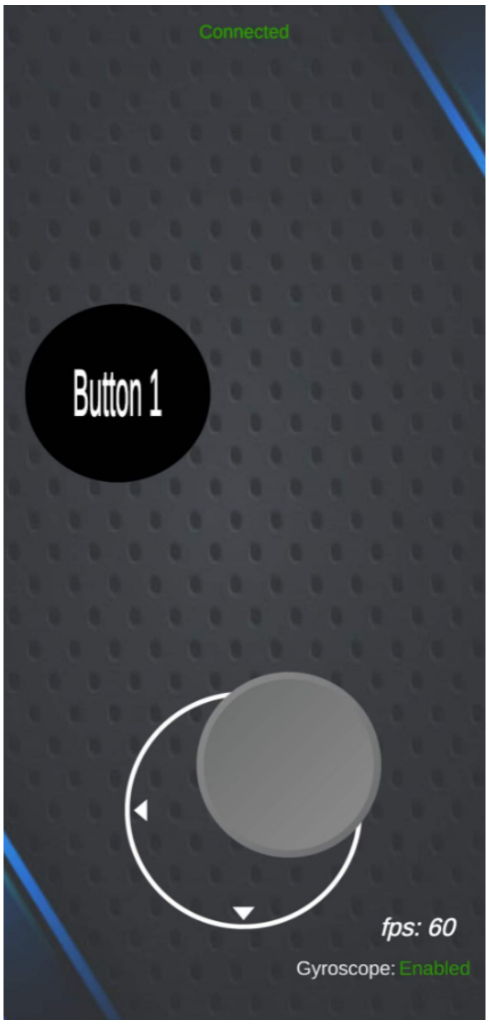


Figure 4.4: Controller's final state.

4.2 Demo Games

4.2.1 3D Environment

Within the framework of this thesis, two 3D games were designed which are based on the Android mobile platform. In the first demo the user is placed in an industrial warehouse. His task is to transport a number of packages from a workbench to another. In order to do that, a virtual hand is used. Its motion is controlled by the orientation of the smartphone used as controller. The second demo is a simple form of a combat simulator and requires the use of two controllers. The user finds his self in a castle equipped with a sword and a shield ready to defend it. In this case, two virtual hands are appeared on the user's eyes. The right is holds the sword while the other holds the shield. Both of them are moving according to the orientation of the smartphones the user holds. In both games the user has a realistic view of 360 degrees of the virtual environment and is able to navigate in it.

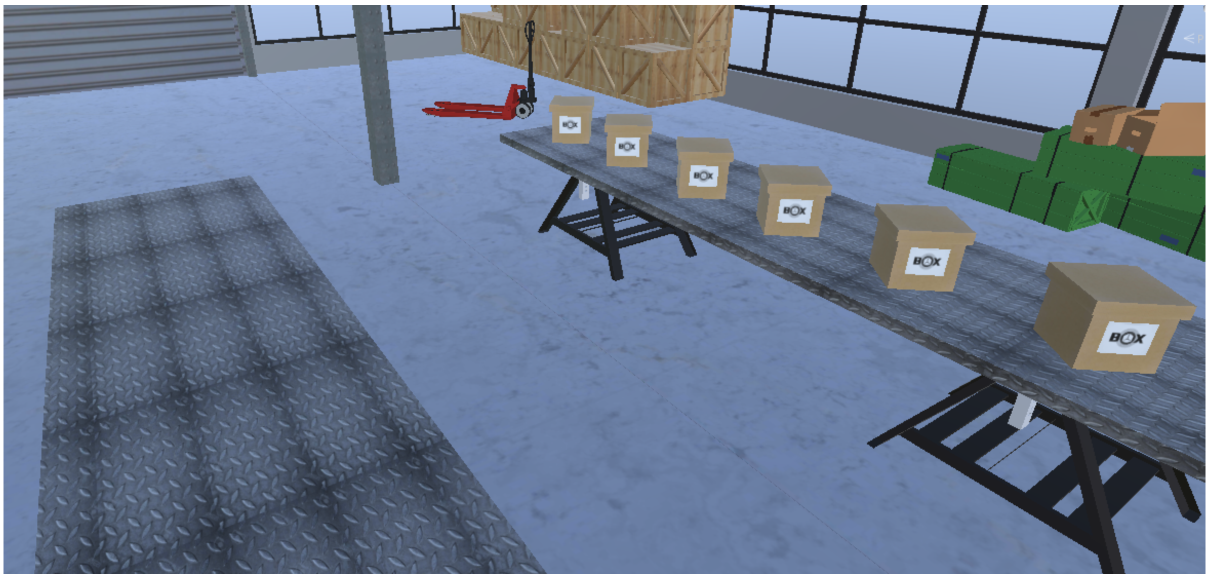


Figure 4.5: First Demo 3D Environment.

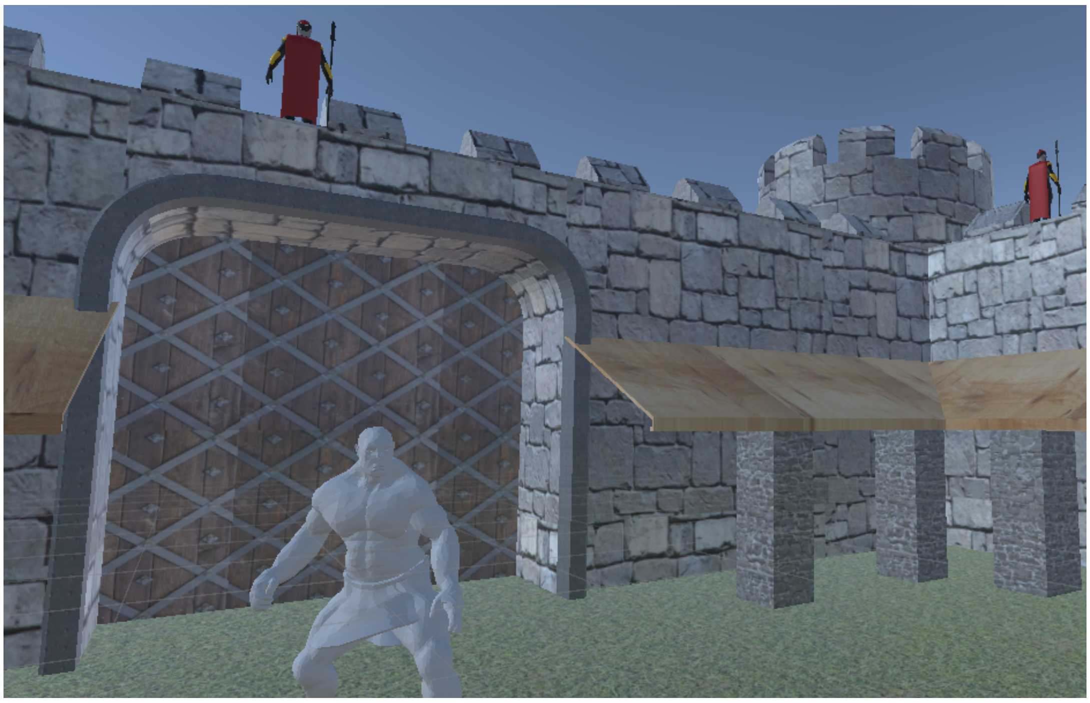


Figure 4.6: Second Demo 3D Environment.

Chapter 5

Implementation

5.1 Unity Scripting API

Generally an application programming interface (API) is defined as a computing interface that defines interactions between multiple software intermediaries. It defines the different types of calls or requests that can be made, the way to make them, the data formats which should be used, the conventions to follow, etc. Also it provides capabilities for extending existing functionality in many ways and to varying degrees. [32]

Unity Scripting API is a collection of namespaces which allow us to work with the various features of Unity game engine and extend Unity editor. A namespase is a collection of prebuilt classes, interfaces, structures, enumerations, events, delegates etc meant to handle a specefic task. The basic namespaces used in this project are:

UnityEngine:
This namespace is the main port of call for most users as it allows us to work with basic and necessary features of Unity game engine like Physics, Particles, Animations, User Interface, Rendering, Audio, etc.

System:
This namespace allows us to create and use most commonly data structures like List, Stack, Queue, Dictionary, etc. [33]

5.2 Connection Software

As mentioned in previous chapters, two applications have been developed in the context of this thesis. The first implements the controllers and the other is the Virtual Reality application, for example a game or an educational application. The user interacts with the virtual environment through the controllers which holds with his hands. This creates the need to establish a connection between the devices that run the applications so that their communication and the transmission of the necessary data is possible.

Nowadays, in almost every home there is a network router which offers wireless connection (WIFI). Modern mobile phones also have wifi antennas so that they can connect to a router, therefore to the internet. However, a network can operate locally. It does not need internet access and the devices that are connected to it communicate with each other. To successfully connect a device to a network, the device must receive an IP address. An Internet Protocol address is a numerical label assigned to each device connected to a network.

The applications developed utilize the above technologies in order to connect and communicate with each other. More specifically, they use the antenna of the home router as an access point without requiring the router to provide an internet connection. As mentioned above, connecting to a network requires an IP address. The application that implements the controllers uses the IP address of the device to which the VR application is running in order to connect to it and start transmitting the necessary data.

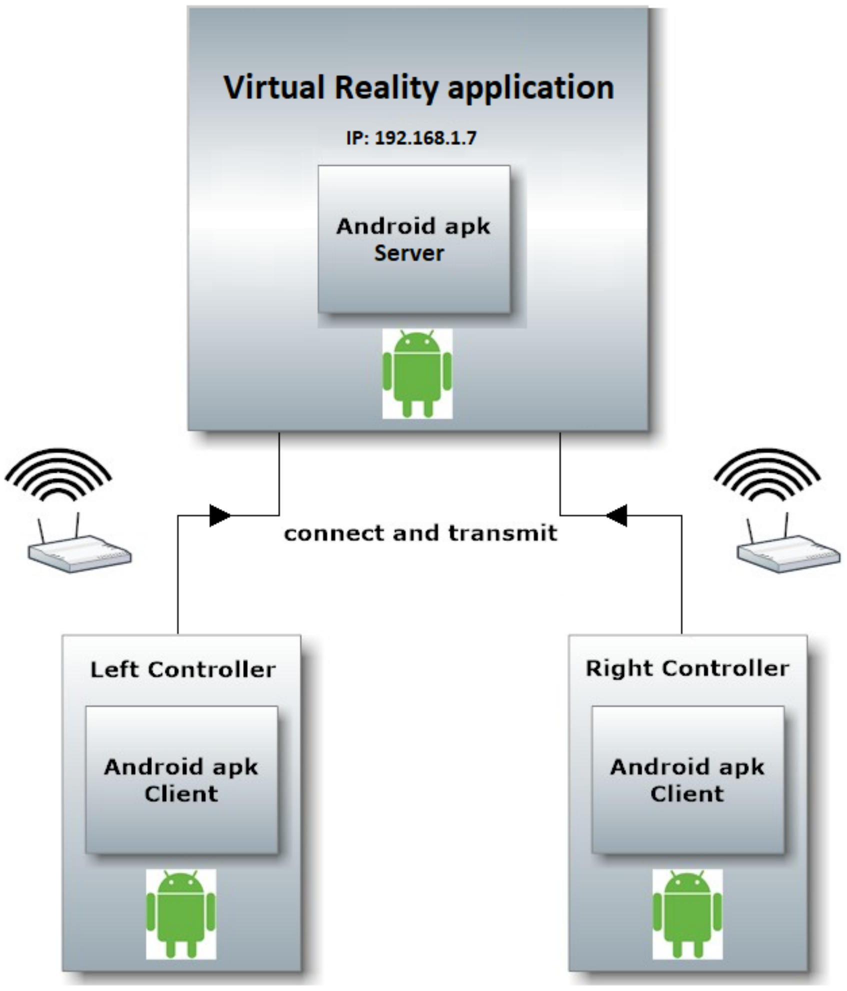


Figure 5.1: Applications connection diagram.

5.2.1 Server

The first step of the connection process was the implementation of the server. In order to achieve that an extension of UnityEngine namespace was used, the UnityEngine.Networking. This extension includes and gives access to classes which are related to network setup and functionality.

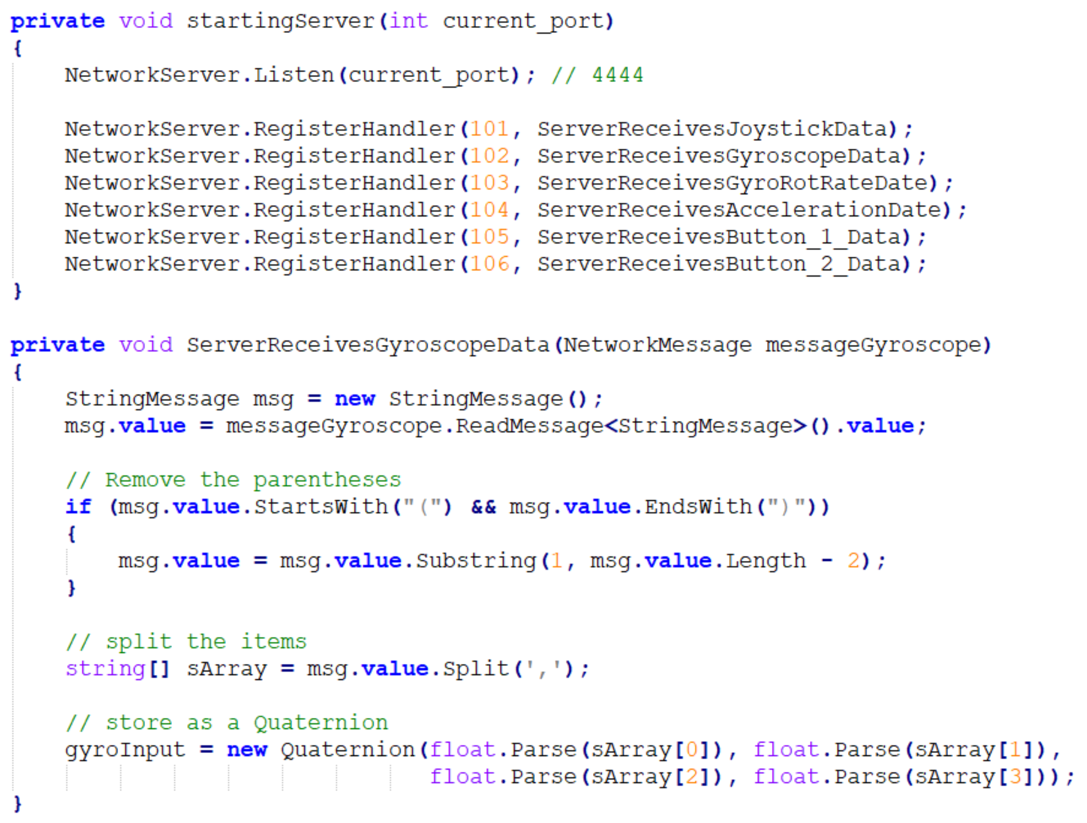
More specifically, the class that starts the server is the NerworkServer class provided by Unity's namespace. This class uses a NetworkServerSimple class for basic network functionality and adds more game-like functionality. The NetworkServerSimple is a basic server class that doesnt contain the "game" related functionality that the NetworkServer class has. It doesnt have features like scene management, spawning, player objects, observers, or static interface.. It is only a server which is able to listen on a port, manages connections, and handles messages.

The first thing to be defined was the port on which the server listens. In our case that port is the port 4444. NetworkServer class can handle remote connections from clients, using a NetworkServerSimple instance. Through those connections messages are being transmitted, so the server must be able to handle them. In order to do that, a dictionary for the transmitted messages had to be defined . The function used to define the dictionary is the RegisterHandler(id, exampleFunction) function.

The dictionary makes it possible for the server to "understand" and separate the different types of data he receives. In our case, the data being transmitted vary from a simple integer to a pack of four floats separated with a blank space. Due to this data type variety the easiest way to transmit them was to convert them to string type before the departure of the message. Server receives the string message, converts the data to their original type and then stores them into variables which are ready to be used in either way.

The function that registers the different types of data in the dictionary needs a unique id number being matched with a function that receives one specific type of data.For example, the receipt of a simple integer happens in the ReceiveInteger(NetworkMessage msg) function and the id for simple integers is 100. Then, the format of the register function will be RegisterHandler(100, ReceiveInteger).

A small part of the server's implementation code is being shown below. The first function starts the server and registers the different data types in the dictionary. The second function receives the data of a smartphone gyroscope sensor. The format in which the data arrive is: (0.0, 1.1, 2.2, 3.3). The function firstly reads the value of the string message, then removes the parentheses, it splits the message value with the comma as a reference point and stores the actual data in a string array. In the end, it converts each single value of the array from string to float and stores them in a variable.



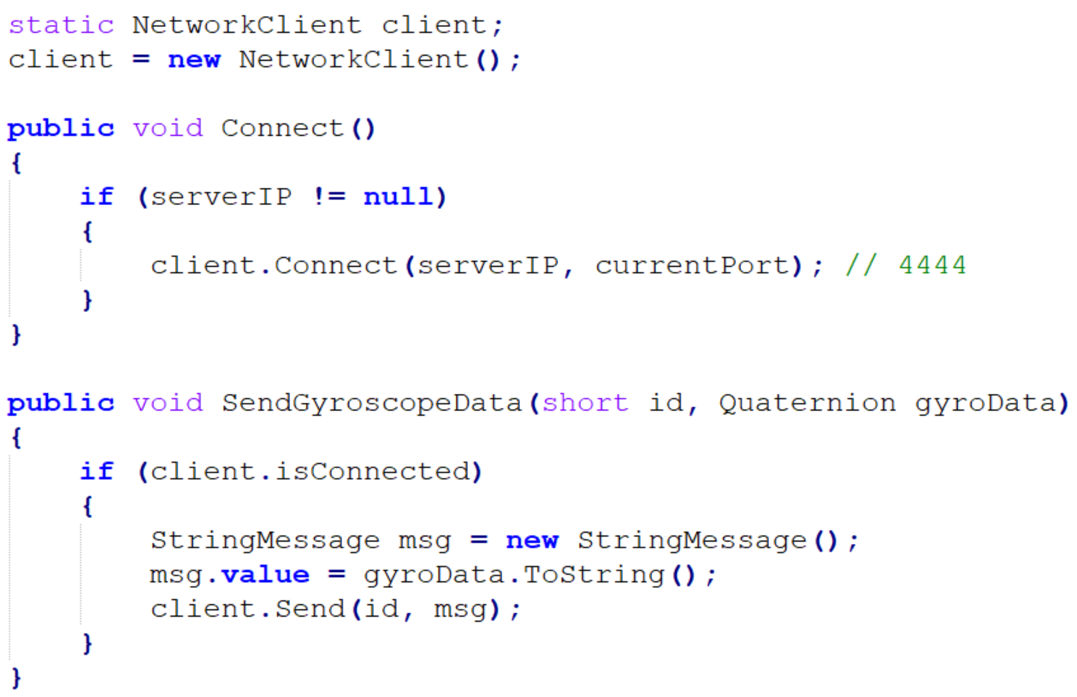
```
private void startingServer(int current_port)
{
    NetworkServer.Listen(current_port); // 4444

    NetworkServer.RegisterHandler(101, ServerReceivesJoystickData);
    NetworkServer.RegisterHandler(102, ServerReceivesGyroscopeData);
    NetworkServer.RegisterHandler(103, ServerReceivesGyroRotRateDate);
    NetworkServer.RegisterHandler(104, ServerReceivesAccelerationDate);
    NetworkServer.RegisterHandler(105, ServerReceivesButton_1_Data);
    NetworkServer.RegisterHandler(106, ServerReceivesButton_2_Data);
}

private void ServerReceivesGyroscopeData(NetworkMessage messageGyroscope)
{
    StringMessage msg = new StringMessage();
    msg.value = messageGyroscope.ReadMessage<StringMessage>().value;

    // Remove the parentheses
    if (msg.value.StartsWith("(") && msg.value.EndsWith(")"))
    {
        msg.value = msg.value.Substring(1, msg.value.Length - 2);
    }

    // split the items
    string[] sArray = msg.value.Split(',');

    // store as a Quaternion
    gyroInput = new Quaternion(float.Parse(sArray[0]), float.Parse(sArray[1]),
                               float.Parse(sArray[2]), float.Parse(sArray[3]));
}
```

Figure 5.2: Server's implementation code.

5.2.2 Client

UnityEngine.Networking namespace used also for the client's implementation. In this case the class which starts the client is the NetworkClient class. This is a class used by the networking system. It contains a NetworkConnection the use of which is to connect to a server. The NetworkClient is able to manage the connection state, messages handlers, and connection configuration. There can be many NetworkClient instances in a process at the same time.

Firstly, a NetworkClient instance had to be created. Then, the function that connects the client to the server is called. This function uses the ip address of the server and the port number he listens to. As soon as the client connects to the server successfully, the smartphone vibrates so the user understands that the connection is established. The next step is the implementation of the functions that send the necessary data. These functions are being called through the User's Interface, manually (case of a button) or in every frame (case of gyroscope sensor data), which runs parallel to the client. They make use of the unique id numbers defined in the server for each type of data. For example, the client's function that sends the gyroscope sensor data is being matched with the server's function that receives the gyroscope data. Finally as mentioned, before sending the messages the data are converted to a string type as shown in the example below.



```
static NetworkClient client;
client = new NetworkClient();

public void Connect()
{
    if (serverIP != null)
    {
        client.Connect(serverIP, currentPort); // 4444
    }
}

public void SendGyroscopeData(short id, Quaternion gyroData)
{
    if (client.isConnected)
    {
        StringMessage msg = new StringMessage();
        msg.value = gyroData.ToString();
        client.Send(id, msg);
    }
}
```

Figure 5.3: Client's implementation code.

5.3 Controllers

5.3.1 Buttons

A convenient and extremely fast way for creating a button in Unity is to use the IMGUI toolkit. IMGUI stands for Immediate Mode Graphical User Interface and it's a code-driven UI toolkit that uses the OnGui function (and scripts that implement the OnGUI function) to draw and manage its user interface. IMGUI can be used in order to create in-game debugging displays, custom Inspectors for script components, and windows or tools that extend the Unity Editor. Although, it's not the best option for building the UI of your game or application because the OnGui functions are being executed in every frame so it has high performance cost.

In this project, the buttons of the UI are created by using the UnityUI toolkit. As mentioned before, this is a GameObject-based UI system and not code-driven like the IMGUI, so the buttons are created through the Unity Editor under the Gameobject hierarchy of the project. By default, a Button gameobject consists of the Button itself and a child Text component. The Text is a standard Unity UI text that is used to render text messages on the screen for various purposes such as labels, buttons, and other information.

On the Button object itself, the two most important components are the Image and the Button component. The Image component is one of the main graphic elements of the UI system in Unity. It is used for many features such as buttons and panel backgrounds, slider handles, speedometers. It is a non-interactive control that displays a sprite, with many options for customization. For example, colour can be applied to the image, assign a material to it, control how much of the image displays or even animations.

The Button component is responsible for the functionality of the button it's attached to. The first options of the component focus on the Button transition, or how it responds when it's interacted with. By default, this transition is set to Colour Tint so it changes colour when interacted, but it can be set to Sprite Swap or Animation. Sprite Swap will change the actual backround image of the Button while the Animation will allow different animations to play depending on what the Button is doing. The Transition can also be set to None, but it is recommended to keep at least the default, as it gives the user a visual cue that they clicked the Button successfully. The bottom section of the component is dedicated to creating OnClick interactions. In other words, the Button can be clicked in order to trigger an event. It is designed to initiate an action when the user clicks or releases it. If the mouse or finger is moved off the button control before the click is released, the action does not take place.

In this project, Buttons are using both PointerDown and PointerUp events. That means, two actions take place when the user presses the button, the first when he presses and the second when he releases. Button events call custom functions from the project scripts. More specifically, Buttons related to controller customization need only one event while those with which the user interacts in a game use both events. This is essential beacause the game must "know" when the player either presses or releases a button.

5.3.2 Navigation Joystick

A virtual joystick is a useful input method for a game. It can be used to navigate a first person character in the game environment, to drive a motorized vehicle, to pilot a helicopter, to move a player in a football game, etc. Such an input method is an essential tool for the controller of this project.

For the joystick implementation, two images were created through the UnityUI toolkit. The first one represents the joystick base and delimits its movement based on the image boundaries while the second image is the mobile part of the joystick. The second image is located inside the first one and its movement is limited from its initial position to the boundaries of the first image. Regarding the hierarchy of objects in the scene of the Unity Editor, the second image is child of the first.

Since the graphical environment of the application is 2d, the position of the joystick can be represented in a two-axis system, Vertical axis and Horizontal axis. Once the controller is connected to the game, the original position of the mobile part of the joystick is stored in a variable. This is necessary for the monitoring of its movement. Its movement is calculated in each frame by comparing its current position with the original one. When the user touches and drags the joystick then the deviation of its position compared to the initial one is calculated, both for the vertical axis and the horizontal axis. The two separate deviations are stored in two variables and sent via the client functions to the game. Nevertheless, these data are raw numbers that represent distance units. For their proper utilization they need to be adjusted, something that will be analyzed in the next chapter.

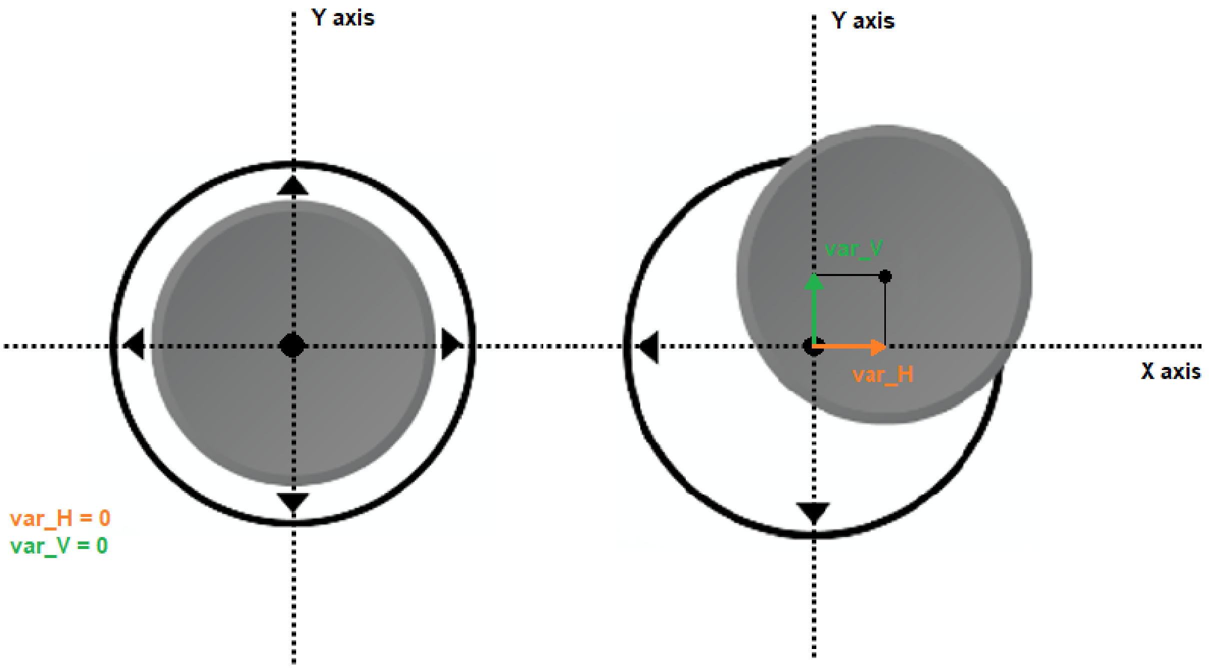


Figure 5.4: Joystick movement axis.

5.3.3 Sensors

The sensors used by the controller are the accelerometer and the gyroscope. Accelerometers in mobile phones are used to detect the orientation of the phone.. That means it uses a three dimensional system. The gyroscope uses an additional dimension compared to the accelerometer. This extra dimension is added by tracking rotation or twist. It simply measures the angular rotational velocity. In contrast with the gyroscope, an accelerometer measures the linear acceleration of movement. Actually, that means it is able to measure the directional movement of a device but it can not resolve its lateral orientation or tilt during that movement accurately unless a gyroscope is there to fill in that info.

UnityEngine namespace provides us with functions that can access the smartphone sensors and read their data. Accelerometer data are recognized by Unity as a Vector3. A Vector3 is a 3-tuple struct that contains three float variables. Vector3 is also used to store the gyroscope rotation rate data. On the other hand gyroscope attidute data are presented as a Quaternion. Quaternions are used to represent rotations. They are really usefull beacuse they are compact, don't suffer from gimbal lock and can easily be interpolated. Gimbal lock is defined as the loss of one degree of freedom in a three-dimensional mechanism that occurs when the axes of two of the three dimensions are driven into a parallel configuration. Thus, the system is "locked" into rotation in a degenerate two-dimensional space. Unity internally uses Quaternions to represent all rotations. Due to the fact that they are based on complex numbers, Quaternions are not easy to be understood. [34] [35]

The first step of the implementation was to check if the device running the application supports the sensors. If the gyroscope and accelerometer are supported and functional then it starts reading their data. The application "pulls" new data in each frame but the transmission doesn't take place immediately. In one way, the data are sent to each frame, which is fast, but there is large deviation between the values of each frame if the user rotates the smartphone very fast. This may not be very functional for the game that receives the data. The other way of transmitting the data is slower, it does not take place in every frame but normalizes the sensor values. This is achieved by storing data samples and calculating their average value. When the average value calculation process is completed it is sent to the server.

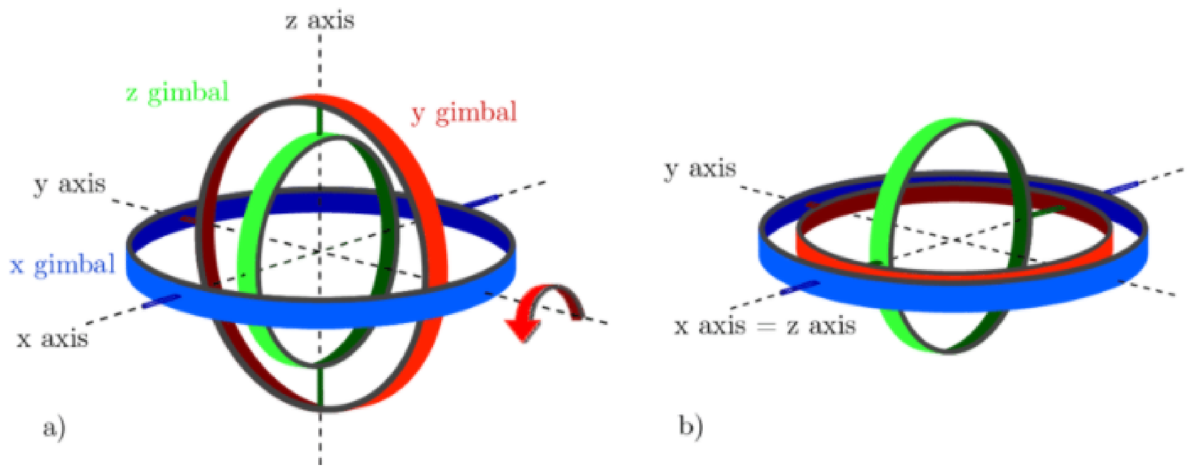


Figure 5.5: Gimbal Lock: (a) All axes are perpendicular to each other. After a rotation of 90 degrees around the y axis (symbolized by the red arrow), the blue and the green frame possess the same rotation axis (b). This situation impedes the clear determination of the rotation axes when subsequently rotating around the x or z axis.

5.3.4 Customization

As mentioned in previous chapters, there was a need for the user to be able to adjust the controller. This need comes from the fact that smartphones on the market and random users hands have different sizes. The user should feel comfortable handling the controller. In addition, each game requires different features from a controller.

Thus, the application offers the user the capability, not only to select the features of the controller, but also to adjust their position on the screen of the mobile phone and their size according to his needs. In order to do this, the UnityEngine.EventSystems extension of the UnityEngine namespace was used. The EventSystem is responsible for processing and handling events in a Unity scene. It is a way of sending events to objects in the application based on input. Inputs methods are a keyboard, mouse, touch, other custom methods.

Drag a Gameobject:
Regarding the change of position of an object on the screen, the OnDrag () function was used, which uses PointerEvent data. Such events are triggered when the user touches the screen of the device or left-clicks with the computer mouse.

The position of each UI element on the Canvas is determined by a reference point. For example, two images which are visually in the same position on the screen but have different reference points, their position is different on the application canvas. The UI objects of the controller also have their own reference points. When the user touches an object and slides his finger on the screen the object must follow its movement. Thus the position of the reference point of this object is updated in every frame with the position of the user's finger.

One issue addressed with this procedure is that the user should be able to drag the object by touching it at any random point of its surface. However, as mentioned above, the position of the object is determined by its reference point and its position is automatically updated with the position of the user's finger. Thus, as soon as the user touched the surface of the object in order to drag it on the screen, the reference point and consequently the whole object was instantly transferred to the contact position. As a result, there was no complete control of the object position on the screen.

To deal with this problem, an offset variable had to be calculated. As soon as the user touches an object, the deviation of the contact position with that of the reference point of the object is calculated. Then, the object position is updated in every frame according to the contact position of the user but subtracted with the deviation (the offset variable) that has been calculated. Thus the object remains still on the first touch and starts moving smoothly when the user slides his finger on the screen.

Scale a Gameobject:
In the Scale process of an object we study the case where the user touches the screen of the smartphone with two fingers. In this case of course, at least one of the two contacts must be on the surface of the object he/she wants to adjust. In this way, the application "understands" whether it will scale an object and which object is that. Given the above,

as soon as there is a second contact on the screen, the initial distance between the two contacts is calculated. In the next step of the implementation, the application checks in every frame if the user moves at least one of his fingers and if so, then the distance between the two contacts is calculated again. At the same time, the object scales accordingly. Its scale is determined by a scalefactor. This factor is calculated, also in every frame, based on the initial and current distance of the two contacts on the screen.

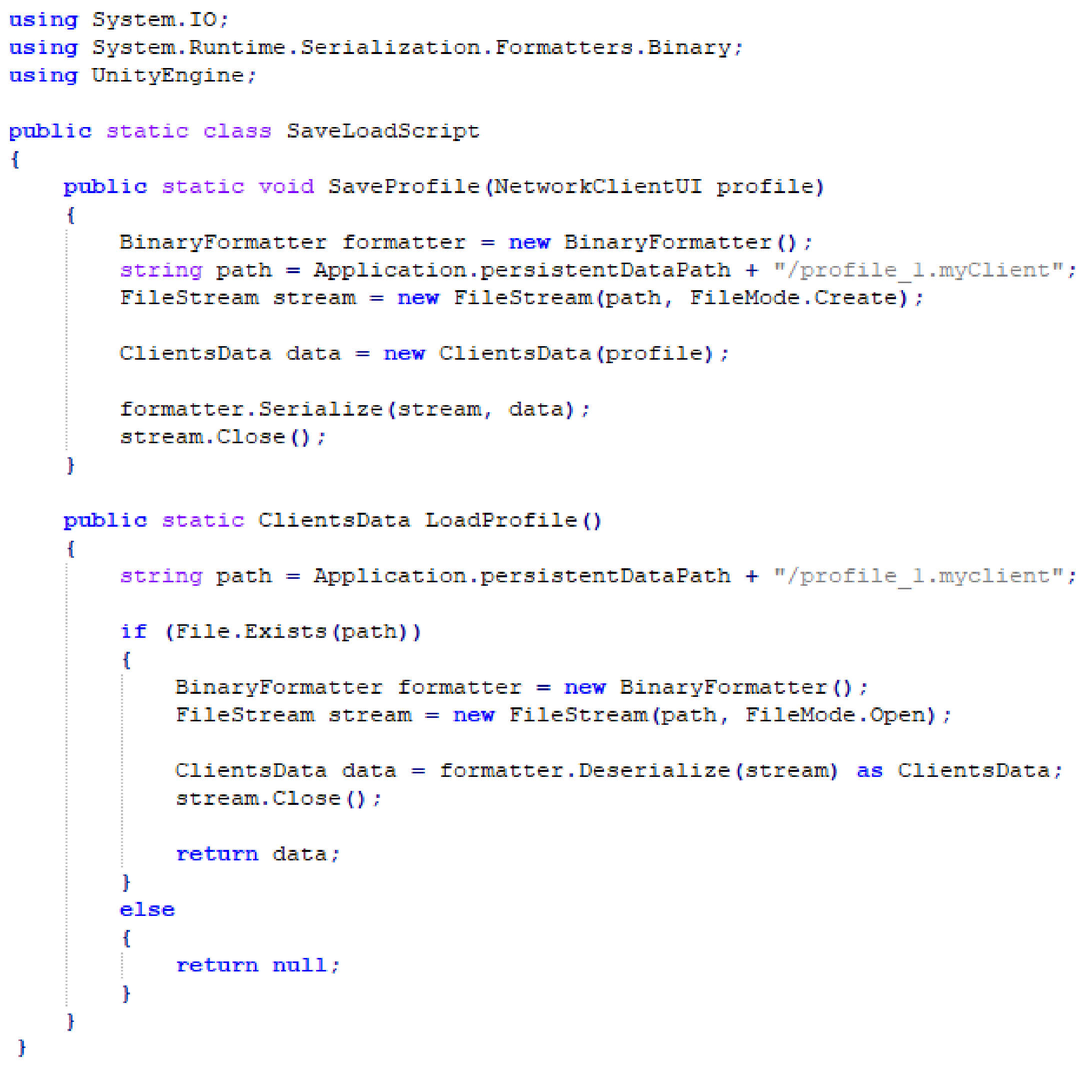
5.3.5 User's Profiles

One capability the controller offers is that it can be used in various types of games. Every game is different, so it requires the controller to have different features. The application developed in this thesis allows the user to adapt the controller to the needs of the game by building a certain controller profile and also to save the profile he made for future use.

To do this, a class named ClientsData was created which contains a data structure with all the necessary informations. As mentioned above, the user can choose to use a gyroscope sensor, an accelerometer sensor, one virtual navigation joystick and two virtual buttons. So, the application saves in the data storage of the device information about the features selected by the user and in the case of the joystick and the buttons, there is also information about the position in which he placed them on the screen as well as the size he gave to them.

At the top of the data structure class, the System.Serializable code is displayed. This makes the whole class serializable. Generally in computing, serialization is the process of translating a data structure or object state into a format that can be stored (for example, in a file or memory data buffer) or transmitted (for example, across a computer network) and reconstructed later. This definition is also applied for Unity game engine.

The client data structure is being saved in a binary file which is stored in the storage of the device. To achieve this, two libraries of the Unity System namespace were used. The System.IO which is a code library to handle (create, open, read, write, delete, etc) files and the System.Runtime.Serialization.Formatters.Binary library which handles binary conversions.



```
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
using UnityEngine;

public static class SaveLoadScript
{
    public static void SaveProfile(NetworkClientUI profile)
    {
        BinaryFormatter formatter = new BinaryFormatter();
        string path = Application.persistentDataPath + "/profile_1.myClient";
        FileStream stream = new FileStream(path, FileMode.Create);

        ClientsData data = new ClientsData(profile);

        formatter.Serialize(stream, data);
        stream.Close();
    }

    public static ClientsData LoadProfile()
    {
        string path = Application.persistentDataPath + "/profile_1.myclient";

        if (File.Exists(path))
        {
            BinaryFormatter formatter = new BinaryFormatter();
            FileStream stream = new FileStream(path, FileMode.Open);

            ClientsData data = formatter.Deserialize(stream) as ClientsData;
            stream.Close();

            return data;
        }
        else
        {
            return null;
        }
    }
}
```

Figure 5.6: Saving/Loading Class.

After the construction of the data structure class, an other class was implemented which handles the saving and the loading process of the data. The SaveProfile function of this class implements the saving process. More specifically, it creates a blank file with a desired name in a path inside the device storage. Then it constructs the structure with the desired data using the ClientsData class and finally it serializes the whole data structure into the binary file and terminates the procedure. Respectively the LoadProfile function is responsible for the loading process of the data. In this case, the function checks whether the requested data recovery file exists. If it does, then the context of the file is deserialized into a data structure again. Then, the loaded data structure is used as a source of information for the variables related to the controller features. As a result, the model of the controller being built is the same as that stored in the user's profile.

5.4 Demos

5.4.1 3D Environment

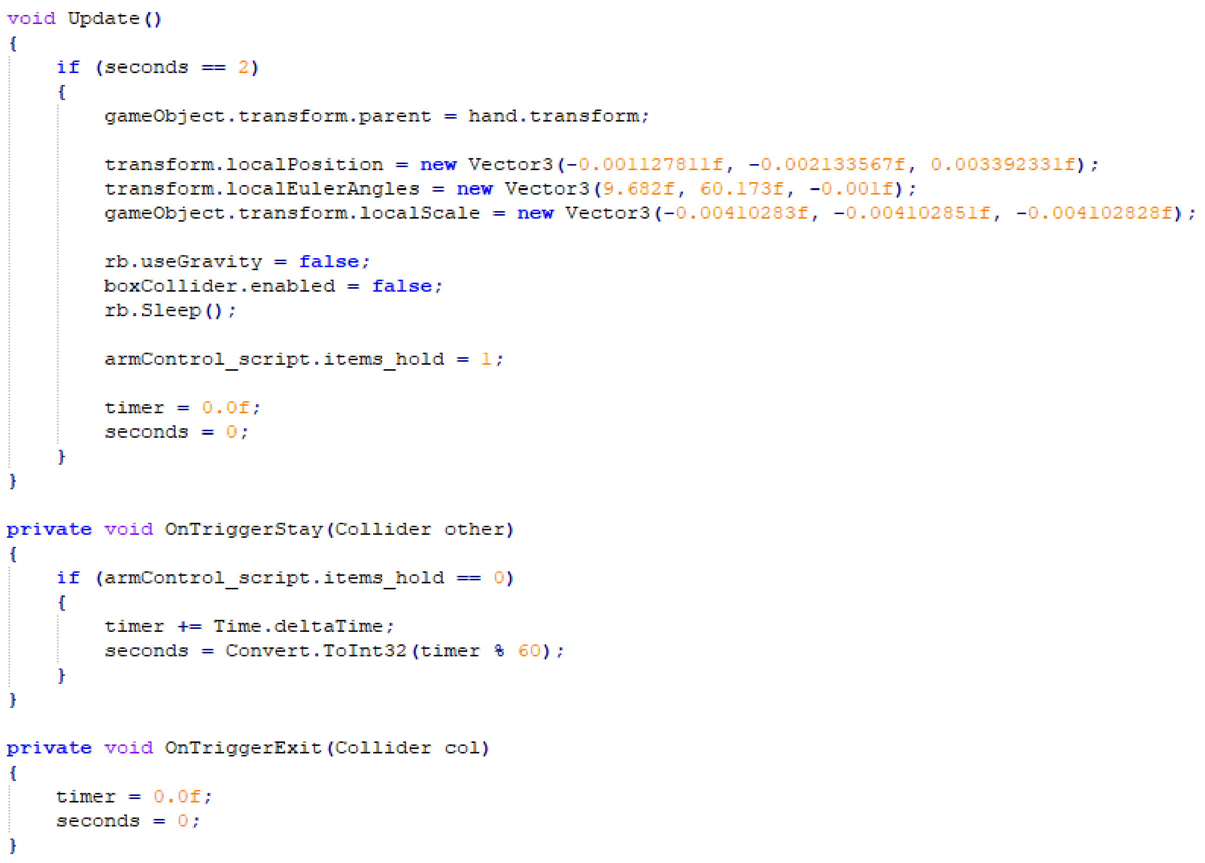
One of the basic parts of a game is its environment. In order to construct its environment, the basic models offered by the Unity game engine were used, such as 3d cubes, spheres, terrain, as well as 3d models which are available for free on the internet. Each of these models was adjusted (placed and resized) appropriately inside the game scene. Then materials were added to them to make them more realistic.

Materials are responsible for the look of an object in the game. They manage the texture of the object, the color, the reflectiveness, how transparent it is. They are assets that take on the properties of something called a Shader. A Shader is a set of code that defines how the game engine renders an object on your screen. It takes into consideration the viewing angle, the lighting and many others properties such as color, texture. The shader "runs" that object through a specific shade and presents what it looks like to your screen. Unity has its own standard shader, that allows us to change many different things. We can add textures, color, transparency, emission, etc.

5.4.2 User's Interaction

In order to make the user able to interact with the objects of the two games, both of them contain and use the server which was built at the beginning of the implementation of this thesis. All the interactios, except the motion of the camera of the game which is the user's eyes in the virtual world, are done through the controllers. The use of the server is essential as all the data transmitted by the controllers are received from the server. Then, each application makes use of these data accordingly.

In both games the user is able to navigate in the virtual environment by using the navigation joystick offered by the controller. Concerning the user's visual contact with the virtual world, the camera that plays the role of his/her eyes rotates based on the gyroscope of the smartphone that "runs" the game and it's mounted inside the VR headset. Thus, the user has a realistic view of 360 degrees of the game. Apart from the player's navigation, the user interacts with specific objects of the scene. In order to do that, Physic Events were used. That kind of events are explained in subsection 3.1.7. An example is shown below:



```
void Update()
{
    if (seconds == 2)
    {
        gameObject.transform.parent = hand.transform;

        transform.localPosition = new Vector3(-0.001127811f, -0.002133567f, 0.003392331f);
        transform.localEulerAngles = new Vector3(9.682f, 60.173f, -0.001f);
        gameObject.transform.localScale = new Vector3(-0.00410283f, -0.004102851f, -0.004102828f);

        rb.useGravity = false;
        boxCollider.enabled = false;
        rb.Sleep();

        armControl_script.items_hold = 1;

        timer = 0.0f;
        seconds = 0;
    }
}

private void OnTriggerStay(Collider other)
{
    if (armControl_script.items_hold == 0)
    {
        timer += Time.deltaTime;
        seconds = Convert.ToInt32(timer % 60);
    }
}

private void OnTriggerExit(Collider col)
{
    timer = 0.0f;
    seconds = 0;
}
```

Figure 5.7: Physic Event example.

As mentioned in the previous subsection, the user has some simple taks within the virtual environment. In the first demo, he/she has to transfer some packages from one workbench to another. In order to be able to do this, a virtual hand was created which is initially aligned with the controller the user holds in his hand. Then, in each frame, the virtual hand rotates based on the controller gyroscope and consequently the smartphone. The orientation of the hand is being updated in each frame in a smooth way in order to give as much realism as possible to the game. Finally, a virtual button is required which is used in order to leave the package the user holds on the second workbench.

In the matter of the second demo the user uses the smartphones to control the motion of a shield and a sword. His task is to "kill" some static enemies who are being around him. As soon as he approaches and hits one of them with the sword then the enemy object is destroyed. However, each enemy hides a trap from which the user must protect his virtual self by using his shield for cover. The motion of the weapons is realistic and corresponds to the motion that one would make in a real battle. For example, in order to be covered by an object falling from the sky the user must raise above his head the hand with which he holds the controller that represents the shield.

5.5 Evaluation

The results of the implemented applications meet the initial expectations of this diploma thesis. Firstly, the controllers connect to the demos without problems and the connection remains stable and without interruptions. Their response is immediate and quite accurate. Of course, this depends on the quality of the smartphones in which the applications run, both in terms of the processing power of their units and in terms of the quality of the sensors they contain. The smartphones in which the applications were tested are of mid-range according to today's market. Nevertheless, the applications ran at satisfactory fps (58 - 64 fps) and their sensors are quite accurate and meet the expectations of the two demo games.

The demos have been implemented for normal use on 2D screens but also for virtual reality mode. In the first case the user can navigate freely in the virtual environment and interact with it. Regarding the virtual reality mode there is a limitation due to the nature of the library used for its implementation. More specifically, as mentioned in a previous chapter there is a camera in the game which plays the role of the user's eyes in the virtual world. Thus, when the user navigates through the 3D space what is really happening is a translation to the camera, according to the user's input. The term "translation" in Unity game engine means the change of the position of an object.

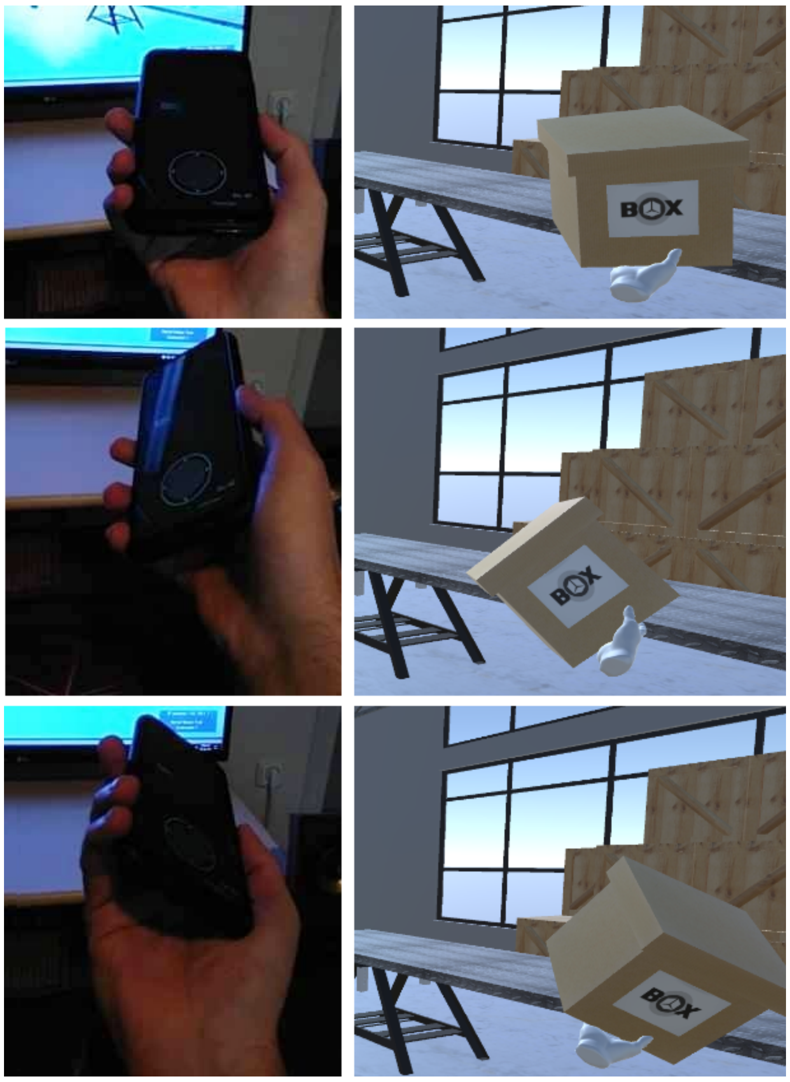


Figure 5.8: Virtual hand's motion.

The GoogleVR library offered by Google for the purpose of creating VR applications on the Unity platform was used in order to "build" the two demos. The limitation mentioned earlier is that when the application is intended for Virtual Reality use, the game camera is completely controlled by this library. The camera can rotate 360 degrees but its position remains stable. A a result, the user has a complete, 360 picture of the virtual environment but can not navigate through it.

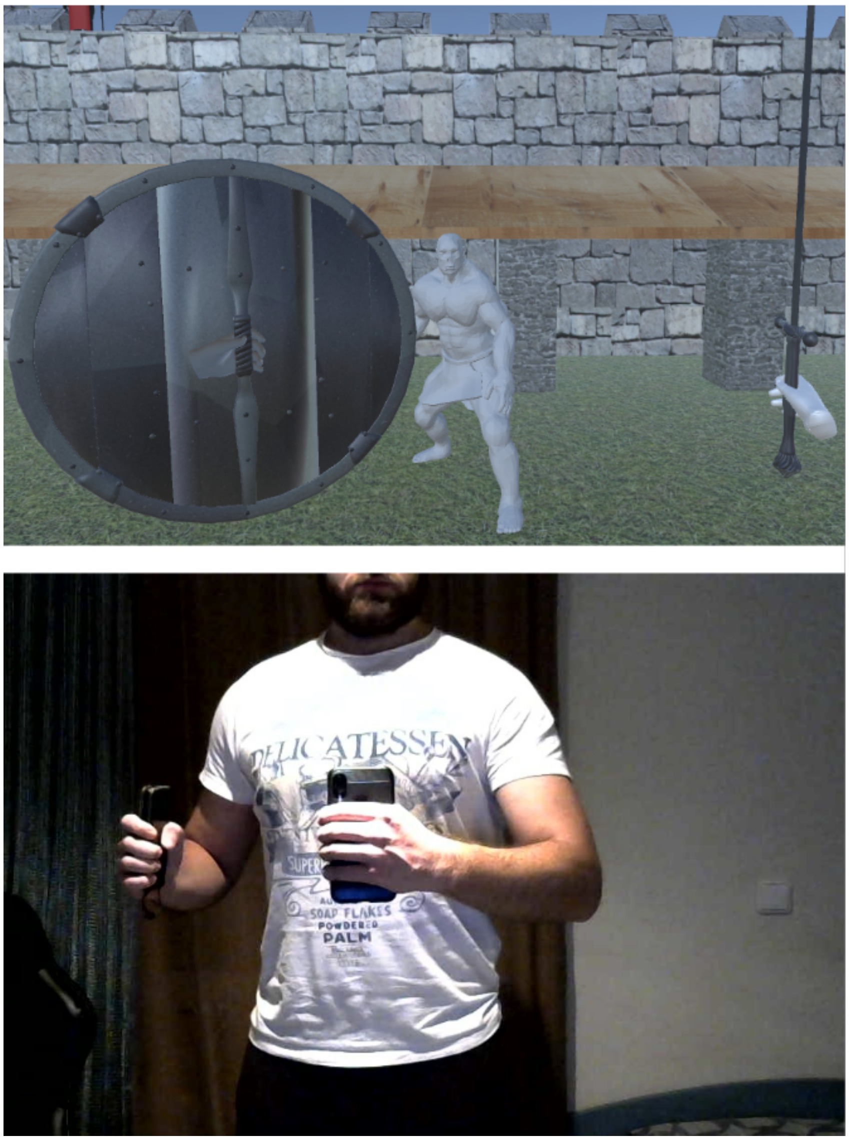


Figure 5.9: Virtual hand's motion.

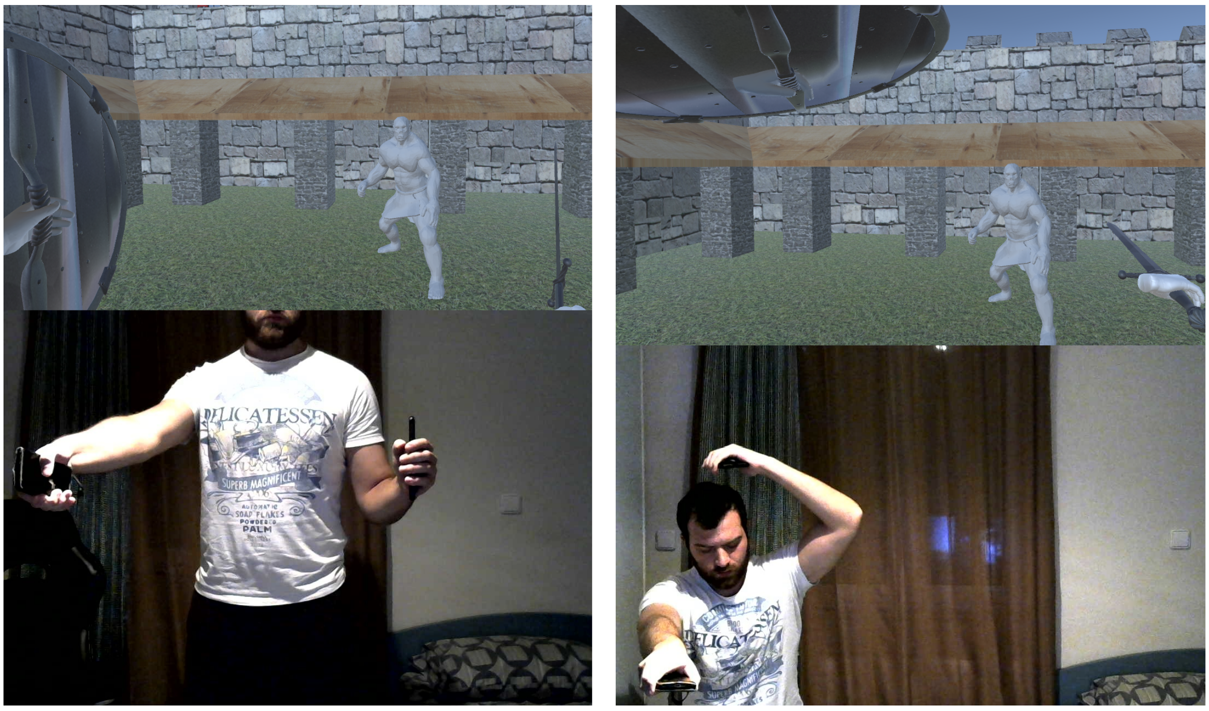


Figure 5.10: Virtual hand's motion.

Comparing to Desktop HMDs:
Head Mounted Displays for desktop VR applications come with their own specially designed controllers. Such controllers hold a gyroscope and an accelerometer in order to calculate the orientation of the controllers but not their position in space. In order to make this possible, they also contain special sensors which can track the position they have each time. This is a big advantage that we do not find in smartphones. In the case of the two demo games of this thesis, the motion of the virtual hands is adjusted accordingly in order to be able to move through the virtual space and not only rotate. Their entire motion depends on the gyroscope orientation. Nevertheless, due to this limitation smartphones function as controllers is significantly limited.

In addition, a gaming controller is allowed to use more buttons as its surface is embossed and the user easily understands which buttons he touches and which he does not, in contrast to a smartphone screen where the use of virtual buttons is limited. Also, the shape of such controllers is not random. They are specially designed to fit in every hand and make the user feel comfortable while holding them. The flat, rectangle shape of a smartphone might be uncomfortable for some users when they hold it. In general, the use of a smartphone as a controller for VR applications lags behind the specially designed controllers of concepts like the Oculus Rift, HTC Vive, Playstation VR, etc, but let us not forget that the purpose for which they were built is by no means to play the role of a controller in a game.

Evaluation from random users:
The applications were tested by former students of the Technical University of Crete and by some people of my social circle. They reported that the response of the controllers in the game is very fast and accurate. In addition the user's interface is very easy in use and well made. Another important fact they mentioned is that the feature which allows the user to adjust the position and size of the virtual buttons and the navigation joystick on the screen of their smartphone seemed very useful as the personal preferences and the size of the hands changes from user to user. As a negative feature, they reported that the graphics of the two demo games are sketchy and made the VR experience unpleasant. Nevertheless, this thesis did not focus on creating a complete, perfect game but on creating software that allows the use of a smartphone as a controller for a Virtual Reality application based on mobile platforms.

Chapter 6

Conclusion

Virtual Reality is getting more and more advanced over the years. Nowadays it is used in many aspects of our lives, not only for entertainment reasons. This is a reason why keep working on VR related technologies is important. Modern impressive computer graphics alongside with hand input technologies have made VR applications very convincing to people. The challenge of this project was to give users the opportunity to use their hands in order to interact in a smartphone based VR application without the need of additional, special hardware. All they need to have is one or more additional smartphones.

The concept of this thesis was to develop a software that consists of two individual android applications connected through a WiFi antenna. The first application needs to be installed on the controllers (smartphones) and it is programmed to utilize some of their technological features. It transforms the mobile phones to hand controllers by reading the required data and then transmits them to the second application which handles them accordingly. As a result the user is able to interact and manipulate an application running on a smartphone by using an other smartphone as hand controller. This is very useful for smartphone based Virtual Reality applications because the user is not able to touch the device running the VR app. The results of the implemented applications meet the initial expectations of this project. The connection of the devices is stable and the controllers' response is immediate and accurate.

6.1 Future Work

This project is implemented in such a way to be functional, easy in use and to offers the user a variety of choices. Although, like all projects this one can be extended and improved in many aspects too. Some improvement suggestions are shown below:

- Improve the graphics of the games so the VR experience is more pleasant
- Improve the user's interface of the controllers (e.g appearance, ease of use)
- Improve performance so the applications run at higher FPS
- Add some controller functions (e.g slider, more buttons if possible, volume +/- buttons)
- Offer more customization choices (e.g colours, shapes of the virtual objects)
- Use Wifi Direct in order to connect the smartphones directly
- Use a BlueTooth connection in order to connect the smartphones
- Latency and accuracy measurements
- Develop a multiplayer game for the controllers
- Use a UDP connection protocol for the applications

References

1. https://www.marxentlabs.com/what-is-virtual-reality
2. https://www.vrs.org.uk/virtual-reality/what-is-virtual-reality.html
3. https://en.wikipedia.org/wiki/Virtual_reality
4. https://en.wikipedia.org/wiki/Virtual_reality_applications
5. https://www.fdmgroup.com/5-exciting-uses-for-virtual-reality/
6. https://en.wikipedia.org/wiki/Smartphone
7. https://www.appliedmaterials.com/en-il/node/3354511
8. https://www.researchgate.net/publication/302074437_FaceTouch_Touch_Interaction_for _Mobile_Virtual_Reality
9. https://www.researchgate.net/publication/301462099_Creating_a_Mobile_Head-mounted _Display_with_Proprietary_Controllers_for_Interactive_Virtual_Reality_Content
10. https://ieeexplore.ieee.org/document/7926626
11. https://docs.unity3d.com/2019.2/Documentation/Manual/OculusControllers.html
12. https://docs.unity3d.com/2019.2/Documentation/Manual/OpenVRControllers.html
13. http://www.3ddatabase.com/3d-models.html
14. http://korichambo.blogspot.com/2016/10/unit-6667-understand-theory-and.html
15. https://en.wikipedia.org/wiki/Game engine
16. https://en.wikipedia.org/wiki/Unity_(game_engine)
17. https://www.cryengine.com/
18. https://www.unrealengine.com/unreal-engine-4
19. https://aws.amazon.com/lumberyard/faq/
20. https://docs.unity3d.com/2020.1/Documentation/Manual/UICanvas.html
21. https://docs.unity3d.com/Manual/UsingTheInspector.html
22. https://docs.unity3d.com/Manual/class-Transform.html

23. https://docs.unity3d.com/560/Documentation/Manual/PhysicsSection.html
24. https://docs.unity3d.com/Manual/comp-MeshGroup.html
25. https://docs.unity3d.com/2020.1/Documentation/Manual/Materials.html
26. https://titanwolf.org/Network/Articles/Article?AID=8daed109-76b3-4ec3-bb32-b82daca7ec75 #gsc.tab=0
27. https://docs.unity3d.com/2019.3/Documentation/Manual/class-AudioSource.html
28. https://docs.unity3d.com/2017.2/Documentation/Manual/CreatingAndUsingScripts.html
29. https://docs.unity3d.com/560/Documentation/Manual/EventFunctions.html
30. https://docs.unity3d.com/Manual/Console.html
31. https://en.wikipedia.org/wiki/User_interface
32. https://en.wikipedia.org/wiki/API
33. https://www.youtube.com/watch?v=7B7uehNBLRY&ab_channel=Chidre%27sTechTutorials
34. https://en.wikipedia.org/wiki/Quaternion
35. https://en.wikipedia.org/wiki/Gimbal_lock#: :text=Gimbal%20lock%20is%20the%20loss, misleading%3A%20no%20gimbal%20is%20restrained.