OULU UNIVERSITY OF
APPLIED SCIENCES

Eetu Suni

CONTROLLING PHYSICAL DEVICES FROM VIRTUAL REALITY

CONTROLLING PHYSICAL DEVICES FROM VIRTUAL REALITY

Eetu Suni
Bachelor’s Thesis
Spring 2019
Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Information Technology

Author: Eetu Suni
Title of the bachelor’s thesis: Controlling Physical Devices from Virtual Reality
Supervisors: Juha Hannula, Kari Jyrkkä
Term and year of completion: Spring 2019 Number of pages: 39

This thesis was commissioned by Nokia. The aim of this thesis was to define interfaces between a real and virtual world, find different ways to give a user input to virtual reality and determine what kind of feedback can be provided from physical devices to virtual reality. As a practical implementation, a virtual reality control system was created for the base station testing laboratory.

The virtual reality application in the practical implementation was created on top of Unity Engine. SteamVR was used as a programming interface to virtual reality hardware. All the needed 3D models were designed using Blender.

As a result, a fully working virtual reality application for the test laboratory controlling was done and the requirements were met. This document provides basic guidelines with a comprehensive set of examples for developing virtual reality applications to control physical devices.

Keywords: Virtual Reality, Unity, SteamVR, Remote Control

CONTENTS

ABSTRACT 3

TABLE OF CONTENTS 4

VOCABULARY 6

1 INTRODUCTION 7

2 VIRTUAL REALITY 8

2.1 Virtual reality equipment 8

2.2 Applications of virtual reality 10

3 UNITY ENGINE 11

3.1 Asset 11

3.2 GameObject 11

3.3 Component 11

3.4 Scene 14

3.5 Shader 14

3.6 Skybox 15

3.7 Prefab 15

4 INTERACTION BETWEEN VIRTUAL REALITY AND REALITY 16

4.1 User inputs and interactions 16

4.1.1 Hand tracking 17

4.1.2 EEG Headset 17

4.1.3 Speech recognition 18

4.2 Feedback from the real world 18

4.2.1 Video stream 18

4.2.2 Positioning 20

4.2.3 Sensor values 21

4.3 Communication between a VR application and physical devices 22

5 THE WALL USE CASE 25

5.1 Virtual Otava 26

5.1.1 Virtual Otava functionality 27

5.1.2 Communication between Virtual Otava and physical switchbox 30

5.1.3 Heatmap generation 32

5.1.4 Code sample for grabbing phones 33

6 CONCLUSION 35

VOCABULARY

API	Application Programming Interface
AR	Augmented Reality
AV	Augmented Virtuality
BTS	Base Transceiver Station
GPS	Global Positioning System
GUI	Graphical User Interface
HMD	Head Mounted Display
IoT	Internet of Things
IP	Internet Protocol
REST	Representational State Transfer
RF	Radio Frequency
RX	Receive
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UE	User Equipment
UX	User Experience
VR	Virtual Reality

1 INTRODUCTION

Virtual reality (VR) is a decades-old technology, but it has not been very popular until now. A Rapid increase of computing power has made it available for a broad user base and companies around the world have adopted virtual reality for many different purposes during the last few years. (1.)

Virtual reality can be used to control the physical world in a way that has not been possible before. The main objective of this thesis was to define interfaces between a digital and physical world, study different methods of giving input to the virtual world and how the feedback from real world can be presented to the user as an intuitive way as possible.

As a practical implementation, a digital copy of the Nokia's over-the-air 5G test laboratory will be done. A user should be able to remotely control physical antennas on the test wall through virtual reality and see how it affects to the signal received from the base station. A Virtual reality environment will be created with Unity Engine.

Nokia is one of the leading multinational telecommunications and information technology companies. Nokia was founded in 1865 in Finland and has been associated in many different industries including forestry, rubber, cables, electricity and electronics. Since 1998, Nokia was a leading mobile phone manufacturer for over 10 years. (2.)

Currently in 2019, Nokia has a focus on developing 5G networks, the Internet of Things (IoT), cloud services and software solutions. Nokia also offers licensing opportunities for companies and conducts research with a lead of Nokia Bell Labs. (3.)

2 VIRTUAL REALITY

Virtual reality is an artificial 3D environment which is completely created or generated by a computer. The user should feel like being part of the environment mentally and physically. This can be achieved with proper VR equipment and careful design of VR environment. That equipment converts an environment into sensory information that a human can see, hear and feel. (4.)

While virtual reality is completely artificial and reality is non-artificial, there are also other levels of reality somewhere between them, such as augmented reality (AR) and augmented virtuality (AV), as illustrated in figure 1. In AR, the environment is real, for example a real-time video feed and it contains overlaid virtual elements, such as 3D objects or text. AV, on the other hand, is an opposite of AR meaning that the environment is virtual and real-world components are integrated into it.

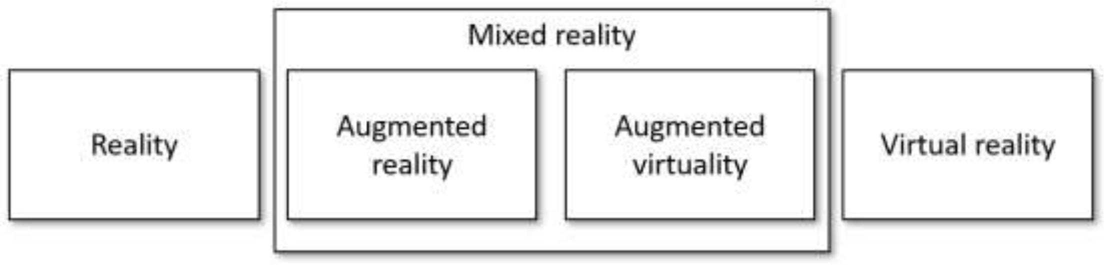


FIGURE 1. The Levels of reality

2.1 Virtual reality equipment

The most common and most important piece of equipment is a head-mounted display (HMD), which provides visual feedback from the virtual environment. HMDs are usually built with two separate displays, one for each eye. Two stereoscopic images from slightly different positions are fed to the displays which provide realistic 3D view of the environment. However, that is not the case with cheaper, usually mobile VR HMDs where only one display is used.

Another common VR device is a controller, usually one for each hand. A typical controller contains few buttons and most of the PC VR controllers have support for a haptic feedback. With a proper tracking system, an HMD and two controllers alone can provide an immersive VR experience. Tracking is accomplished with built-in accelerometers and position sensors, and advanced systems also contain base station tracking. Base stations provide an optical signal, which VR devices can receive and calculate their position accordingly (5). A complete PC VR set called HTC Vive is shown in figure 2.

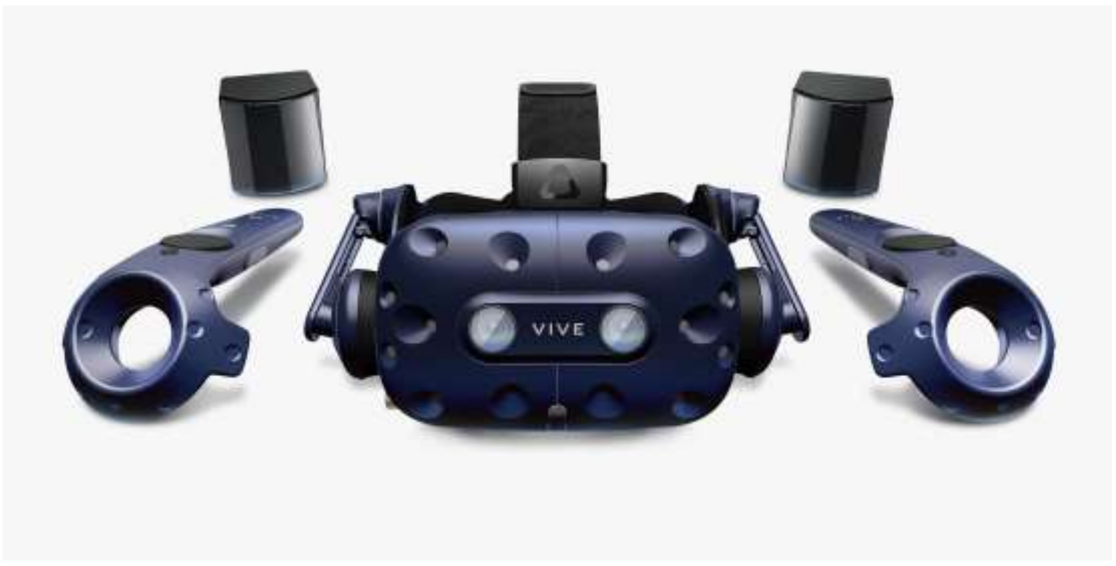


FIGURE 2. The HTC Vive full kit (6)

There are also many other VR gadgets available, such as VR gloves, general purpose trackers, hand trackers, VR treadmills and mechanized chairs. An example of VR tracker is shown in figure 3. It is a small gadget which can be attached to a body or to an object, such as toy gun, to enable tracking of that specific object in VR. (7.)

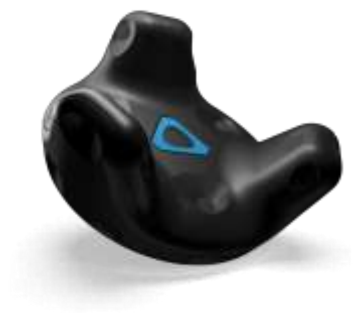


FIGURE 3. The Vive Tracker (8)

2.2 Applications of virtual reality

Virtual reality can be used for many different purposes such as education, visualization, industrial design and entertainment. One of the most important applications of VR is training. It might be dangerous or expensive to train employees in a real world thus a lifelike environment in VR is an inexpensive and safe solution for that. (4.)

VR is also a great way to visualize and even design things in 3D. For example, the user could select and see immediately in real size how different pieces of furniture match together. Industrial designers can use VR for quick prototyping without needing always create a new physical mock-up after making changes to the design. (4.)

VR is widely used for entertainment purposes. Especially different kinds of simulator and racing games are popular in VR. However, high quality and realistic VR environments are computationally too heavy even for a modern gaming PC and therefore, VR has not been able to revolutionize the gaming industry at least yet. Most likely this will change in the future. (4.)

3 UNITY ENGINE

Unity is a multiplatform game engine developed by Unity Technologies (9). It supports the development with C# language but the engine itself is based on C++ (9). Even though Unity is called a game engine, it is widely utilized also in other industries for visualization and simulation purposes (9). Especially, native support for VR and AR and an optimized rendering pipeline makes Unity Engine attractive for visualization purposes (10). Unity is currently in 2019 the world's most popular game engine (11).

This chapter explains the most important concepts and practices of developing content with Unity. The practical implementation of this thesis was created with Unity. Thus, basic understanding of following terms and concepts is needed.

3.1 Asset

In Unity, all supported files or items, which are used to build the game or project, are called assets. An asset can be a representation of a file created outside of Unity, such as audio, an image, a 3D model, a script or any other supported file. Some assets are created inside Unity. (12.)

3.2 GameObject

GameObject is a fundamental object in Unity (13). Every single object in the game is a GameObject (13). It can be thought as an empty container which gets its behaviour and possible visual representation from Components attached to it (13). Components are explained in chapter 3.3. Every GameObject has at least one Component called Transform, which determines the position, rotation and scale of the object (14).

3.3 Component

A Component is an essential building piece of GameObject. Multiple Components can be attached to the GameObject to achieve desired functionality. Unity provides plenty of built-in components to cover the most common functionalities in games, such as a camera, renderers, an audio source and listener, physics and colliders, effects, UI. However, those are not enough for complex features. Thus, Unity provides a way to create new Components by using scripts. (15.)

A New Component can be done by creating a new C# script file. By default, a script file contains a class which derives from a MonoBehaviour. The MonoBehaviour is a base class for all Components in Unity and it provides a connection to Unity's internal methods and properties. A Newly created script file also contains two basic methods provided by the MonoBehaviour. Those are Start and Update. (16.)

A Start method is always called before the first Update call. Start is only called once in the lifetime of the script. The purpose of the Start method is to do required initialisation, such as copying initial position of an object into a variable or finding and saving a reference to another object on the scene. (17.)

An Update method is called once per every frame when a script is enabled. As the name suggests, the main purpose of the Update method is to run game logic that is meant be continuously updated. That kind of logic might be, for example, a movement or an animation. (17.)

A MonoBehaviour class also contains also many other methods in addition to Start and Update, as illustrated in figure 4.

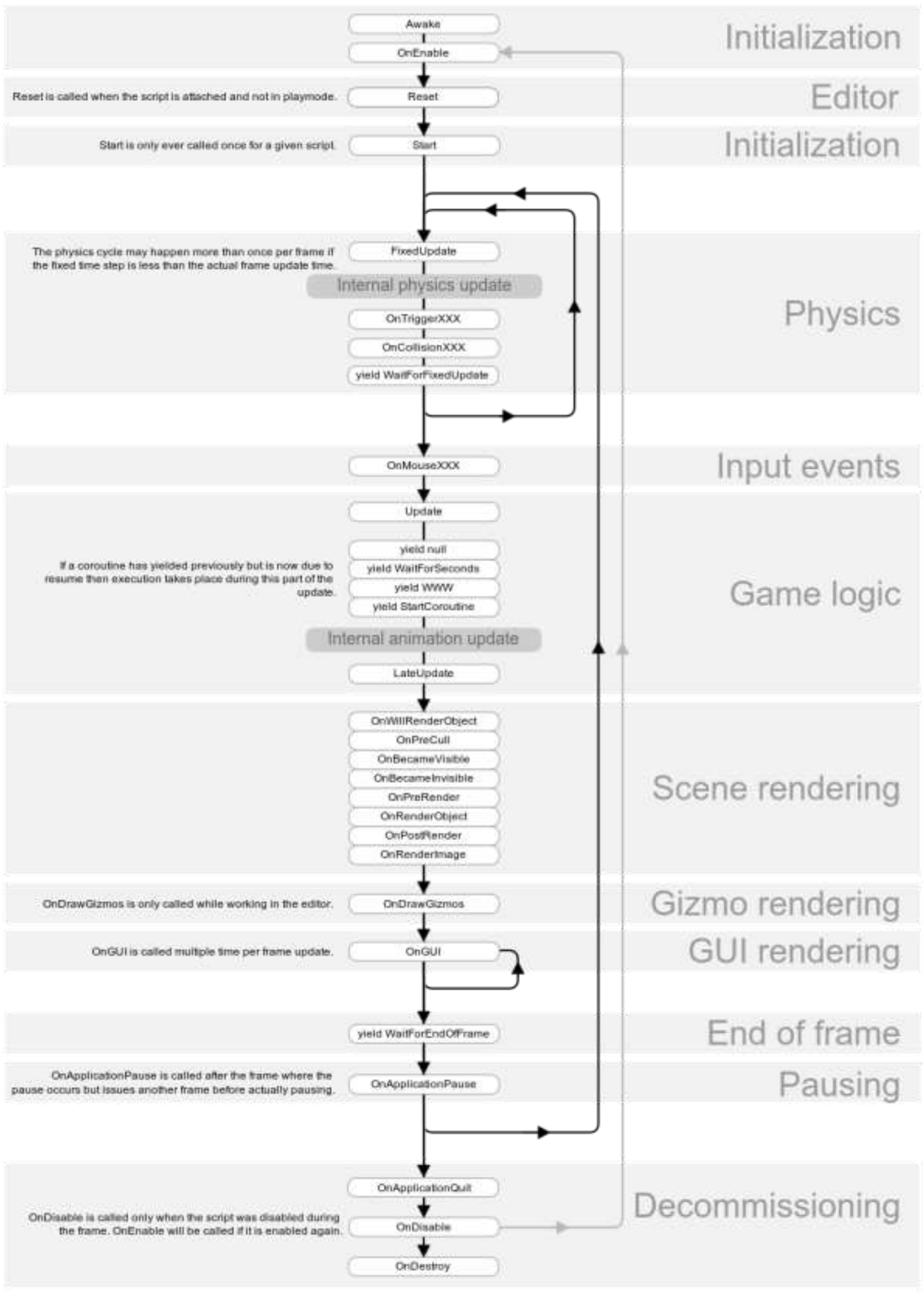

Awake
OnEnable
Initialization
Reset is called when the script is attached and not in playmode.
Reset
Editor
Start is only ever called once for a given script.
Start
Initialization
The physics cycle may happen more than once per frame if the fixed time step is less than the actual frame update time.
FixedUpdate
Internal physics update
OnTriggerXXX
OnCollisionXXX
yield WaitForFixedUpdate
Physics
OnMouseXXX
Input events
Update
If a coroutine has yielded previously but is now due to resume then execution takes place during this part of the update.
yield null
yield WaitForSeconds
yield WWW
yield StartCoroutine
Internal animation update
LateUpdate
Game logic
OnWillRenderObject
OnPreCull
OnBecameVisible
OnBecameInvisible
OnPreRender
OnRenderObject
OnPostRender
OnRenderImage
Scene rendering
OnDrawGizmos is only called while working in the editor.
OnDrawGizmos
Gizmo rendering
OnGUI is called multiple time per frame update.
OnGUI
GUI rendering
yield WaitForEndOfFrame
End of frame
OnApplicationPause is called after the frame where the pause occurs but issues another frame before actually pausing.
OnApplicationPause
Pausing
OnApplicationQuit
OnDisable is called only when the script was disabled during the frame. OnEnable will be called if it is enabled again.
OnDisable
OnDestroy
Decommissioning

FIGURE 4. The Script Lifecycle Flowchart (17)

As seen in the figure 4, there are multiple different update methods. The first type of update in the loop is a *FixedUpdate.* It is called at specific intervals and is not tied to frame rate of game. It should be used for relatively light calculations that require reliable timing, such as physics calculations. (17.)

The second update method is just a regular *Update* which was covered earlier in this chapter.

The last type of update in the loop is a *LateUpdate.* It is called after a regular *Update* has finished. The *LateUpdate* is called once per frame. It could be used, for example, to update a third-person camera. The character position would be first updated in the *Update* method and the follower camera later in the *LateUpdate*. This is how the camera would be exactly synchronized with the character. (17.)

3.4 Scene

A Scene is essentially a set of GameObjects that a developer has put together using Unity. A Scene can be, for example, the main menu of the game, an environment or another group of GameObjects. Usually, each level of the game is separated as own scene. (18.)

Multiple scenes can be running simultaneously (19). It is common to have one scene which contains a Player object and settings related to that. That scene is usually loaded first, and it will stay active through the lifetime of the game. Then, there are usually multiple different environment scenes which are loaded one at a time, and when the user wants to change the environment, a new environment scene is loaded and the previous unloaded.

3.5 Shader

A Shader is a relatively small piece of code made to run on a graphics processing unit (GPU). The main purpose of the shader is to calculate a color of each pixel rendered on the screen. Calculations are usually based on the lighting input and material configuration. Material is essentially an asset which contains pre-defined values for the shader properties. Those values can be for example, a surface color, a texture image or a surface reflectivity. Multiple materials with different value setups can be created from one shader. (20.)

3.6 Skybox

A skybox is basically a visual wrapper around the environment. It is rendered last behind all other objects. A skybox is either a box with six different textures or a tessellated sphere. For example, a picture taken with a 360-camera can be split into six smaller pictures with a special algorithm and those pictures can be then applied into the cubemap-based skybox material. Panorama images can also be used as a skybox. (21.)

3.7 Prefab

Prefab is a pre-created and configured GameObject which is stored as an asset in the assets folder. Prefab can be thought as a template from which new instances can be created. There might be multiple GameObjects in the scene created from one prefab. If a developer makes changes to that particular prefab, those changes are automatically reflected in all instances in the scene. Therefore, it is recommended to use prefabs instead of copying and pasting GameObjects in the scene. Prefabs can also be instantiated at run-time. (22.)

4 INTERACTION BETWEEN VIRTUAL REALITY AND REALITY

In this chapter, different methods of giving user input to virtual reality and methods of providing feedback from real-world devices are covered. Communication between a VR application and physical devices is also explained.

4.1 User inputs and interactions

The user has different ways to give inputs to the virtual environment. Of course, it depends on the hardware in use. The most commonly used set of hardware consists of HMD and two controllers with multiple buttons on each. That combination alone provides a great foundation to build intuitive and powerful methods to interact with the virtual environment.

The biggest advantage of interactions in virtual reality is the absence of physical constraints. The interaction to achieve a desired outcome can be completely different in VR. For example, if the user would like to turn lights on or off in the real world, the user would have to move close to the switch and tap it. In VR, the user could just point a controller towards the light source and press the button. The same philosophy can be applied in controlling physical devices from VR.

One of the most challenging tasks in the VR development is to design well-functioning interaction mechanisms. There are few considerations that should be kept in mind:

- How important is the effectiveness? When designing tools, it is usually one of the most important aspects and the time required for the task should be minimized. Button delays, large arm movements and highly chained actions should be avoided if possible. (23, p. 284.)
- Interaction should be easy to learn (23, p. 284).
- Interaction should require minimal attention to perform after some practice (23, p. 284).
- Comfort is important especially for applications that are usually consumed for long periods of time. The user should not develop a muscle fatigue. Therefore, large movements should be avoided and the user

shouldn’t have to stay at an uncomfortable position for too long. (23, p. 284.)

There are also other ways to provide an input to the VR application in addition to HMD and controllers. Some of them are described below.

4.1.1 Hand tracking

With a hand tracker, such as Leap Motion, the user can use bare hands to interact in VR. Leap Motion is a small device which can be attached to HMD. Tracking of hands and fingers is based on analysing images received from the tracker device. Leap Motion consists of two cameras and three infrared LEDs. (24.)

4.1.2 EEG Headset

Another, not so common way to provide an input is an electroencephalography (EEG) headset, such as EMOTIV EPOC+, which is shown in figure 5. Basically, the user can interact with the VR environment just by thinking something. However, the EEG headset must be trained before it can be used. Training is done so that the user must think something for a while and the headset records electrical activity of brains during that time. Machine learning is used to detect patterns from the signal. (25.)

Later when the user is using the headset, the Emotiv application will continuously try to find patterns from the live signal and compare them to the training data. If similar patterns are detected, a command is forwarded to the VR application as an input. (25.)

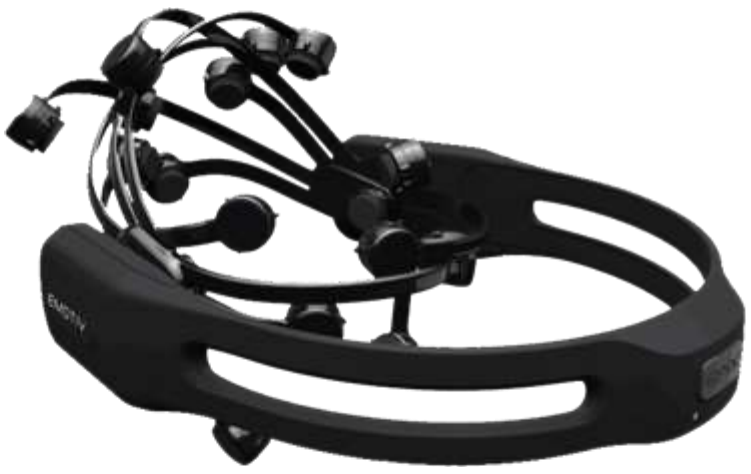


FIGURE 5. The EMOTIV EPOC+ headset (26)

4.1.3 Speech recognition

Speech recognition can be used as an input method in the VR application. Especially, if the application contains a lot of elements or actions that the user should be able to access quickly, the speech recognition might be a more robust solution than a complex graphical user interface (GUI) system.

Unity Engine has a built-in phrase recognition system. However, it currently works only on Windows 10. (27.)

4.2 Feedback from the real world

One of the most important considerations of controlling physical devices from VR is the feedback from the real world. How the user can be sure that the action performed in VR also happened in the real world or how it affected to the system? Some feedback methods are described below.

4.2.1 Video stream

A real-time video stream is a powerful way to provide feedback from the real world. The user can see exactly what the situation is in real-time and how the controlled device or robot is behaving. A video stream can be, for example, a regular 2D or 360-video. In Unity, a 2D video can be rendered to the surface of any object in the scene, as seen in figure 6.

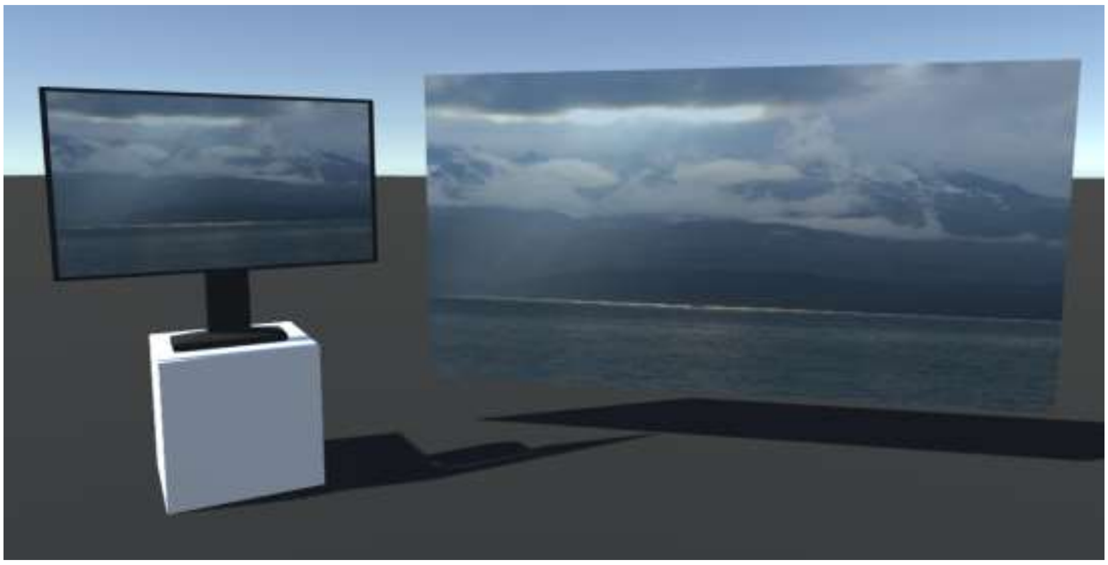


FIGURE 6. The 2D Video stream rendering in Unity

A 360-video stream could be applied to the skybox in Unity, which creates an immersion of that the user is physically there. This is a useful technique for controlling moving things such as drones or radio-controlled cars with a 360-camera attached.

As an example, a raw 360-panorama taken from a drone (seen in figure 7) was brought into Unity and converted to a skybox (seen in figure 8). A virtual cockpit was also added. Thus, the feeling for the user is more natural than it would be if the user would just have to float in the air. Similarly, if that panorama image would be replaced with a 360-video stream from a drone, it would provide an immersive way to control the drone remotely from VR.

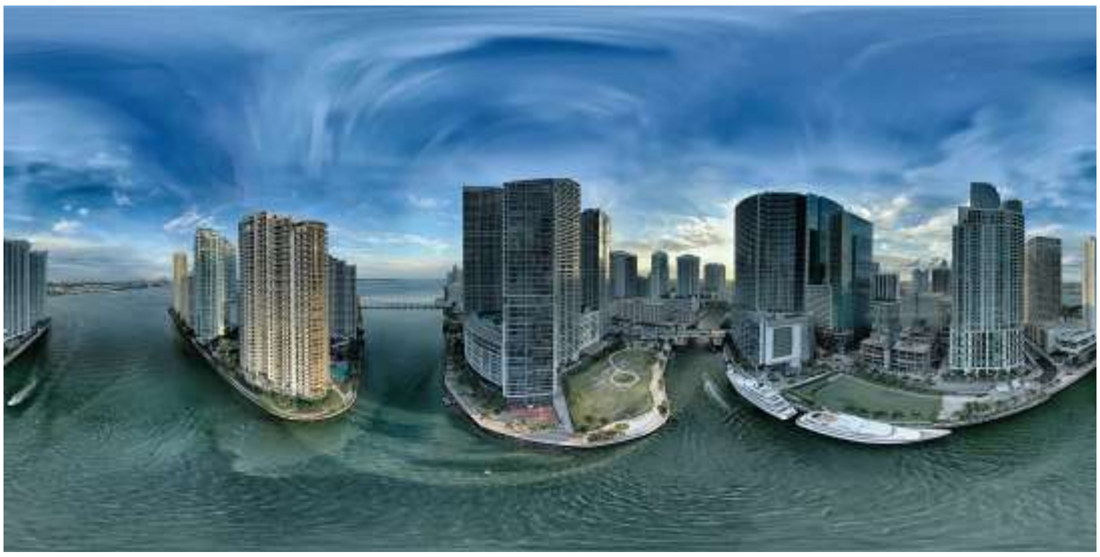


FIGURE 7. The Miami 360-panorama key (28)

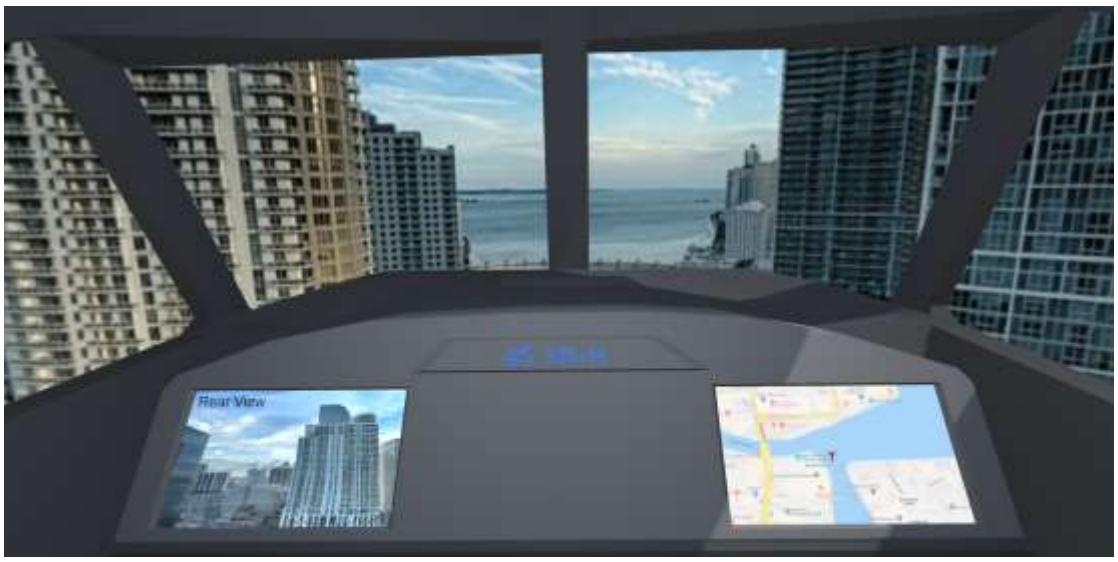


FIGURE 8. The 360-panorama as a skybox and cockpit for a drone in VR

4.2.2 Positioning

Positioning is an essential type of feedback in many applications where devices are controlled from VR. There might be, for example, a factory containing mobile robots. If the VR application is used to control or command those robots, indoor positioning is most likely required to keep track of them.

The virtual cockpit for a drone was illustrated in figure 8 and there is a map view on the right side of the cockpit. To show the real-time position of drone in the map, some kind of outdoor positioning technology is needed.

There are many different technologies for positioning. The global positioning system (GPS) is often used for outdoor positioning. For indoor positioning, there are radio frequency (RF) based systems, such as proximity, Wi-Fi, ultra-wide-band (UWB). Acoustic and infrared (IR) systems can also be used for indoor positioning. (29.)

4.2.3 Sensor values

There are hundreds of different sensors that can be utilized to provide feedback from the real world. Sensor values can be shown to the user in a numerical form, but the real strength of VR is that those values can be converted into another form, such as a three-dimensional visual element, audio or haptic feedback.

A real-time replication of weather in VR is a good example of making use of sensors. A rain sensor could drive directly the amount of rain drops in VR. A photoresistor or other light sensitive sensor could provide information about the brightness of the environment which could be used to adjust the intensity of environmental light. Wind sensors (anemometers) could drive values of the wind system in VR.

Converting a sensor value into haptic feedback in the controller is a great way to inform the user that something has happened. For example, the user might be controlling a robotic arm and the feedback might be the angular values of the arm joints. Those values, of course, will be used to update the pose of the virtual robotic arm, but they can be also used to generate a short haptic pulse to indicate that the maximum angle has been reached or to continuous vibration if the angle is about to reach the maximum value. These kinds of small details are really important from the user experience (UX) point of view.

In the practical implementation of this thesis, the real-time measurement of RF power level values was used to generate a heatmap visualization. The practical implementation is covered in chapter 5.

4.3 Communication between a VR application and physical devices

The communication between VR and physical devices is an essential part of the interaction between VR and the real world, and the communication method or technology should be chosen carefully. At least these three considerations should be kept in mind:

- Maximum allowed latency
- Minimum bandwidth
- Required level of reliability

For example, if the Internet Protocol (IP) based networking system is being used, there are two commonly used transport layer protocols that could be chosen based on the requirements. Those are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). Both can provide a high bandwidth connection, but they differ in reliability and latency. The TCP ensures that the packets are received and in a correct order. Lost packets are re-sent. However, the TCP has a higher latency because the receiver must send an acknowledgement to the sender after each received packet. The UDP on the other hand, does not have any kind of verification that the packets are received successfully, which makes it better in terms of latency but worse in reliability.

An IP-based technology is recommended if the VR system is physically far from the controlled devices. If the VR system is in the same room with the controlled devices, a short distance communication technology, such as Bluetooth or the Universal Serial Bus (USB) could be chosen. The advantage of Bluetooth and the USB compared to the IP is that they do not require configuration at all on the device end, such as connecting to a password protected Wi-Fi router.

If multiple devices are controlled from the same VR environment, there are several ways to implement the communication. Communication to each device can be piped through one API, as shown in figure 9. Another way is to create a connection from each virtual device to its physical counterpart, as shown in figure 10. The first method is preferred if all physical devices are in the same space. It is used in the practical implementation of this thesis, which is covered in chapter 5.

The second method could be used if the devices are located far away from each other and cannot be connected to one common router or API. For example, IoT sensors might be distributed around the city and connected to a VR application using mobile networks. In that case, it is convenient to let each virtual device have a separate connection to its physical counterpart.

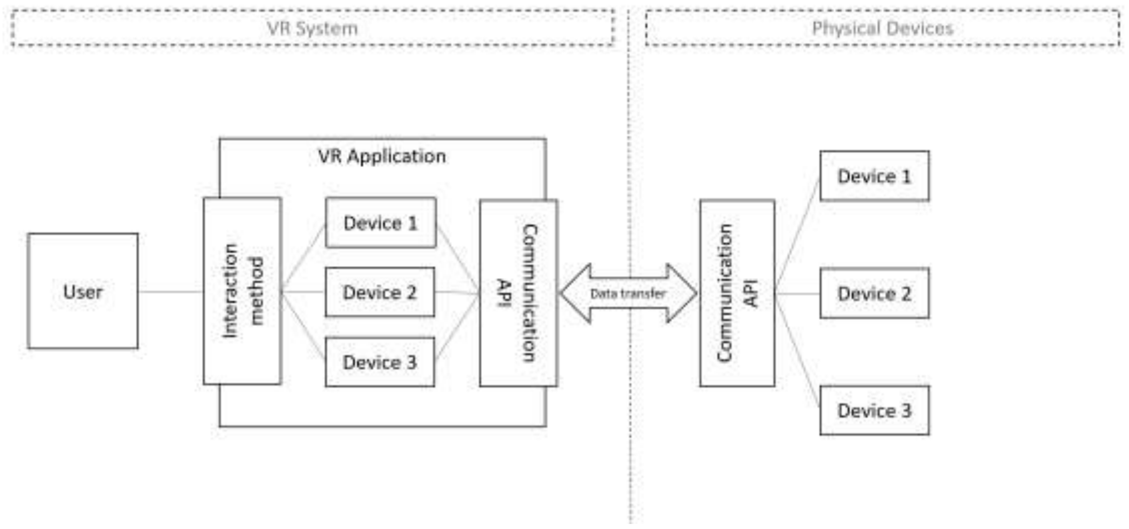


FIGURE 9. The Communication between VR and physical devices through a single API

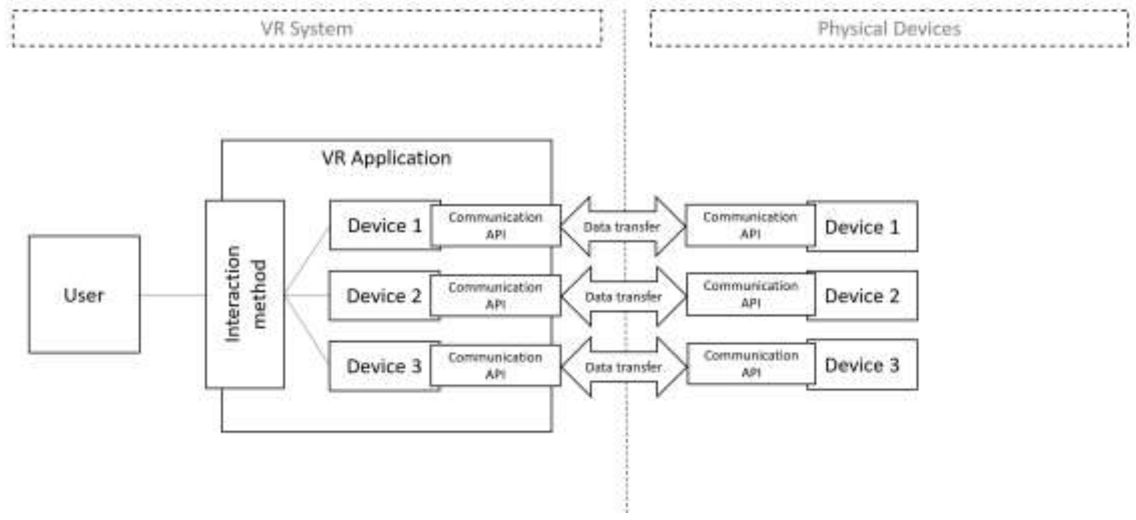


FIGURE 10. The Communication between VR and physical devices through multiple APIs

5 THE WALL USE CASE

Nokia has built an over-the-air test laboratory called Otava at Oulu campus. The main purpose of the laboratory is to test and validate new beamforming base transceiver stations (BTS) and see how they perform in different conditions and with different parameters.

An antenna wall or “The Wall” is one of the most important elements in Otava. It is used to test and validate the performance of the base stations. The wall contains 64 cross-polarized passive antennas. Those antennas are connected to a switchbox behind the wall with a coaxial RF cable. Test mobile phones are connected to another end of the switchbox.

The switchbox can be controlled remotely through a representational state transfer (REST) application programming interface (API). The switchbox has also an ability to measure a received (RX) power level (a signal strength from the BTS) on each antenna on the wall in real-time. Those power level values can be read through the REST API.

Due to the complexity of RF systems and the fact that a human cannot see radio waves, it is difficult to analyse and understand all related numbers and results. For that reason, VR has been taken into use.

One important feature of new 5G base stations is beamforming which is difficult to test in the field conditions. Beamforming allows a BTS to form multiple narrow beams towards users simultaneously, instead of sending with one large beam that covers all users. With the wall and VR, testing and validating these kinds of features is easier.

A VR application called Virtual Otava is a virtual version of the Otava laboratory. Virtual Otava is used to control the physical laboratory, or, more precisely, the wall in the laboratory. The user can connect phones to the antennas on the wall. The state of the switchbox in VR is sent to the real switchbox for synchronization. After synchronization, test phones can communicate with a real BTS over the air through the antennas on the wall. During the communication, the switch-box measures how much power each antenna on the wall is receiving from the BTS and sends that information back to Virtual Otava which then creates a heatmap visualization based on those power levels. The layout of the Otava wall and connection to VR is shown as a block diagram in figure 11.

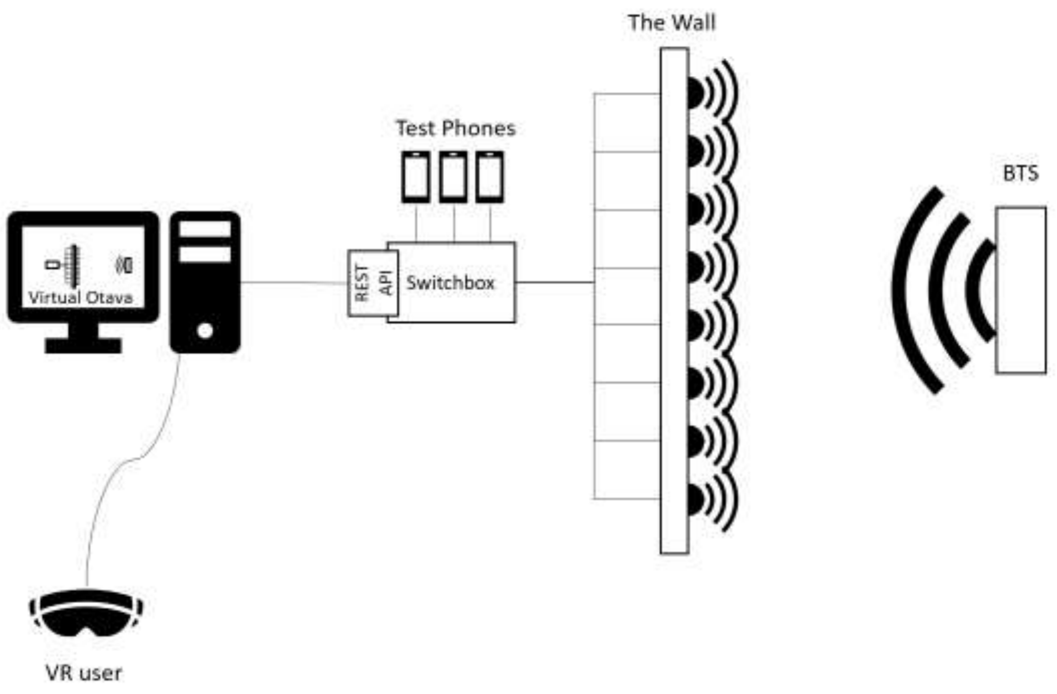


FIGURE 11. The Otava wall block diagram

5.1 Virtual Otava

A virtual version of the Otava laboratory was created using Unity Engine. The target platform is Windows and the preferred VR hardware is HTC Vive. In theory, the environment should also be usable with Oculus Rift and other SteamVR supported hardware. However, all the optimizations and design choices, such as button mappings, were targeted to HTC Vive.

Virtual Otava contains 3D models (shown in figure 12) of the most important elements, such as an antenna wall, a BTS booth and other smaller objects, but an absolute accuracy or realism is not very important in this case. Virtual Otava acts as an intuitive 3D interface to the physical laboratory and therefore, the ease of use was taken priority over the visual appearance.

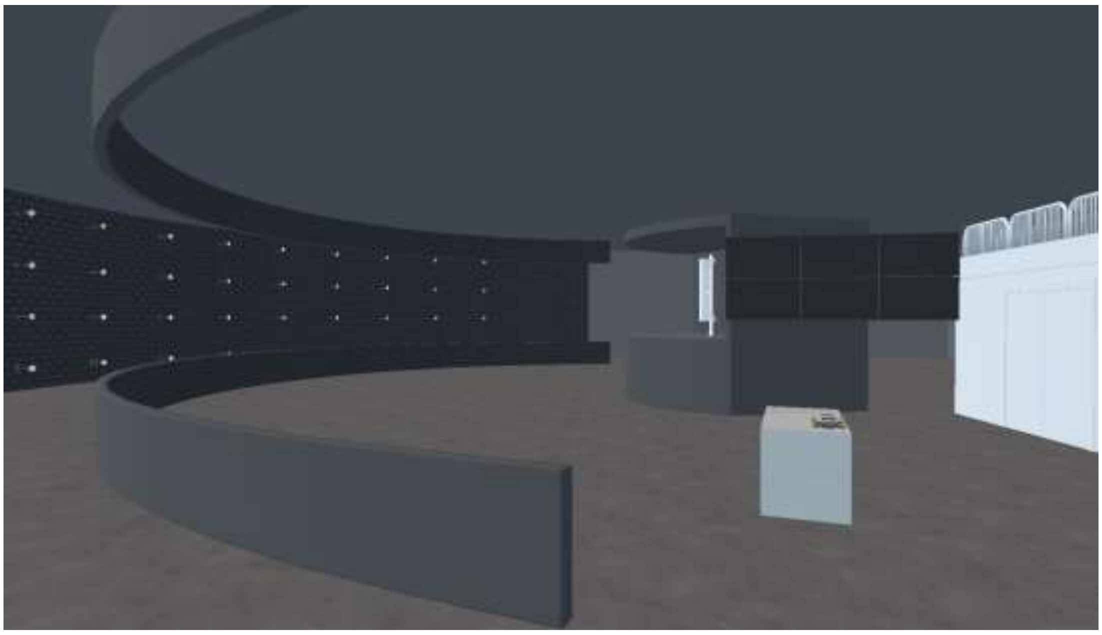


FIGURE 12. Virtual Otava

5.1.1 Virtual Otava functionality

The functionality of the Virtual Otava is rather simple. There are two interactable mobile phones and a drone placed on the table, as seen in figure 13. The user can grab those phones and connect them to the desired antenna on the wall.

The drone was added to allow the user to attach a mobile phone on it and to set it fly on a randomly generated path behind the wall. The drone exists only as a virtual object. There is no physical counterpart for that flying behind the real wall. The purpose of the virtual drone is to simulate moving UE. The phone attached to the drone is always connected to an antenna on the wall that is closest to an imaginary straight line from the phone to the BTS. Such a feature is needed to test how the BTS can handle moving UEs.

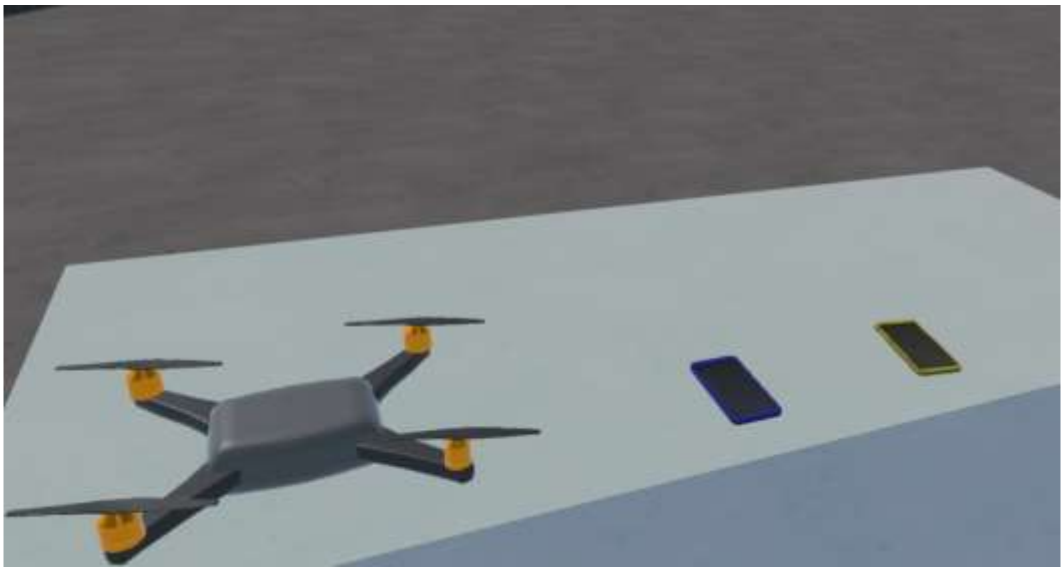


FIGURE 13. Interactable objects

When the user moves their hand close to the interactable phone, yellow outline and button text hint are shown to indicate that the phone can be grabbed, as seen in figure 14. When the user presses the grab button, the hand and controller models will be replaced with the mobile phone. The phone can be released from the hand by pressing the same button again.

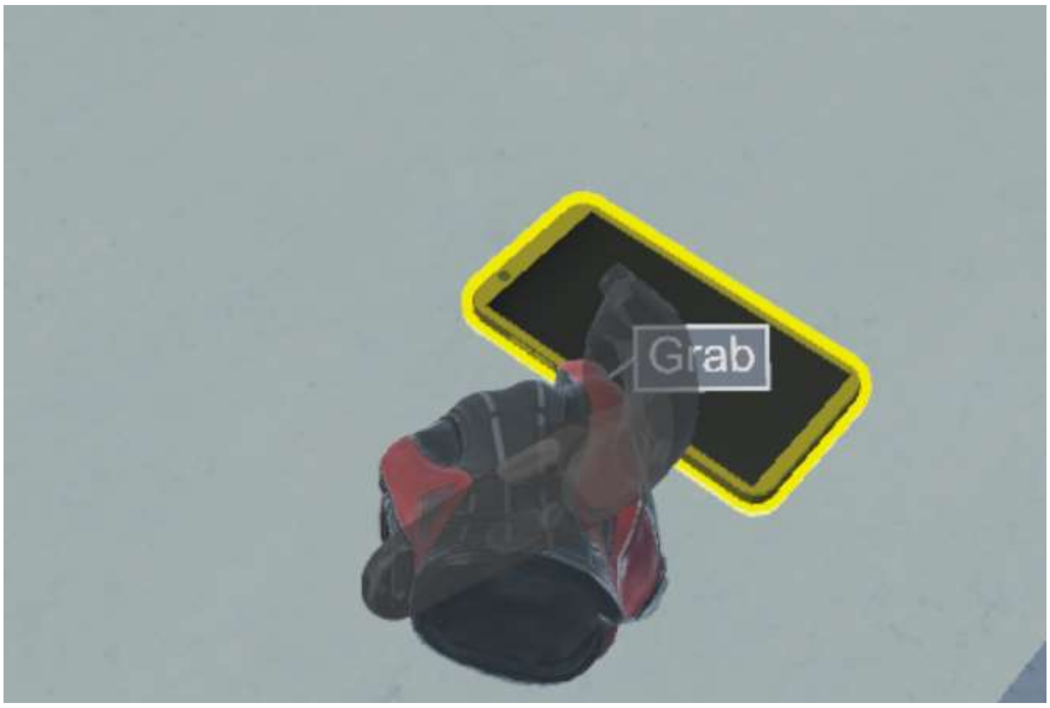


FIGURE 14. The user hovering hand above the interactable mobile phone

Mobile phones can be connected to an antenna on the wall by pointing towards the desired antenna and pressing a controller index finger button which is also known as a trigger. The orange line indicates on which antenna the phone would be connected if the button was pressed, as seen in figure 15. The suggested antenna is always the nearest one to the forward line from the mobile phone. It is found by looping through all antennas on the wall and calculating a cross product of the normalized forward vector of the phone and a vector from the phone to the antenna. Finally, the antenna with the lowest value is selected as a suggestion.

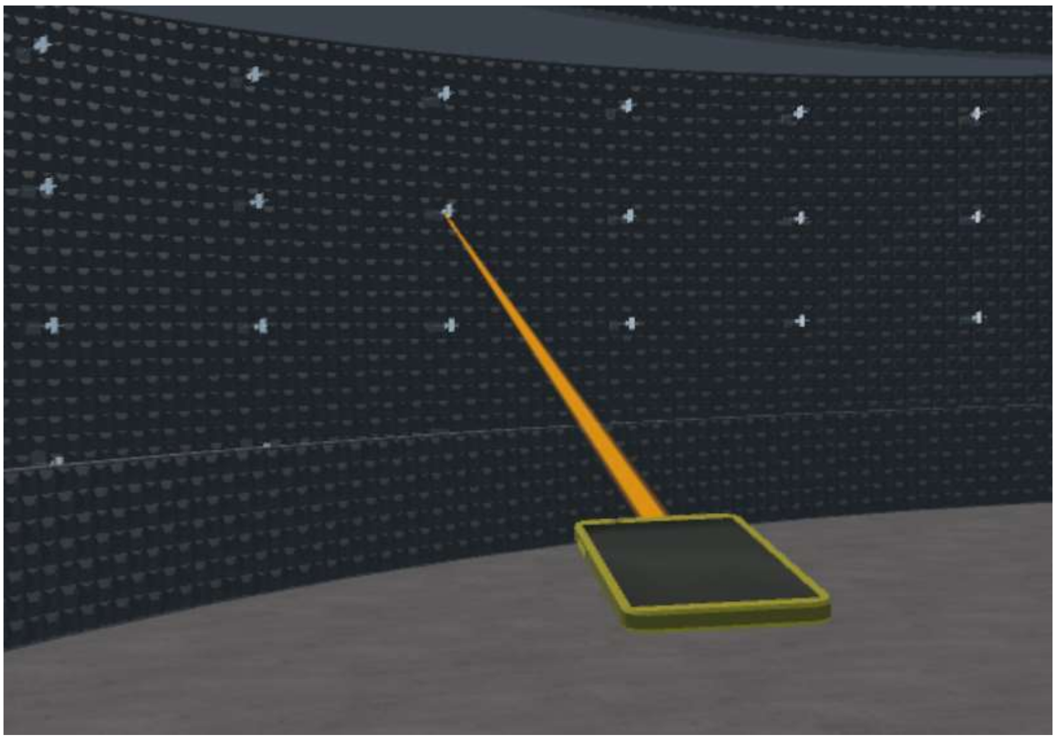


FIGURE 15. The user pointing mobile phone towards an antenna

After the mobile phone is connected to one of the antennas, a dashed line will be drawn from the phone to the antenna and a sine wave is drawn from the antenna to the BTS, as seen in figure 16. The dashed line tells that the phone is transmitting through that specific antenna on the wall. The sine wave is just indicating that the phone is actually communicating with the BTS. A heatmap is drawn on the wall, based on the measured RX power level values that are received from the switchbox. Heatmap visualization can also be seen in figure 16. The communication between the VR application and the switchbox is explained in the chapter 5.1.2 and the heatmap generation is explained in the chapter 5.1.3.

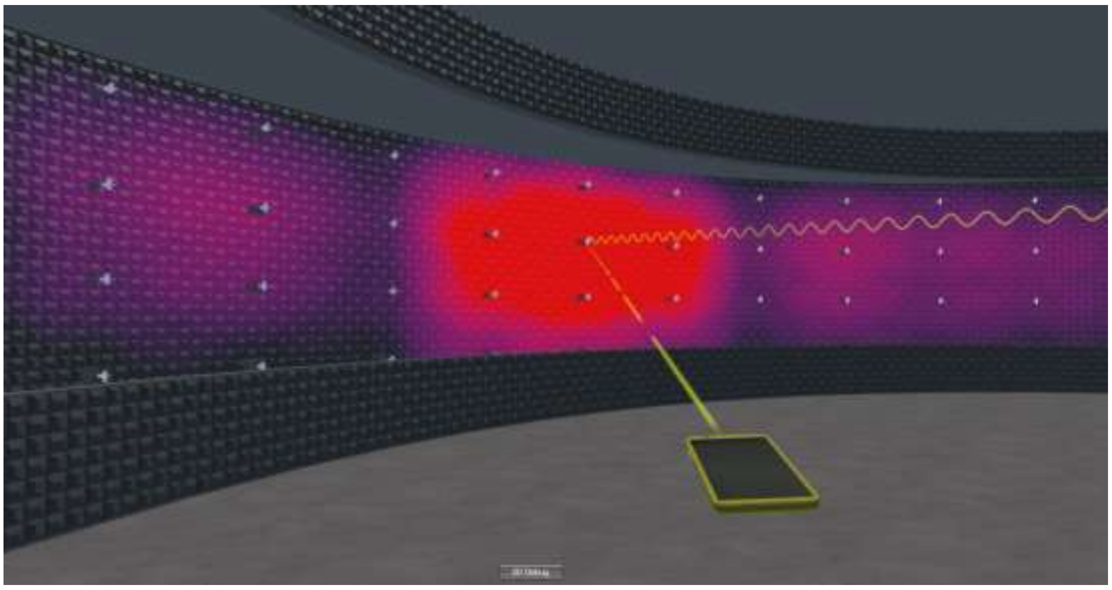


FIGURE 16. A mobile phone connected to a wall antenna

5.1.2 Communication between Virtual Otava and physical switchbox

The first version of the communication layer was implemented using a simple string-based messaging protocol built on top of the TCP. The messaging protocol was defined by Otava developers. However, it was later changed so that all the communication between Virtual Otava and the wall switchbox is handled through the REST API. Different parts of the Virtual Otava are illustrated in figure 17.

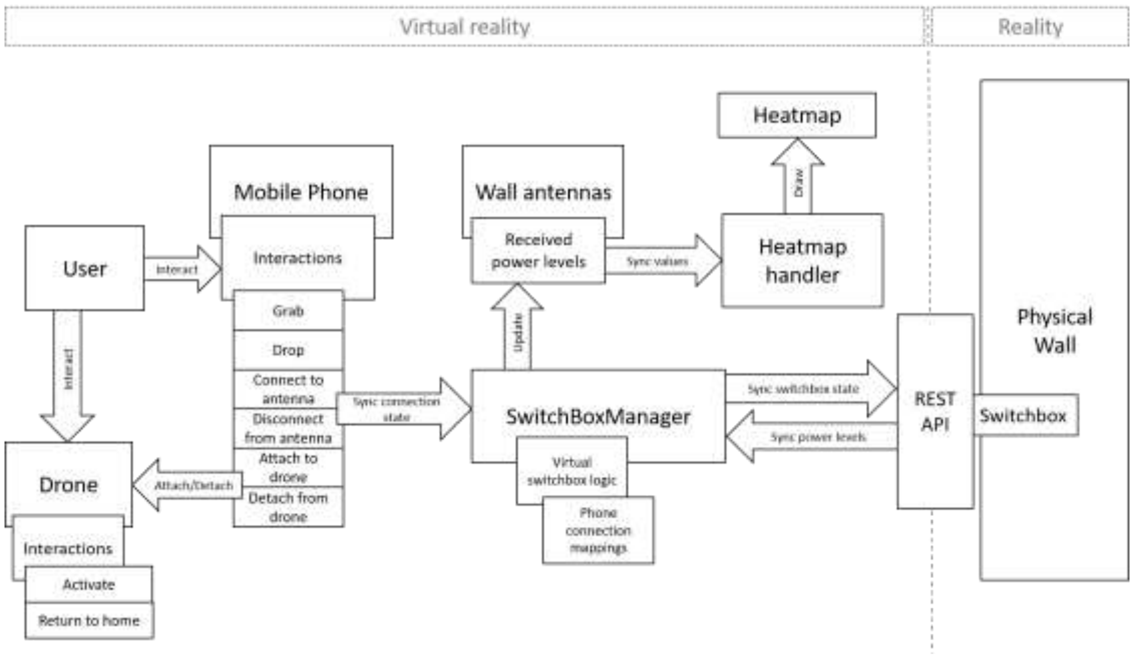


FIGURE 17. The Virtual Otava block diagram

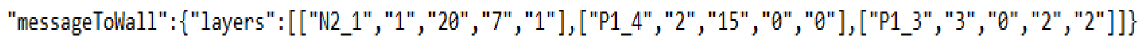
A GameObject with a SwitchBoxManager component is placed into the Virtual Otava scene. Its responsibility is to keep track of which phone is connected to which antenna on the wall. Whenever the user connects or disconnects a phone from the antenna, a *Connect* / *Disconnect* method inside the SwitchBox-Manager is invoked. The SwitchBoxManager is also responsible for sending state updates to the physical switchbox through the REST API at specific inter-vals. The last responsibility of the SwitchBoxManager is to fetch RX power level values from the switchbox and update them. There is a component called Probe in each GameObject that represents an antenna on the wall. A variable called *rxPowerLevel* is declared in the Probe component and is updated by the SwitchBoxManager.

The switchbox REST API has two endpoints needed by Virtual Otava. Those are *setWallState* which is called using a PUT request and *getRFPowers* which is called using a GET request. GET, PUT and other request methods are speci-fied in the Hypertext Transfer Protocol (HTTP) standard (30). All requests and their responses are converted into the JavaScript Object Notation (JSON) for-mat before sending.

Payload for the *setWallState* request is an array of layers. One layer defines which phone is connected to which antenna, the amount of attenuation to the signal between a phone and an antenna, a phone identifier and switch configuration. An example of possible payload is shown in table 1. It would connect three mobile phones to different antennas on the wall. The same payload also converted into the JSON format is illustrated in figure 18. The response for the *setWallState* request is just “done” or “error”.

TABLE 1. An example of setWallState request payload.

Switch Config	**Phone ID**	**Attenuation**	**Position X**	**Position Y**
N2_1	1	20	7	1
P1_4	2	15	10	0
P1_3	3	0	2	3



```
"messageToWall":{"layers":[["N2_1","1","20","7","1"],["P1_4","2","15","0","0"],["P1_3","3","0","2","2"]]}
```

FIGURE 18. An example of setWallState request payload in the JSON string format

The *getRFPowers* request, as the name suggests, is a GET request. It does not have any payload. A response for that request is a two-dimensional float array of measured power levels in dBm.

5.1.3 Heatmap generation

The original idea was to generate a 3D model of the antenna beam, based on the feedback from the wall, but it was quickly discovered that it did not work very well in closed space, such as the Otava environment. For that reason, the heatmap solution was taken into use.

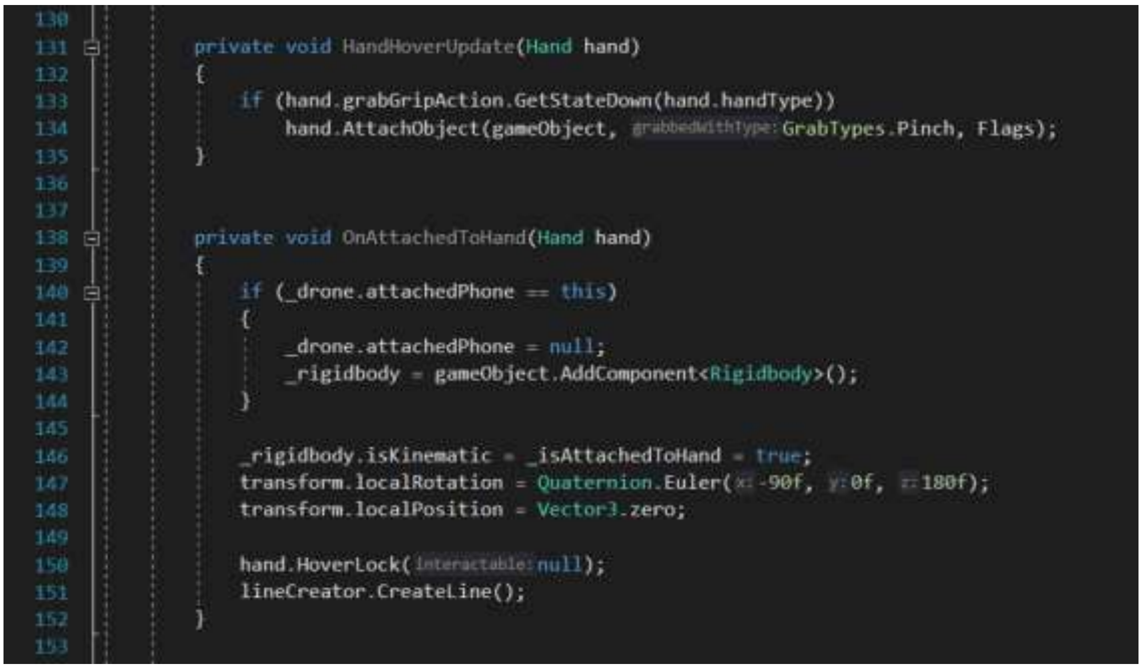
When the RX power level on each antenna is received from the switchbox, it can be used to generate a heatmap on the surface of the wall. The main purpose of the heatmap is to visualize how much power each part of the wall is receiving from the BTS. For example, the user might want to test a new beamforming algorithm. If the mobile phone is transmitting from the antenna which is located on the left corner of the wall, the strongest signal from the BTS should also be detected on the left corner. The heatmap is an easily understandable way to visualize and verify that. The visualization of the heatmap was shown in figure 16.

The Heatmap is generated with a shader thus it is important to know how shaders work in Unity in order to understand how the heatmap works. Shaders were explained in the chapter 3.5. The heatmap shader takes two arrays as an input. The first array contains position vectors of the wall antennas and the second array contains received power levels of those antennas. The shader calculates a pixel color by looping through those arrays. The closer the position is to the pixel and the higher the power level, the more the pixel intensity is increased.

There is also a C# script which updates the power level array for the shader at specific interval. Passing data for the shader is an expensive operation thus the update rate is limited to 10 updates per second. Power levels are normalized into a range between 0 and 1 before an update.

5.1.4 Code sample for grabbing phones

As mentioned in the chapter 5.1.1, there are many different interactions included in Virtual Otava. The SteamVR Unity plugin is used as a core for the interactions. SteamVR is an API which provides interfaces for developing VR interactions and getting inputs from most of the popular VR devices (31). A small sample of the script of an interactable phone is shown in figure 19. That part of the script reveals how the grabbing of the phone was implemented.



```
        private void HandHoverUpdate(Hand hand)
        {
            if (hand.grabGripAction.GetStateDown(hand.handType))
                hand.AttachObject(gameObject, grabbedWithType: GrabTypes.Pinch, Flags);
        }


        private void OnAttachedToHand(Hand hand)
        {
            if (_drone.attachedPhone == this)
            {
                _drone.attachedPhone = null;
                _rigidbody = gameObject.AddComponent<Rigidbody>();
            }

            _rigidbody.isKinematic = _isAttachedToHand = true;
            transform.localRotation = Quaternion.Euler(x: -90f, y: 0f, z: 180f);
            transform.localPosition = Vector3.zero;

            hand.HoverLock(interactable: null);
            lineCreator.CreateLine();
        }
```

FIGURE 19. C# code for grabbing a phone with a hand

The HandHoverUpdate function is called repeatedly when one of the hands is near the phone. The purpose of that function is to check if the grip-button of the controller was pressed down during the last frame and to attach the phone to the hand.

Another function in the script is *OnAttachedToHand*. It is called after the phone was attached to the hand. *OnAttachedToHand* is responsible for clearing a possible reference from the drone object, temporarily shutting down physics from the phone, setting the position of the phone and the rotation on the hand and finally creating a line which was illustrated in figure 15.

Both functions are called from the player’s hands and the player GameObject is provided as a prefab in the SteamVR Unity plugin.

6 CONCLUSION

The main objective of this thesis was to study how physical devices can be controlled remotely from virtual reality and create an antenna wall controlling system as a practical implementation.

As a result, this thesis provides basic guidelines on what kinds of things should be considered in making a virtual reality control interface for physical devices. A comprehensive set of examples are provided from different input, feedback and communication methods.

The practical implementation was started by learning how the newest version of SteamVR works. After that, VR interactions and 3D models were implemented. The final part was to create a heatmap system and implement the communication module. It was necessary to consult creators of the physical wall to get understanding how it works and what kinds of APIs were built on it. It was also important to get familiar with the basics of the BTS features and some theory related to RF.

The VR application was created, and the requirements were met but testing with the real wall was not possible due to the change of communication API on the wall control box. New API specifications were received from the wall developers and the communication module of the practical implementation was built based on that. However, the new API was not yet implemented on the wall control box, thus the VR application had to be tested using a simulator application with real measured data and it worked as expected. Testing with the real wall should be done after the new API is finished on the wall control box.

The VR application could be further improved by implementing communication with the base station and test phones. Those would provide plenty of useful data to play with. Also, some kind of access management system would be probably needed because in a current form, basically anyone in the company could launch the VR application and command the wall simultaneously.

REFERENCES

1. Arvanaghi, B. & Skytt, L. 2018. Virtuaalitodellisuus – tulevaisuus on täällä tänään. Date of retrieval 27.5.2019 https://tieku.fi/teknologia/vempaimet/virtuaalitodellisuus

2. Nokia. Historiamme. Date of retrieval 27.5.2019 https://www.nokia.com/fi_fi/tietoa-meista/keita-olemme/historiamme/

3. Nokia. What we do. Date of retrieval 27.5.2019 https://www.nokia.com/about-us/who-we-are/what-we-do/

4. Woodford, C. 2019. Virtual reality. Explainthatstuff. Date of retrieval 27.5.2019 https://www.explainthatstuff.com/virtualreality.html

5. Buckley, S. 2015. This Is How Valve's Amazing Lighthouse Tracking Technology Works. Date of retrieval 27.5.2019 https://gizmodo.com/this-is-how-valve-s-amazing-lighthouse-tracking-technol-1705356768

6. Wired. Review: HTC Vive Pro. Date of retrieval 28.5.2019 https://www.wired.com/review/review-htc-vive-pro/

7. VR Today Magazine. Top 5 Amazing VR Gadgets To Try In 2019. Date of retrieval 27.5.2019 https://vrtodaymagazine.com/vr-gadgets/

8. Vive. VIVE TRACKER GO BEYOND VR CONTROLLERS. Date of retrieval 28.5.2019 https://www.vive.com/us/vive-tracker/

9. Unity. The world’s leading real-time creation platform. Date of retrieval 27.5.2019 https://unity3d.com/unity

10. Unity Technologies. Unity for VR and AR. https://unity3d.com/unity/features/multiplatform/vr-ar

11. Dillet, R. 2018. Unity CEO says half of all games are built on Unity. Date of retrieval 27.5.2019 https://techcrunch.com/2018/09/05/unity-ceo-says-half-of-all-games-are-built-on-unity/

12. Unity. Asset Workflow. 2018. Date of retrieval 27.5.2019 https://docs.unity3d.com/Manual/AssetWorkflow.html

13. Unity. GameObjects. 2017. Date of retrieval 27.5.2019 https://docs.unity3d.com/Manual/GameObjects.html

14. Unity. Transform. Date of retrieval 27.5.2019 https://docs.unity3d.com/Manual/class-Transform.html

15. Unity. Using Components. 2018. Date of retrieval 27.5.2019 https://docs.unity3d.com/Manual/UsingComponents.html

16. Unity. Creating and Using Scripts. 2018. Date of retrieval 27.5.2019 https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html

17. Unity. Order of Execution for Event Functions. 2018. Date of retrieval 27.5.2019 https://docs.unity3d.com/Manual/ExecutionOrder.html

18. Unity. Scenes. 2017. Date of retrieval 27.5.2019 https://docs.unity3d.com/Manual/CreatingScenes.html

19. Unity. Multi-Scene editing. 2017. Date of retrieval 27.5.2019 https://docs.unity3d.com/Manual/MultiSceneEditing.html

20. Unity. Materials, Shaders and Textures. 2017. Date of retrieval 27.5.2019 https://docs.unity3d.com/Manual/Shaders.html

21. Unity. Skybox. Date of retrieval 27.5.2019 https://docs.unity3d.com/Manual/class-Skybox.html

22. Unity. Prefabs. 2018. Date of retrieval 27.5.2019 https://docs.unity3d.com/Manual/Prefabs.html

23. La Valle, S. M. 2019. Virtual reality. Date of retrieval 28.5.2019 http://vr.cs.uiuc.edu/vrch10.pdf

24. Colgan, A. 2014. How Does the Leap Motion Controller Work? Date of retrieval 28.5.2019 http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/

25. Emotiv. The Science behind our technology. Date of retrieval 28.5.2019 https://www.emotiv.com/our-technology/

26. Emotiv. Homepage. Date of retrieval 28.5.2019 https://www.emotiv.com/

27. Unity. PhraseRecognitionSystem. Date of retrieval 28.5.2019 https://docs.unity3d.com/ScriptReference/Windows.Speech.PhraseRecognitionSystem.html

28. Nascimento, R. 2019. MIAMI 360 AERIAL PANORAMA KEY. Date of retrieval 28.5.2019 https://pixexid.com/free-photo/295-miami-360-aerial-panorama-key

29. Ray, B. 2018. How An Indoor Positioning System Works. Date of retrieval 28.5.2019 https://www.airfinder.com/blog/indoor-positioning-system

30. Mozilla. HTTP request methods. 2019. Date of retrieval 28.5.2019 https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

31. STEAMVR. SteamVR Unity Plugin. Date of retrieval 28.5.2019 https://valve-software.github.io/steamvr_unity_plugin/#documentation