UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

FINAL DEGREE THESIS

Mechanical engineering degree

VIRTUAL MIRROR: KINECT AND UNITY INTEGRATION

Memory and annexes

Author: Albert Ruiz Gracia
Director: Jordi Torner Ribe
Co-Director: Gil Serrancolí Masferrer
Convocatòria: October 2018

Abstract

Virtual reality (VR) technology is every day becoming a more popular application for physical rehabilitation and motor control research. The goal of this project is to help with the rehabilitation of patients with injuries that affect their mobility or people with movement disability, remotely. There are studies that show that the visualisation of our own movements in an augmented way, nourishes our brain positively and accelerates the recovery.

This Kinect-based rehabilitation application consists in accessing the Kinect body joints orientations defined with quaternion and apply them, through a code in Visual Studio (VS), to our avatar in Unity. Kinect is our input that captures the trajectories of skeleton points. Those data are received in Unity, a game development engine. We then wrote a code in VS to process those data and, once the joint orientations were processed, we were able to apply them into an avatar in Unity so that it reproduces our movements in real-time.

The project opens a door to a wide variety of future medical applications since a full human body is tracked and can help with any possible avant-garde rehabilitation techniques.

Resumen

La tecnología de realidad virtual (VR) se está convirtiendo cada día en una aplicación más popular para la rehabilitación física y la investigación de control motor. El objetivo de este proyecto es ayudar a rehabilitar a distancia a los pacientes con lesiones que afectan su movilidad o personas con discapacidad motriz. Hay estudios que muestran que la visualización de nuestros propios movimientos de una manera aumentada, nutre nuestro cerebro positivamente y acelera la recuperación.

Esta aplicación de rehabilitación basada en Kinect consiste en acceder a las orientaciones de las articulaciones del cuerpo de Kinect definidas con cuaterniones y aplicarlas, a través de un código en Visual Studio (VS), a nuestro avatar en Unity. Kinect es nuestra entrada que captura las trayectorias de los puntos del esqueleto. Esos datos se reciben en Unity, un motor de desarrollo de juegos. Luego escribimos un código en VS para procesar esos datos y, una vez que almacenamos las orientaciones de las articulaciones, pudimos aplicarlos en un avatar en Unity para que reproduzca nuestros movimientos en tiempo real.

El proyecto abre la puerta a una amplia variedad de aplicaciones médicas futuras ya que se hace un seguimiento de todo el cuerpo humano y puede ayudar con cualquier técnica de rehabilitación de vanguardia.

Resum

La tecnologia de realitat virtual (VR) es converteix cada dia en una aplicació més i més popular per a la investigació de rehabilitació física i control del motor. L'objectiu d'aquest projecte és ajudar a la rehabilitació de pacients amb lesions que afecten la seva mobilitat o persones amb discapacitat de moviment, de manera remota. Hi ha estudis que mostren que la visualització dels nostres propis moviments d'una manera augmentada, nodreix el nostre cervell de forma positiva i accelera la recuperació.

Aquesta aplicació de rehabilitació basada en Kinect consisteix a accedir a les orientacions de les articulacions del cos de Kinect definides amb quaternions i aplicar-les, a través d'un codi en Visual Studio (VS), al nostre avatar en Unity. Kinect és la nostra entrada que captura les trajectòries dels punts d'esquelet. Aquestes dades es reben a Unity, un motor de desenvolupament de jocs. A continuació, vam escriure un codi en VS per processar aquestes dades i, un cop emmagatzemades les orientacions de les articulacions, vam poder aplicar-les a un avatar en Unity perquè reproduís els nostres moviments en temps real.

El projecte obre una porta a una àmplia varietat de futures aplicacions mèdiques, ja que es fa un seguiment de tot un cos humà i pot ajudar-se amb qualsevol possible tècnica de rehabilitació avantguardista.

Thanks

At first, I didn't know much about Virtual Reality and its possible application on the rehabilitation field. That is why I would like to thank Jordi Torner and Gil Serrancolí for this offer and for transmitting me the passion for developing a project with a powerful purpose and with a wide range of improvement. Also thanks to Gil Serrancolí for giving me access to the SIMMA Lab and for providing me with the necessary equipment of this project.

Finally I would like to thank family and friends. They gave me a lot of support during the project when I needed the most and they encouraged me to do my best every single day. I specially want to thank my partner, Patricia Moyano, for her patience and ability to transmit motivation day by day.

Thank you.

Figure list

INTRODUCTION

Figure 1.1. Virtual Reality evolution..........2

KINECT

Figure 4.1. Kinect parts (Frontal view/Kinect OFF)..........8

Figure 4.2. Kinect parts (3D view/Kinect ON)..........9

Figure 4.3. Distance measurement of ToF cameras..........10

Figure 4.4. Kinect interaction with an application..........11

UNITY

Figure 5.1. Gimbal lock problem..........13

Figure 5.2. 3D skeleton joints tracked..........14

Figure 5.3. Kinect skeleton joints hierarchy..........14

Figure 5.4. Unity 3D display..........16

Figure 5.5. Unity avatar scene display..........17

Figure 5.6 Configure Avatar mode..........18

Figure 5.7. Avatar Mapping..........18

EVALUATION

Figure 6.1. Kinect commands for Unity projects..........19

Figure 6.2. Body joints dictionary..........20

Figure 6.3. Body tracked condition..........20

Figure 6.4. Auto mapped Joint GameObjects..........20

Figure 6.5a. Access to joint Kinect orientations..........23

Figure 6.5b. Access to joint Kinect orientations....24

Figure 6.6a. BodySourceManager script 26

Figure 6.6b BodySourceManager script....27

Figure 6.7. Joint transforms function....28

Figure 6.8. Avatar joint GameObjects....29

Figure 6.9. Quaternion application code on avatar Neck example....30

Figure 6.10 Kinect skeleton local coordinate system....31

Figure 6.11 SpineMid Unity coordinate system....31

Figure 6.12 Coordinate system rotated....31

Figure 6.13 Coordinate systems adaptation code....32

RESULTS

Figure 7.1 Orientations array....34

Figure 7.2 Txt for loop....34

Figure 7.3 Orientations array txt (One frame)....35

Table list

KINECT

Table 4.1. Kinect specifications....................10

UNITY

Table 6.1. Unity and Kinect joints....................22

ECONOMICAL ANALYSIS

Table 9. Acquisition project costs....................44

Table 10. Project realization costs....................45

Table 11. Total project costs....................45

Glossary

API	Application programming interface
VS	Visual Studio
MOCAP	Motion capture
RGB	Red, Green and Blue
IDE	Integrated development environment
AR	Augmented reality
VM	Virtual Mirror
VR	Virtual Reality
SIMMA Lab	Simulation and Movement Analysis Laboratory
PC	Personal Computer

Index

ABSTRACT ____ I

RESUMEN ____ II

RESUM ____ III

THANKS ____ V

FIGURE LIST ____ VII

TABLE LIST ____ IX

GLOSSARY ____ XI

1. PREFACE ____ 1
   - 1.1. Motivation 3
   - 1.2. Previous requirements 3

2. INTRODUCTION ____ 5
   - 2.1. Objectives 5
   - 2.2. Scope 6

3. STATE-OF-THE-ART ____ 7

4. KINECT SENSOR ____ 8
   - 4.1. Sensor characteristics 9
     - 4.1.1. Kinect software development kit for Windows (SDK) 11

5. APPLICATION PROGRAMMING INTERFACE (API) ____ 12
   - 5.1. Kinect 13
   - 5.2. Unity 3D 15
     - 5.2.1. Avatar set up 17

6. EVALUATION ____ 19
   - 6.1. Kinect and Unity integration 19
   - 6.2. Kinect 20
     - 6.2.1. Main script: BodySourceView 20
     - 6.2.2. BodySourceManager script 25
   - 6.3. Avatar movements 28
     - 6.3.1. Coordinate systems adaptation 30

7. RESULTS ____ **34**

8. ENVIRONMENT IMPACT ANALYSIS ____ **37**

9. IMPROVEMENT PROPOSALS AND FUTURE APPLICATIONS ____ **39**

CONCLUSIONS ____ **41**

ECONOMIC ANALYSIS ____ **44**

BIBLIOGRAPHY ____ **46**

ANNEX A ____ **49**

A1. Kinect specifications 49

A2. Body Source View script 52

A3. Body Source Manager script 76

1. Preface

Technology has changed the way we communicate, listen to music, exercise, do the shopping, play games and much more. It is not surprising that technology has an impact on the healthcare industry. Healthcare technology is helping people live longer, reducing wait times and making it easier for doctors to diagnose diseases.

Here is where a directed telerehabilitation system based on new technologies of virtual reality plays a role. The first thought about Virtual Reality is to turn to a modern VR headset as well as all of the various PC applications which are beginning to include virtual-reality support. Virtual-reality actually has an extensive history with a concept that dates all the way back to the 1930s and developed until what we nowadays know as Virtual Reality as shown in *figure1.1*.

It was in the 1935 when Stanley G. Weinbaum, an American science fiction writer, created a story called Pygmalion's Spectacles. In the story, the main character, Dan Burke, met an elfin professor, Albert Ludwig, who invented a pair of goggles which as Stanley said, "enabled a movie that gives one sight and sound taste, smell, and touch. You are in the story, you speak to the shadows (characters) and they reply, and instead of being on a screen, the story is all about you, and you are in it" (1).

Later in 1968 Ivann Sutherland, with the help of his student Bob Sproull, created the first VR head mounted display system. It was simple and primitive, and it was used only for military purposes but it was the first approach and the starting point to the VR headsets that are used nowadays.

The next milestone was when it was first introduced to the masses. In 1995, Nintendo created "Virtual Boy", it was the first portable game console capable of displaying "true" 3D graphics. It was reported by many that prolonged use of the Virtual Boy gave you headaches and some even claimed it to induce seizures, so less than a year since its launch it was discontinued.

Finally, in April 2012, Luckey announced a virtual reality headset designed for video gaming, and launched a Kickstarter campaign to make virtual reality headsets available to developers. Then, in 2014, Facebook CEO Mark Zuckerberg agreed to acquire Oculus VR. Later on, many uses would be given to Virtual Reality, not only for gaming purposes, but for 3D modelling or even telerehabilitation for patients with movement disability.

In the past years, parallel to VR progress, low-cost depth-sensing cameras used for VR purposes, have also become commercially available, including the well-known Microsoft Kinect, which have made it possible to sense the full-body pose for multiple users without the use of markers or handheld devices (2). As we know, VR also has continued its development until the point of having a wide range of applications.

When a technology is so developed and studied as VR is nowadays, we must go further on the technology's applications. We need to focus on applications that can make a real difference in this world and improving people's life is one way of achieving it. This is how our project was born, by the necessity of helping people's health through a remote rehabilitation application. A positive input is sent to our brain by seeing our own movements in an augmented way.

With the only need of a depth-sensing camera, a game development software and VR glasses we can develop a application to accelerate the rehabilitation of people with physical impairment or just physical injuries.

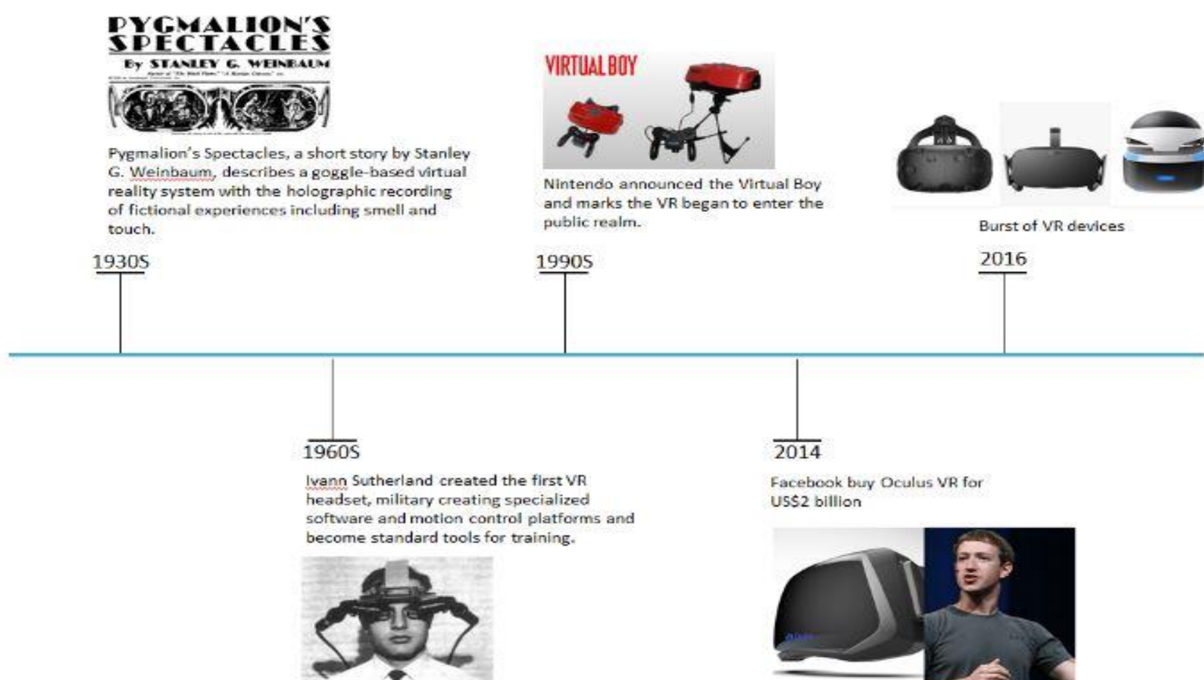

***Figure 1.1** Virtual Reality evolution*

1.1. Motivation

One of the main objectives is offering patients a remote rehabilitation therapy without the need of medical assistance. At the same time, it is possible to record these data for a later patient analysis that will help to a patient progression control. Nowadays is really important to make those technologies and applications affordable for everyone and we can ensure that this low-cost VR application can ensure good results as multiple projects have been carried out on this topic as explained on state-of-the-art.

1.2. Previous requirements

This Kinect-based application for VR, as mentioned before, as a low-cost application, not many requirements are needed. The Kinect v2 developed by Microsoft was used to capture the body joints data and the software was written in the C# programming language and developed by the Unity3D video game engine. The software used list in this project is:

- Software development kit 2.0 (SDK)
- Visual Studio 2017
- Unity 3D
- Windows 8

2. Introduction

Rehabilitation technology can allow patients with movement disability exercise at home under supervision of their rehabilitation team. Currently it is unclear how effective this approach is. Patients who are assisted by new rehabilitation technology at home are expected to have better fitness and less symptoms. This approach can be extended to people with different diseases related to mobility impairment and it can be used not only for physical but also for cognitive and occupational rehabilitation. The tracking of human movement is also implemented on the analysis of sport athletes performances in order to improve their results or just for clinical purposes (3).

In this project we present the first step to develop an application to help people's rehabilitation by interacting with VR. It is proved that it feeds the brain with positive inputs which can accelerate the person's recovery. The project is divided in three parts: the Kinect integration to its use, coding the body movements in C# in Visual Studio (VS) and the load all the gestures data from VS into an avatar in Unity 3D.

2.1. Objectives

This project is a Kinect-based rehabilitation application for patients with mobility impairment or injuries that affect the person's mobility. It represents the first step into developing a full VR game where patients could see themselves in a daily life environment through VR headsets.

The main objective of this thesis is to animate an avatar in real time by following our own movements. In order to do this, Microsoft Kinect V2 will be used as a motion capture (MOCAP) sensor, which allows you to obtain digitally the position and orientation, in the three-dimensional space, of the different anatomical points of the subject in each frame, at a speed of 30 fps (frames per second). In each frame, these data are sent to the avatar model in order to reproduce in real time the subject body position captured by the sensor. Through Visual Studio (VS) we are able to code, in C#, the functions needed in order to integrate these data acquired by the sensor and apply them into the avatar joints in Unity so it reproduces our desired movements. Unity framework is used to implement our system because it enables us to use virtual reality techniques to see detailed movements of the patient.

The key contribution of our research is a rule-based approach to real-time exercise quality assessment and feedback.

The partial objectives of this project:

- Obtaining the desired body joints orientations from our Kinect sensor
- Establish connection between Kinect and Unity. Connecting those different API is a must when trying to develop an application in VR. We have to be able to call our Unity GameObjects by defining some functions
- Real-time avatar movements following the subject gestures, as if it was a mirror, by applying the quaternions acquired from the Kinect into our avatar

2.2. Scope

This project is the first part into developing a full VR application for rehabilitation purposes. It is based on acquiring the data from the Kinect sensor, processing these data, integrating Kinect with Unity thanks to the "Unity Pro Package", applying these data on the avatar and, in real-time, watch your own movements in your PC displayed through an avatar.

However more improvements and further research has to be made. Future students' projects will take this project as a starting point and continue its development by integrating this system into a VR environment to achieve the user immersion. A HMD will be used through which the VM will be rendered with the avatar, along with a suitable and modifiable virtual environment that can influence positively on the evolution of the affected subjects.

3. State-of-the-art

Because of their attractiveness and potential, several studies have been dedicated to Motion Capture (MOCAP) cameras and its integration on avatars for VR purposes. Motion capture techniques are used over a broad field of applications, ranging from gaming animations for entertainment to biomechanics analysis for clinical and sports applications. Due to some comparisons with other optical motion capture system, it is known that Kinect offers enough precision for most applications (4) so we can ensure a better quality control process for example for our patient rehabilitation application.

In the field of rehabilitation technologies previous projects have been done. Mainly what those show is that this technology is used to reduce staff and enhance participants' motivation, interest and perseveration. The participants on those studies significantly increased their motivation for physical rehabilitation, so they improved their exercise performance (5),(6). However our project is more focused on working with the patient's brain in terms of seeing each other own movements augmented so that they receive a positive input and accelerates the rehabilitation. Other studies are focuses on developing also a rehabilitation application but with emphasis on adults with neurological injuries (7). Their main goal following spinal cord injury (SCI) and traumatic brain injury (TBI) is to promote a maximal level of recovery. Full reintegration into the community are the ultimate goals Our project is related to the neurological injuries applications but we don't acquire that level of detail as we work not only for those injuries but for people with mobility impairment or simply physical injuries rehabilitations.

On the other hand, there are applications that also combine a simple motion capture (MOCAP) camera with Unity or other 3D software but are more focused on the user experience than any health care connotation. All those projects, although its objectives, have the same background as they collect data from Kinect, they process those data and send information to a software. Some of them, as mentioned before, are focused on improving athletes' performance. They describe a novel system that automatically evaluates for example dance performances against a gold-standard performance and provides visual feedback to the performer in a 3D virtual environment. The system acquires the motion of a performer via Kinect-based human skeleton tracking, making the approach viable for a large range of users (8).

4. Kinect sensor

The Kinect sensor is motion capture devices based on a webcam-style add-on peripheral. It enables users to control and interact with their console/computer without the need for a game controller, through a natural user interface using gestures and spoken commands. Kinect is a marker less and low-cost technology that guaranties enough precision for most applications like rehabilitations treatments. It is composed of two cameras, a Red-Green-Blue (RGB) camera equipped with a standard complementary metal–oxide–semiconductor (CMOS) sensor through which the coloured images of persons and objects are acquired, and an infrared (IR) camera (Figure 4.1). The IR emitters and the IR camera form the 3D sensor are shown in Figure 4.2.

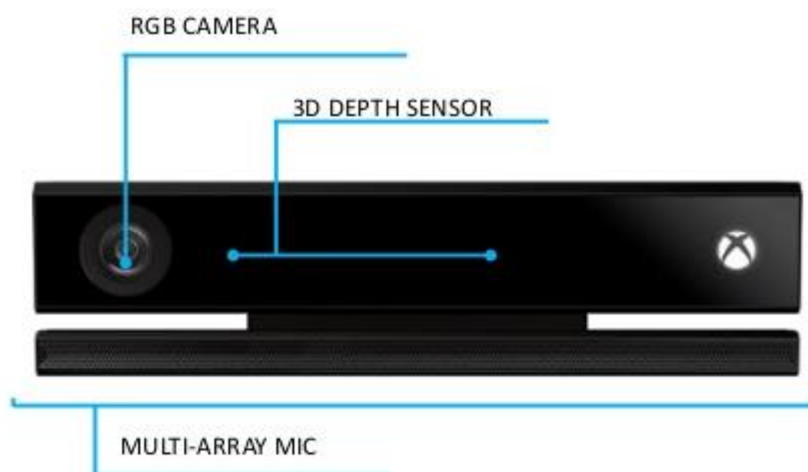

Figure 4.1 *Kinect parts (Frontal view/Kinect OFF)*

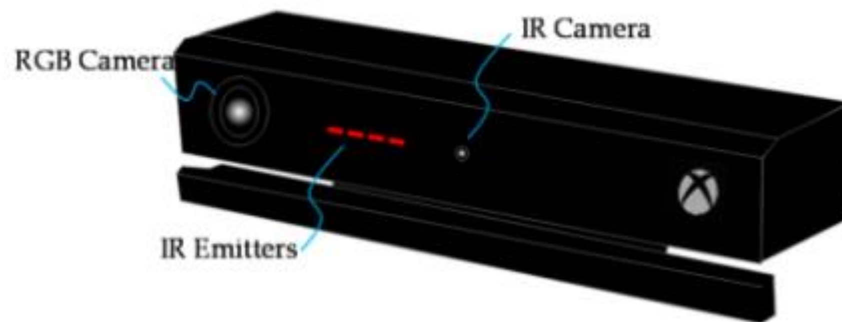

Figure 4.2 *Kinect parts (3D view/Kinect ON)*

4.1. Sensor characteristics

The IR sensor is based on Time of Flight (ToF) principle as shown in *Figure 4.3*. The basic principle is as follows: knowing the speed of light, the distance to be measured is proportional to the time needed by the active illumination source to travel from emitter to target. Thus, matricial ToF cameras enable the acquisition of a distance-to-object measurement, for each pixel of its output data (9).

The Kinect sensor as shown in table 4.1 has the following properties and functions:

- An RGB Camera that stores three channel data in a 1280x960 pixel resolution at 30Hz. The camera's field of view as specified by Microsoft is 43° vertical by 57° horizontal. The system can measure distances with a 1cm accuracy, at 2 meters distance
- An infrared (IR) emitter and an IR depth sensor used for capturing depth image. The IR sensor is based on Time of Flight (ToF) principle as shown in *Figure 4.3*. Thus, matricial ToF cameras enable the acquisition of a distance-to-object measurement, for each pixel of its output data (9).
- An array of four microphones to capture positioned sounds
- A tilt motor which allows the camera angle to be changed without physical interaction and a three-axis accelerometer which can be used to determine the current orientation of the Kinect

Infrared (IR) camera resolution	512 × 424 pixels
RGB camera resolution	1920 × 1080 pixels
Field of view	70 × 60 degrees
Framerate	30 frames per second
Operative measuring range	from 0.5 to 4.5 m
Object pixel size (GSD)	between 1.4 mm (@ 0.5 m range) and 12 mm (@ 4.5 m range)

Table 4.1 *Kinect specifications*(10)

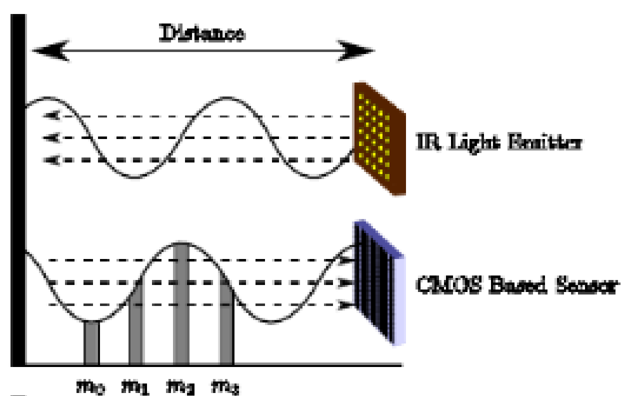

Figure 4.3 *Distance measurement of ToF cameras*(9)

4.1.1. Kinect software development kit for Windows (SDK)

In order to work with Kinect in the computer, it is required to download the Kinect Software Development Kit (SDK) 2.0 developed by Microsoft for the Kinect sensor.

The Windows SDK provides us with several libraries for creating Windows applications that use native code and provides us with script samples so it can help us to begin with our programming. Microsoft states that the SDK 2.0 enables developers to create applications that support gesture and voice recognition. It is possible to create applications using the device, and the positions of 25 human joints can be estimated using de body tracking algorithm (11). Using this algorithm, the body joints are inferred using a machine learning algorithm called randomized decision forest.

Figure 4.4 shows how Kinect communicates with an application. The SDK in conjunction with the Natural User Interface (NUI) library provides the tools and the Application Programming Interface (APIs) needed such as high-level access to colour and calibrated depth images, the tilt motor, advanced audio capabilities, and skeletal tracking (12).

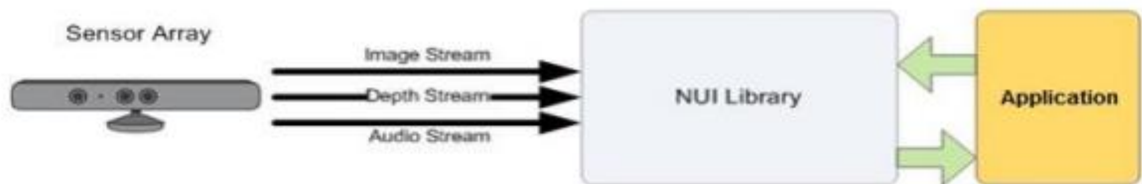

Figure 4.4 *Kinect interaction with an application*(12)

5. Application Programming Interface (API)

The formal definition of API by *TechTerms* states that “An API is a set of commands, functions, protocols, and objects that programmers can use to create software or interact with an external system. It provides developers with standard commands for performing common operations so they do not have to write the code from scratch ". There are many different types of APIs for operating systems, applications or websites. Kinect and Unity have each one a different API and one of our goals is to communicate those API.

Visual Studio (VS) is a programming environment from Microsoft in which a programmer uses a graphical user interface (GUI) to choose and modify preselected sections of code written in the BASIC programming language. The programming language used to program in Unity is C# (C Sharp). It's a language that is derived from C and C ++ and was created by Microsoft as part of the .NET platforms.

VS user works with scripts. In computer programming, a script is a program or sequence of instructions that is interpreted or carried out by another program rather than by the computer processor.

The integrated development environment (IDE) of the project was Visual Studio 2017. This program allowed the creation of the necessary scripts thanks to automatic construction tools or the easy detection of errors. C# is a quite elaborate programming language. In addition, this language has countless predetermined functions that allow performing many different operations easily.

5.1. Kinect

The Kinect libraries allow us to detect the three-dimensional positions of 25 anatomical points distributed on the user's body. Each detected point can refer to a real joint (neck, shoulders, hips, pelvis, elbows, knees, wrists or ankles) or to the centre of a body segment as showed in *Figure 5.2.* A Joint is a structure that includes:

- The position in the 3D space
- The type/name of the joint
- The tracking accuracy

The goal of tracking the human body segments is to get their orientations. The accuracy of how the joint positions is sensitive to the position and orientation of the camera regarding the location of the body (13). For example, self-occlusion of some body parts by other parts could lead to a poor skeleton's model estimation by the camera. So a possible move for the future could be using more than one Kinect to improve this tracking.

The body joints have the following information:

- *Position:* It represents the absolute position of each body point through a 3D vector.
- *Orientation*: the body orientations are captured as absolute rotations respect to the (*Figure 5.3*). The rotations are represented by quaternions (x, y, z, w). A quaternion is an axis in 3D space with an angle of rotation around the axis. Four values make up a quaternion, namely x, y, z and w. Three of the values are used to represent the axis in vector format, and the forth value would be the angle of rotation around the axis. They are, specifically, unit quaternion and they avoid the Gimbal lock problem.
  Gimbal lock (Figure 5.1) is the loss of one degree of freedom on a three-dimensional. That occurs when the axes of two of the three gimbals are driven into a parallel configuration, leading into a degenerate two-dimensional space.

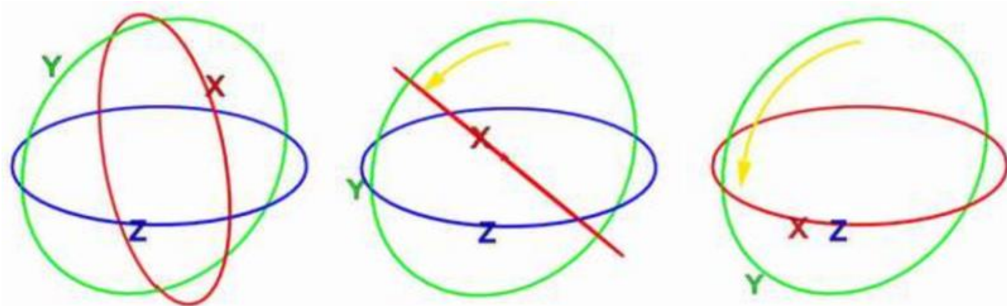

Figure 5.1 *Gimbal lock problem*(14)

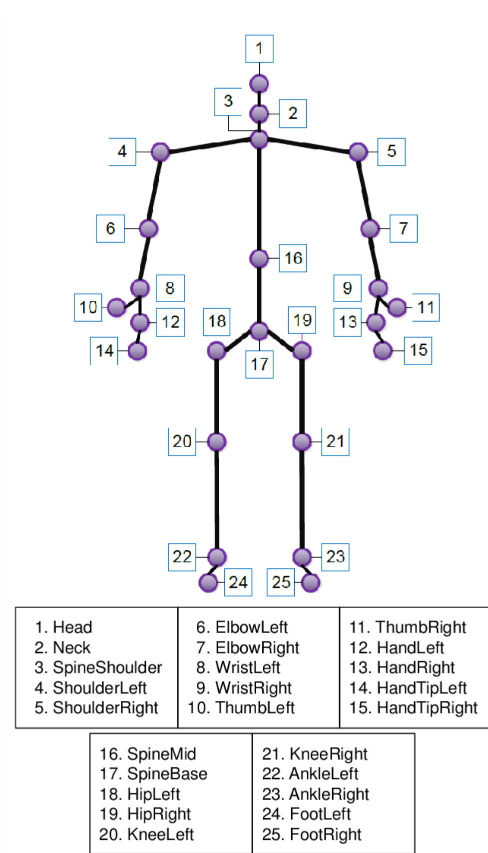

Figure 5.2 *3D skeleton joints tracked*

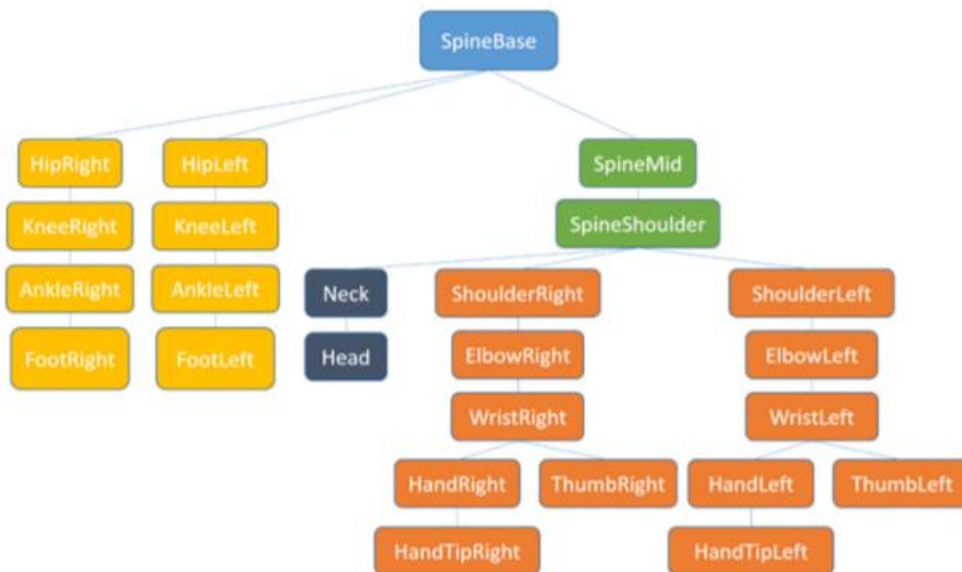

Figure 5.3 *Kinect skeleton joints hierarchy*

5.2. Unity 3D

Unity gives users the ability to create games in both 2D and 3D, and the engine offers a primary scripting API in C#, for both the Unity editor in the form of plug-in, and games themselves, as well as drag and drop functionality.

Making the Unity work is really simple. A project is created and it consists on different scenes, which contain different elements known as "GameObjects".This basic element in Unity is the main class in which the rest of the attributes are added as components, scripts, audio sources, cameras, etc. The way to organize is through scenes which represent the different levels of the application, from the menus to the credits (15).

The objects of a scene are created by loading and destroyed when changing to another, so to preserve any of the objects between scenes must be specified by code. Some of the most important components of the GameObjects are:

- The *renderer*, which ensures that the object is visible, giving it a shape and color or texture
- The *rigidbody* and *collider*, responsible for managing collisions with other elements and, in particular, rigidbodys, physical characteristics
- The camera represents the vision that will have each level
- The component to which scripts are added

Some of the attributes of the components can be edited both by code and from the editor as are the public variables of the added scripts. All components can be enabled or disabled except the transform that comes by default in all gameobjects and decides the position, rotation and scale of each object.

To facilitate the creation and multiplication of objects, prefabs are used. They act as temaplates from which you can create new object instances in the scene and can be made to which they are assigned a previously configured object to then be able to instantiate them by code or from the editing window of Unity3D.

The life cycle that the gameobjects follow is based on the classic life cycle of the graphic engines. They are initialized for the first time and then they enter in a loop until it is destroyed. In the loop the code or animations updates are made and the object is rendered until it is destroyed.

We use the library offered by Unity3D called UnityEngine and the scripts must inherit from the MonoBehaviour class that helps us access to several methods used in our projects.

The Unity display is really intuitive and approachable to any user with little experience on this field of game development. Below is shown how to easily cope with this sofware as the common display structure is defined in figure 5.4:

1. These are three selectable tabs:
    - Scene is used to select and position scenery, characters, cameras and other types of Game Object.
    - Game shows our final game when all is ready.
    - Asset store is a platform where we can find several applications and pruchase some games.
2. The project window give us a view of oy project folders where we have all of our scripts, assets, pluggins, etc... Here we can also drag and drop anything to include it in our project.
3. Those are the play mode buttons, while the play button is on yor game will be running until you press play again.
4. The console give us feedback of anything that occurs while the porgram is running such as warnings or errors.
5. On the Hierarchy tab we have our GameObjects classified so whenever we want to edit one we just need to select it and then access the information displayed on the Inspector tab.
6. The inspector give us detailed information of the GameObject currently selected and we can also attach scripts to our desired GameObject as explained later on.

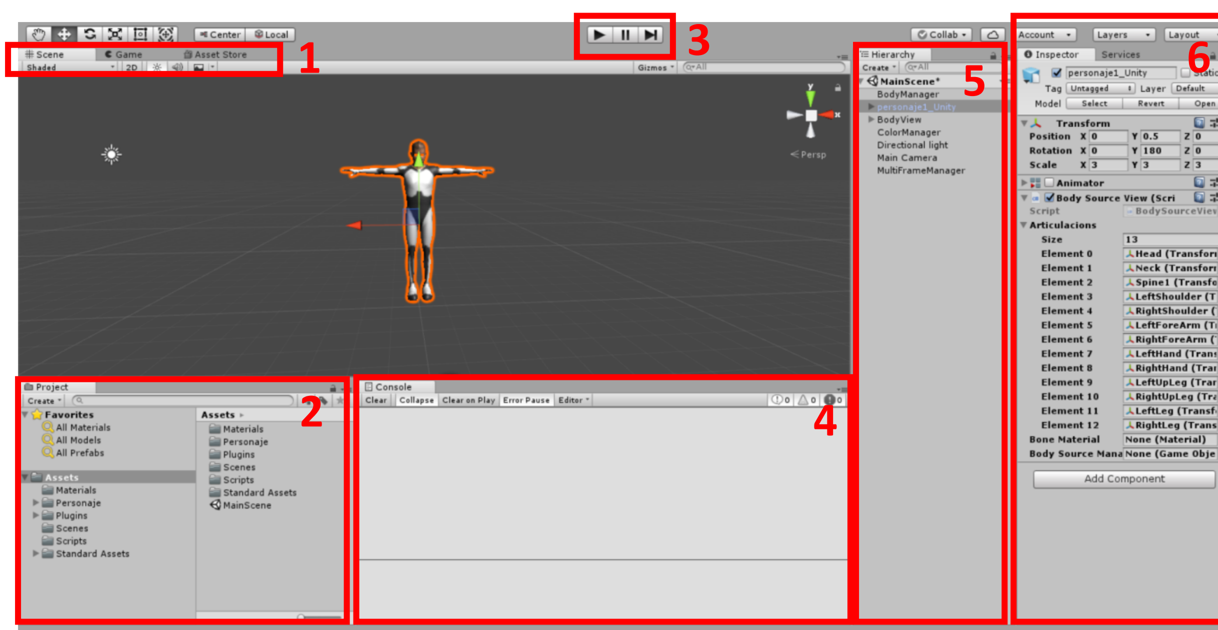

***Figure 5.4** Unity 3D display*

5.2.1. Avatar set up

The first step in Unity is to integrate the avatar into our scene. Scenes contain the environments and menus of your game. Think of each unique Scene file as a unique level. In each Scene, you place your environments, obstacles, and decorations, essentially designing and building your game in pieces (16).

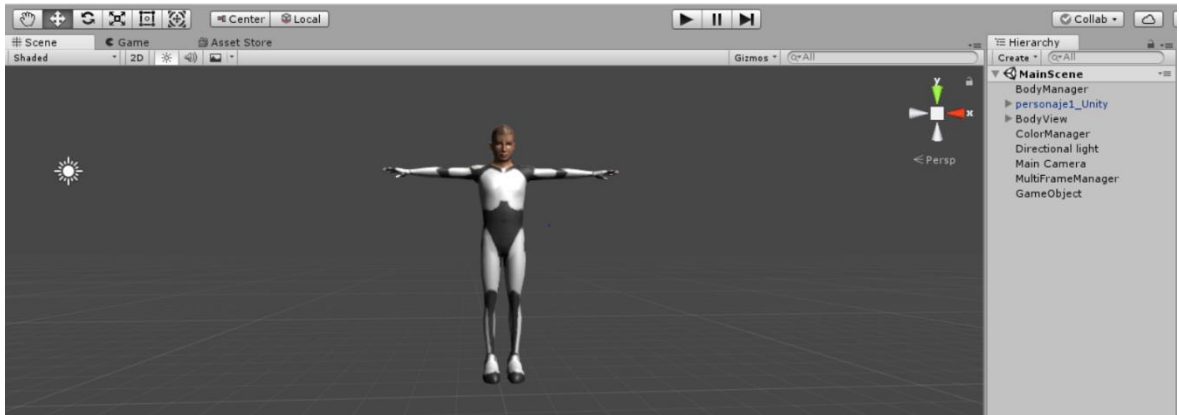

Figure 5.5 *Unity avatar scene display*

As we are getting the avatar (17) from outside Unity, it is needed to import it and any related textures into the assets folder in the project window. After you the model in the assets folder, we must ensure it has the correct settings set, under the rig tab (Figure 5.6) in the inspector the Animation Type should be set to Humanoid.

Setting up a humanoid Avatar in Unity involves matching every “human bone” to one of the transforms in the model. It’s possible to do this manually in Unity by clicking Configure. Selecting the Configure Avatar mode to check that your Avatar is valid and properly set up as shown in figure 5.6. It is important that the character’s bone structure matches Unity’s avatar predefined bone structure and that the model is in T-pose(16) because is the pose predefined in Unity. As the avatar is now humanoid, it allows to map the avatar body. The Avatar Mapping (*figure 5.7*) indicates which of the bones are required (solid circles) and which are optional (dotted circles). Unity can interpolate optional bone movements automatically.

Once the avatar is correcty defined, it has to be dragged into the hierarchy or directly to the scene.

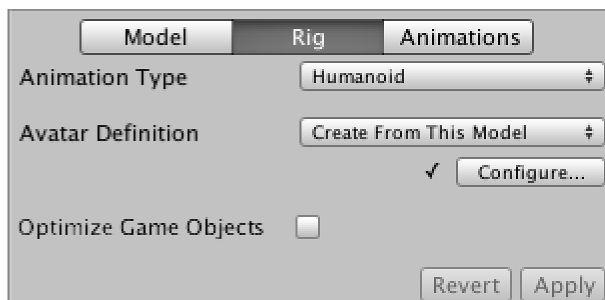

Figure 5.6 *Configure Avatar mode*

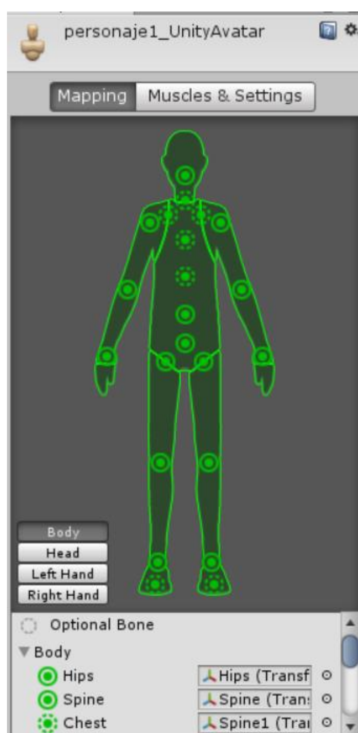

Figure 5.7 *Avatar Mapping*

6. Evaluation

In this section, we describe the experimental part of our Kinect-based rehabilitation system and the design of the avatar movements in Unity 3D.

6.1. Kinect and Unity integration

In 2014, Microsoft released a Unity3D plug-in for the Kinect 2. A package called "Unity Pro packages" is available to install from the Microsoft Kinect SDK for Windows site (18). It is used for Kinect-based applications development through Unity. The package is simply imported, and all required assets will appear in the project.

The package, contains three more packages inside. The first file "Kinect.2.0.xxxx.unitypackage" contains base functionality of Kinect SDK for Unity. That is our plug in, along with all the scripts needed to build a Kinect-enabled Unity application it will allow tracking bodies, leans, colours and so on. But if you want to use functionality, related to face recognition (emotions, face HD tracking etc.), the second package is required, Kinect.Face.2.0.xxxx.unitypackage. Finally, the last package contains API which will help to use data from Visual Gesture Builder in order to simplify a way to understand predefined gestures.

Kinect Sensor command is used in order to have access to Kinect. Kinect Sensor class provides some properties, which allows us to get sources' references. Since it is important to know basic body movements only, as Kinect itself calculate all the body movements by interpolating between those basic body movements, just the "BodyFrameReader" command will be used. Additionally, an array of Body class is needed in order to store current information about the body as shown in Figure 6.1. To "Kinectize" the game, an invisible GameObject called "**Body[] _Data**", which exchange data from Kinect and the game, is inserted.

```
private KinectSensor _Sensor;
private BodyFrameReader _Reader;
private Body[] _Data = null;
```

Figure 6.1 *Kinect commands for Unity projects*

6.2. Kinect

On this chapter, the coding part of the project is described. Two scripts will be treated as those are the scripts in C# used to achieve our objective, apply the orientations into the avatar.

6.2.1. Main script: BodySourceView

First, Kinect body joints dictionary needs to be defined (Figure 6.2) similar to the Kinect Body Basics sample so we can call any joint and work with them. If we did not have this dictionary, our script wouldn't get this information which is crucial.

```
private Dictionary<Kinect.JointType, Kinect.JointType> _BoneMap = new Dictionary<Kinect.JointType, Kinect.JointType>()
{
    { Kinect.JointType.FootLeft, Kinect.JointType.AnkleLeft },
    { Kinect.JointType.AnkleLeft, Kinect.JointType.KneeLeft },
    { Kinect.JointType.KneeLeft, Kinect.JointType.HipLeft },
    { Kinect.JointType.HipLeft, Kinect.JointType.SpineBase },

    { Kinect.JointType.FootRight, Kinect.JointType.AnkleRight },
    { Kinect.JointType.AnkleRight, Kinect.JointType.KneeRight },
    { Kinect.JointType.KneeRight, Kinect.JointType.HipRight },
    { Kinect.JointType.HipRight, Kinect.JointType.SpineBase },

    { Kinect.JointType.HandTipLeft, Kinect.JointType.HandLeft },
    { Kinect.JointType.ThumbLeft, Kinect.JointType.HandLeft },
    { Kinect.JointType.HandLeft, Kinect.JointType.WristLeft },
    { Kinect.JointType.WristLeft, Kinect.JointType.ElbowLeft },
    { Kinect.JointType.ElbowLeft, Kinect.JointType.ShoulderLeft },
    { Kinect.JointType.ShoulderLeft, Kinect.JointType.SpineShoulder },

    { Kinect.JointType.HandTipRight, Kinect.JointType.HandRight },
    { Kinect.JointType.ThumbRight, Kinect.JointType.HandRight },
    { Kinect.JointType.HandRight, Kinect.JointType.WristRight },
    { Kinect.JointType.WristRight, Kinect.JointType.ElbowRight },
    { Kinect.JointType.ElbowRight, Kinect.JointType.ShoulderRight },
    { Kinect.JointType.ShoulderRight, Kinect.JointType.SpineShoulder },

    { Kinect.JointType.SpineBase, Kinect.JointType.SpineMid },
    { Kinect.JointType.SpineMid, Kinect.JointType.SpineShoulder },
    { Kinect.JointType.SpineShoulder, Kinect.JointType.Neck },
    { Kinect.JointType.Neck, Kinect.JointType.Head },
};
```

Figure 6.2 *Body joints dictionary*

All the processes referring to the body tracking will be called after an "**if (body.IsTracked)**" condition (Figure 6.3) so false or contaminated data is not captured. That means that only when the Kinect detects that there is a body on its field of view, the script will begin to store, frame per frame and the orientations.

```
if (body.IsTracked)
{
```

Figure 6.3 *Body tracked condition*

The 25 joints that Kinect tracked are analysed and to their properties can be accessed. Before coding and getting those orientations, a checking if the Unity avatar joints matches the skeleton Kinect joints must be made. They do not match perfectly as the Unity Automap (*Figure 6.4*) gives a wide range of joints and some of them are named and placed different comparing to the Kinect default skeleton. In order to match both joint skeletons, a matching list was made comparing *Figure 5.2 and 5.7* as shown in *Table 6.1.* As seen in the table, there are 22 joints that matches both API, Kinect and Unity. SO, the project will consists on analysing those 22 joints.

Figure 6.4 *Auto mapped Joint GameObjects*

	Kinect skeleton joints	Unity avatar joints GameObjects
1	Head	Head
2	Neck	Neck
3	SpineShoulder	-
4	ShoulderLeft	LeftArm
5	ShoulderRight	RightArm
6	Elbowleft	LeftForeArm
7	ElbowRight	RIghtForeArm
8	WristLeft	LeftHand
9	WristRight	RightHand
10	ThumbLeft	LeftHandThumb1
11	ThumbRight	RightHandThumb1
12	HandLeft	LeftFingerBase
13	HandRight	RightFingerBase
14	HandTipLeft	-
15	HandTipRight	-
16	SpineMid	Spine2
17	SpineBase	Spine
18	HipLeft	LeftUpLeg
19	HipRight	RightUpLeg
20	KneeLeft	LeftLeg
21	KneeRight	RightLeg
22	AnkleLeft	LeftFoot
23	AnkleRight	RightFoot
24	FootLeft	LeftToeBase
25	FootRight	RightToeBase

Table 6.1 *Unity and Kinect joints*

6.2.1.1. Quaternions (Orientations processing)

Now that it is ensured what joints are going to be processed, we can access now the joint properties, in particular, the orientations. As mentioned before, the orientations are defined by quaternions. In Computer Graphics, quaternions are sometimes used in place of matrices to represent rotations in 3-dimensions (19). Quaternions ensure us a smooth and direct interpolation.

By default Kinect have all the 25 joints stored as well as their transform which includes the position, orientation and scale.

It is possible to access the joint information by a series of commands proper to C# as shown in Figure 6.5a-6.5b. The variable orientation[i] (with "i" going from 1 to 22) saves and updates frame per frame the orientation of each joint. The quaternion variables (x, y, z, w) are given by radians but if an orientation wants to be changed manually, directly in Unity, the orientations must be written in degrees as Unity only works with quaternion internally.

```
// *************************** JOINT ORIENTATIONS****************************************//


var orientation1 = Body.JointOrientations[JointType.Head].Orientation;

var orientation2 = Body.JointOrientations[JointType.Neck].Orientation;

var orientation3 = Body.JointOrientations[JointType.SpineMid].Orientation;

var orientation4 = Body.JointOrientations[JointType.ShoulderLeft].Orientation;

var orientation5 = Body.JointOrientations[JointType.ShoulderRight].Orientation;

var orientation6 = Body.JointOrientations[JointType.ElbowLeft].Orientation;

var orientation7 = Body.JointOrientations[JointType.ElbowRight].Orientation;

var orientation8 = Body.JointOrientations[JointType.WristLeft].Orientation;

var orientation9 = Body.JointOrientations[JointType.WristRight].Orientation;

var orientation10 = Body.JointOrientations[JointType.HipLeft].Orientation;

var orientation11 = Body.JointOrientations[JointType.HipRight].Orientation;

var orientation12 = Body.JointOrientations[JointType.KneeLeft].Orientation;

var orientation13 = Body.JointOrientations[JointType.KneeRight].Orientation;
```

Figure 6.5a *Access to joint Kinect orientations*

```
var orientation14 = Body.JointOrientations[JointType.SpineBase].Orientation;

var orientation15 = Body.JointOrientations[JointType.AnkleLeft].Orientation;

var orientation16 = Body.JointOrientations[JointType.AnkleRight].Orientation;

var orientation17 = Body.JointOrientations[JointType.FootLeft].Orientation;

var orientation18 = Body.JointOrientations[JointType.FootRight].Orientation;

var orientation19 = Body.JointOrientations[JointType.HandLeft].Orientation;

var orientation20 = Body.JointOrientations[JointType.HandRight].Orientation;

var orientation21 = Body.JointOrientations[JointType.ThumbLeft].Orientation;

var orientation22 = Body.JointOrientations[JointType.ThumbRight].Orientation;
```

Figure 6.5b *Access to joint Kinect orientations*

6.2.2. BodySourceManager script

This script is a must in every project that involves Kinect and Unity as explained later on.

First of all, all the variables must be declared, if we declare a variable private, it means they are accessible only within the body of the class or the structure in which they are declared. On the other hand, if we declare it public, the variable can be accessed from outside the class where is defined.

The function Start is called once when a script is enabled just before any of the Update methods are called the first time. Start is called exactly once in the lifetime of the script. In the figure 6.6a, what we will call from the Start function will be getting the Kinect sensor ready and activated once the program is started.

Continuing with the script, we now focus on the Update function. As mentioned previously, here we write those functions that we want to be updated every frame. As shown in figure 6.6b, a function is created which will store every new data frame so it doesn't overlap.

Finally, we need a function that closes our sensor once we end the application session. We set the "OnApplicationQuit()" function to assure the full closing of the sensors used because as mentioned before any false data before or after the run of the application is wanted.

```
public class BodySourceManager : MonoBehaviour
{
    private KinectSensor _Sensor;
    private BodyFrameReader _Reader;
    private Body[] _Data = null;

    public Body[] GetData()
    {
        return _Data;
    }


    void Start ()
    {
        _Sensor = KinectSensor.GetDefault();

        if (_Sensor != null)
        {
            _Reader = _Sensor.BodyFrameSource.OpenReader();

            if (!_Sensor.IsOpen)
            {
                _Sensor.Open();
            }
        }
    }
```

Figure 6.6a *BodySourceManager script*

```
void Update ()
{
    if (_Reader != null)
    {
        var frame = _Reader.AcquireLatestFrame();
        if (frame != null)
        {
            if (_Data == null)
            {
                _Data = new Body[_Sensor.BodyFrameSource.BodyCount];
            }

            frame.GetAndRefreshBodyData(_Data);

            frame.Dispose();
            frame = null;
        }
    }
}

void OnApplicationQuit()
{
    if (_Reader != null)
    {
        _Reader.Dispose();
        _Reader = null;
    }

    if (_Sensor != null)
    {
        if (_Sensor.IsOpen)
        {
            _Sensor.Close();
        }

        _Sensor = null;
    }
}
```

Figure 6.6b *BodySourceManager script*

This script must be dragged into our main script, in the inspector window in order to be initialized at the same time as the main script and the avatar.

6.3. Avatar movements

Once the avatar is fully functional and the joints are mapped we can move on by attaching the *BodySourceView* script to the humanoid model. We need to drag this script from the project window to our avatar in the inspector window. We are able to apply every command we want to the avatar although it will not respond because we did not relate the orientations stored in the script with the avatar joint GameObjects. In Figure 5.4 (sector 6) we can see that we put the script into our avatar and it is marked as "okay".

Now we have to return to our main script (BodySourceView) where we calculated all the joints rotations because we need to call those joints from the avatar. In Figure 6.7, the function called, "*articulacions*" consists on an array of joints transforms. As stated before, it is used to store and manipulate the position, rotation and scale of the object. In our project we only focus on the rotations transform. This array at first is empty and requires inputs.

```
public Transform[] articulacions;
```

Figure 6.7 *Joint transforms function*

Returning to the hierarchy window in Unity, the script, where we placed the function mentioned before, is selected. In the inspector, appears a box with the function name "*articulacions*" and the size of it is 0. That means the array has no information. It is needed to write the number of joints that are wanted the transform from. Now, there are joint rotations values of 22 joints.

Now manually every joint GameObject from the avatar hierarchy must be dragged and dropped, matching the order defined previously on the *BodySourceView* script, on the blank spaces in "*articulacions*" as shown in Figure 6.7. C# arrays are zero indexed, which means that the array indexes start at zero. The default number of array elements is set to zero and the reference element is set to null.

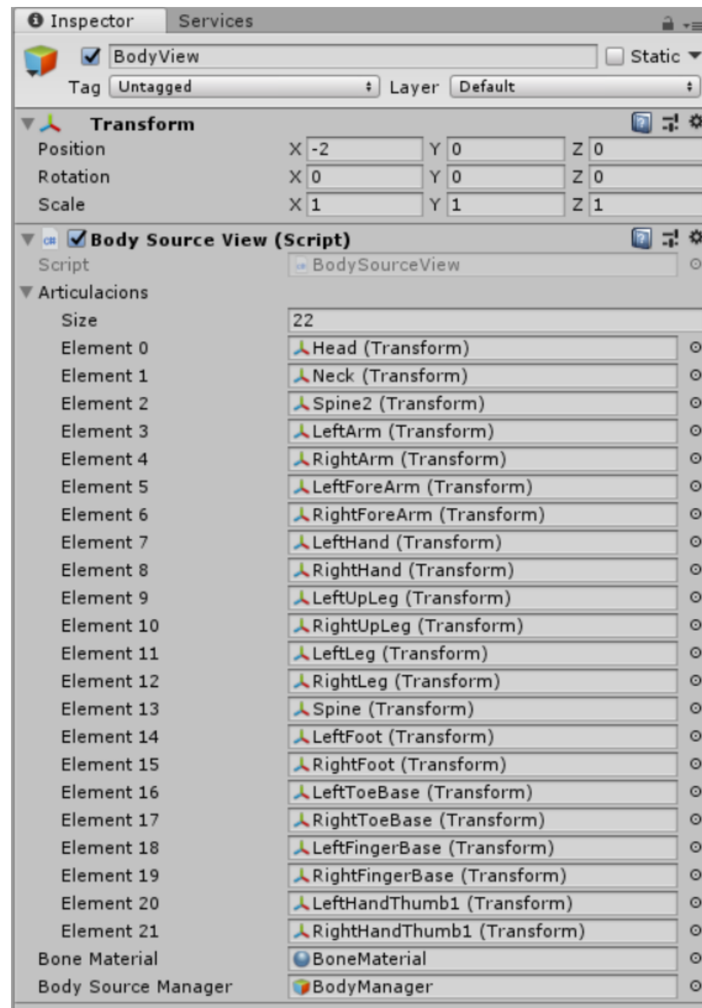

Figure 6.8 *Avatar joint GameObjects*

Now it is possible to call from our script the joint GameObjects classified in Figure 6.8. The process would be the same as the one followed from entering to the Kinect orientations. But in this case, another variable is created to save, frame per frame, the Kinect orientations.

As Kinect and Unity have different quaternion definitions on their orientations, the new variables are defined as “***UnityEngine.Quaternion*** **variable_name**" so later can be transferred to the Unity joints. Due to the fact that those quaternions are defined differently, we cannot equal the new variables with the Kinect orientations. So the process will be equalling value per value accessing to the quaternions variables as shown in Figure 6.9.

Now the new variables are prepared to take the Kinect orientations values. So the next step is applying those Unity orientations into our joint avatar GameObjects. This is the most important step since it is the main objective. Note that all rotations occur in absolute space, both Kinect and Unity orientation.

As it is known, the function "*articulacions*[]" save all the avatar joints that we defined on the inspector window in Unity. As shown in Figure 6.9, we access every joint orientation and apply the new orientations frame per frame, as commented many times. "*articulacions[].rotation*" allows to work with the rotation of the transform in world space as a Quaternion.

```
// NECK
or1_unity.x = orientation1.X;
or1_unity.y = orientation1.Y;
or1_unity.z = orientation1.Z;
or1_unity.w = orientation1.W;
articulacions[0].rotation = or1_unity;
```

Figure 6.9 *Quaternion application code on avatar Neck example*

At this point, all the quaternion were applied to each avatar joint. However, the avatar was not following our movements correctly because in order to use those rotations it’s necessary to know what position a given bone is being rotated from. For example, if a person’s arm is to be rotated “up” by ninety degrees, the final direction the arm is pointing, is different if the arm started off pointing forward as opposed to pointing straight down. What it means is that every bone has its own coordinate system so if it does not match the Kinect coordinate system we will have to apply rotations.

6.3.1. Coordinate systems adaptation

In this section we state how to work with different coordinate systems between two API, Kinect and Unity. In order to use the Kinect data, it is necessary to “remap” the Kinect rotations so that they treat

bones as being aligned along the desired axis. Then, it is important that each bone is oriented so that its three vectors are pointing in the same direction as the Kinect data.

The "SpineMid" joint example is showed in Figure 6.10-Figure 6.11. The "SpineMid" has its own coordinate system and it is not the same as the coordinate system from Kinect. At this point, a rotation is needed so those coordinate systems match.

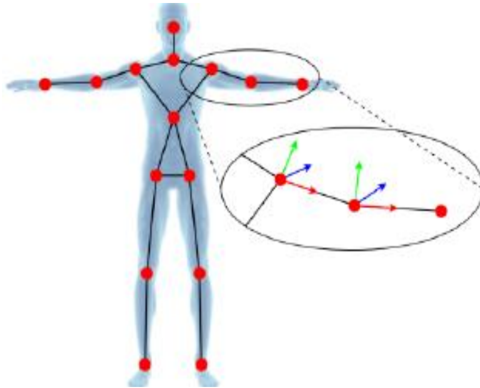

Figure 6.10 *Kinect skeleton local coordinate system*

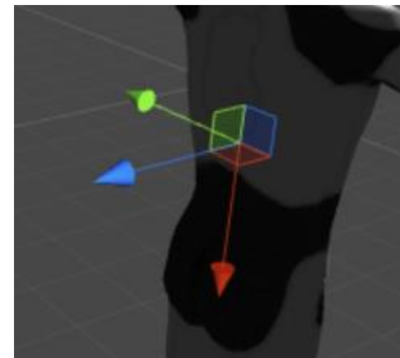

Figure 6.11 *SpineMid Unity coordinate system*

Note the avatar is being watched from behind. First, the X axis will be rotate to be aligned as the Kinect X axis (pointing right). An auxiliar quaternion needs to be created to rotate about the Z axis -90 degrees. Now the X axis is orientated to the desired direction (right) but the Y axis is pointing down and the Z axis pointing at us. In order to correct the Y and Z axis, a rotation of 180 degrees is applied to the Y axis so the Kinect coordinate system is achieved as shown in Figure 6.12.

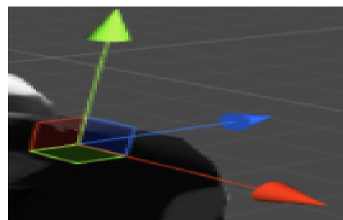

Figure 6.12 *Coordinate system rotated*

As mentioned before, the code to rotate those quaternion is created. The rotation quaternion are created as auxiliar quaternion to rotate the bodies. We create as many auxiliar quaternion as axis

rotations to be made. Then in order to apply those quaternion we multiply the new orientations from the kinect to the auxiliar quaternion. As a result, it is obtained a new quaternion ("or3_new") with the rotations applied.

Finally, the "or3_new" quaternion is applied and the avatar can now follow our movements correctly as desired.

```
//////SPineMid
or3_unity.x = orientation3.X;
or3_unity.y = orientation3.Y;
or3_unity.z = orientation3.Z;
or3_unity.w = orientation3.W;

Quaternion or3_unity_aux;
Quaternion or3_unity_aux2;
Quaternion or3_new;

or3_unity_aux = Quaternion.AngleAxis(90, Vector3.back);
or3_unity_aux2 = Quaternion.AngleAxis(180, Vector3.up);
or3_new = or3_unity_aux * or3_unity_aux2 * or3_unity;
articulacions[2].rotation = or3_new;
```

Figure 6.13 *Coordinate systems adaptation code*

This process needs to be made through all the joints. However, there are groups of joints that use the same rotations. For example, all the spine joints have the same coordinate system so the same auxiliary quaternion are applied to them in order to achieve our objective.

7. Results

In this section, the data obtained, through our code will be showed as a txt file. Before that, we have all the avatar rotations in an array where all the 22 joints are placed as shown in Figure 7.1. The array has 22 rows, like the amount of joints studied, and 4 columns, like the 4 quaternion variables (x, y, z, and w). Once the array is defined, a for loop command (Figure 7.2) can be created in order to evaluate each array value. The for loop executes a block of statements repeatedly until the specified condition returns false. Note that the quaternion are rounded to 4 decimals so we can analyse the data easily.

```
//*************************************Orientations array**********************************//

orientationsarray = new double[22, 4]{{ or1_new.x,or1_new.y,or1_new.z,or1_new.w},
                                { or2_new.x,or2_new.y,or2_new.z,or2_new.w },
                                { or3_new.x,or3_new.y,or3_new.z,or3_new.w },
                                { or4_new.x,or4_new.y,or4_new.z,or4_new.w },
                                { or5_new.x,or5_new.y,or5_new.z,or5_new.w },
                                { or6_new.x,or6_new.y,or6_new.z,or6_new.w },
                                { or7_new.x,or7_new.y,or7_new.z,or7_new.w },
                                { or8_new.x,or8_new.y,or8_new.z,or8_new.w },
                                { or9_new.x,or9_new.y,or9_new.z,or9_new.w },
                                { or10_new.x,or10_new.y,or10_new.z,or10_new.w },
                                { or11_new.x,or11_new.y,or11_new.z,or11_new.w },
                                { or12_new.x,or12_new.y,or12_new.z,or12_new.w },
                                { or13_new.x,or13_new.y,or13_new.z,or13_new.w },
                                { or14_new.x,or14_new.y,or14_new.z,or14_new.w },
                                { or15_new.x,or15_new.y,or15_new.z,or15_new.w },
                                { or16_new.x,or16_new.y,or16_new.z,or16_new.w },
                                { or17_new.x,or17_new.y,or17_new.z,or17_new.w },
                                { or18_new.x,or18_new.y,or18_new.z,or18_new.w },
                                { or19_new.x,or19_new.y,or19_new.z,or19_new.w },
                                { or20_new.x,or20_new.y,or20_new.z,or20_new.w },
                                { or21_new.x,or21_new.y,or21_new.z,or21_new.w },
                                { or22_new.x,or22_new.y,or22_new.z,or22_new.w }
};
```

Figure 7.1 *Orientations array*

```
            };
            //************************************* TXT file creation **********************************//

            for (int i = 0; i < 22; i++)
            {
                for (int j = 0; j < 4; j++)
                {
                    double datos1 = Math.Round(orientationsarray[i, j], 4);

                    string datos = datos1.ToString();
                    File.AppendAllText(@"C:\Users\USUARIO\Documents\TFG\virtualMirror_3108\orientationsarray.txt", datos);

                }
                File.AppendAllText(@"C:\Users\USUARIO\Documents\TFG\virtualMirror_3108\orientationsarray.txt", Environment.NewLine);
            }
            File.AppendAllText(@"C:\Users\USUARIO\Documents\TFG\virtualMirror_3108\orientationsarray.txt", Environment.NewLine);


        }


    }

}
```

Figure 7.2 *Txt for loop*

Now the quaternion applied to the avatar are printed on a txt file (Figure 7.3) to make sure, the orientations of the avatar are correct or at least, no false data is received. In Figure 7.3 show the quaternion obtained in a upright body posture with no movements.

```
Head        0000  x      y      z      w
            0.0303-0.04410.7006-0.7115
            0.0613-0.06140.6993-0.7095
            0.5560.8234-0.11340.0075
            0.0076-0.1251-0.82690.5483
            -0.73470.0980.66570.0864
            0.6462-0.2602-0.677-0.2374
            0.0194-0.3923-0.7599-0.5179
            -0.47560.1533-0.4489-0.7408
            -0.67280.6948-0.11460.2267
            0.6266-0.1531-0.1626-0.7466
            -0.7028-0.03980.69890.1268
            -0.0867-0.06510.6671-0.737
            0.0676-0.0550.6987-0.71
            -0.69890.0230.7140.0348
            -0.005-0.00930.6809-0.7323
FootLeft    0000
FootRight   0000
            0.12040.8725-0.3504-0.3185
            0.4602-0.318-0.80930.1794
ThumbLeft   0000
ThumbRight  0000
```

***Figure 7.3** Orientations array txt (One frame)*

The Figure 7.3 show that even though the body posture is still, there are initial rotations which come from the initial coordinate systems adaptation. This txt file show the same joint order followed during the project. So, it is fast to analyse the desired joint. There are some values from which there is no information of its orientation or simply they are 0.

- *Head*: This joint give us 0 values because the parent bone of the joint Head is itself. Unity defines rotations with quaternion of solids relative to the ground, except solids that do not have a parent joint, which assigns a zero value.
- *FootLeft/FootRight*: Those joints give 0 value because leaf joints have no orientation data.
- *ThumbLeft/ThumbRight*: Those joints have the same problem as the feet. The give 0 value because leaf joints have no orientation data.

For leaf joints the orientation quaternions returned have all components set to 0.

8. Environment impact analysis

This project does not have an environmental impact as itself, since during its realization no residue is created. The only cost to consider is the energy used by the electronic devices during the execution of the project but the cost associated to this is minimum compared with the electricity consumption of the facilities of the Barcelona East School of Engineering (EEBE).

What is more, telerehabilitation means environmentally friendly. Emissions are reduced because we can minimize the number of trips to a physical therapy centre. Each telerehabilitation unit is a small emissions saving, and this savings can be significant if these services extend widely to the society.

Finally, we must consider the deterioration of the electronic equipment used: the Microsoft Kinect sensor and the computer. Once they have reached the end of their useful life, they must be withdrawn as indicated Directive 2012/19/EU of the European Parliament and of the Council on the waste of electrical equipment and Electronic (waste electrical and electronic equipment - WEEE) (20), which sets the objectives of its collection, recycling and recovery.

9. Improvement proposals and future applications

In the contemporary world, computer is a source of fun for most people. They spend up to a couple of hours per day in front of the screen, which points to the fact that interactive games keep the people's attention. Besides the motivational aspect, the advantage of computer technology is that the practical element resembles real life situations and as such allows the user to make mistakes and learn in a safe environment.

VR offers a unique medium in which rehabilitation treatments can be offered within a functional, purposeful and motivating context, which can be readily graded and documented.

As it was first said, this project is the first step into developing a full rehabilitation VR application integrating Kinect v2 and Unity. So, the next step is creating a full environment on Unity to improve the patient experience. This environment could simulate daily life situations such as supermarkets, parks, etc.

Another possible functionality to implement in the future would be the management of different users and the register of their progression. When the user logs in their account, the percentage of the improvement achieved in each movement will be shown. This feature will allow us to easily manage and modify the recorded movements. In that case, it would be useful to tell a patient with, for example, an arm injury, if the movement they are doing in the rehabilitation is being correct or not.

In the midterm, those rehabilitations applications will be thrown out to the next level. The next progression of VR will be AR (Augmented Reality). In the AR world, you will be able to mix the real world with the virtual. This takes all of the advantages of VR and puts them at your fingertips and confuses your brain further. AR supplements the real world with virtual (computer-generated) objects that appear to coexist in the same space as the real world. AR was recognised by MIT as one of ten emerging technologies of 2007 (21), and with today's smart phones and AR browsers we are starting to embrace this very new and exciting kind of human-computer interaction(22).

Conclusions

In this project, the first step on creating a telerehabilitation system based on the new VR technologies was presented with the aim of generating a future virtual mirror (VM). Certain studies have shown that the union of VR and rehabilitation generate satisfactory results in both the evolution of patients (23) and their satisfaction (24). Currently, most VR applications in rehabilitation simulate real life activities such as grasping and manipulating objects or performing everyday tasks. These VR systems help patients enhance improve functional ability and realise greater participation in community life (25).

In terms of the realization of the project, at first, the Microsoft SDK for Windows had to be download in order to enable the computer to work with the Kinect v2. Then we had to code in order to get the Kinect orientations of the skeleton joints in quaternion which give us better results than working with Euler angles. In certain analytical procedures and in some applications it is found that the quaternion can offer fundamental computational, operational and/or implementation, and data handling advantages over the conventional rotation matrix (26). What is more, Euler angles have the disadvantages of ambiguity and Gimbal Lock.

During this project I saw some limitations in terms of data acquisition as there were some occlusion on some body parts depending on the subject position, so the data of those parts was not accurate. As mentioned before, one of my future proposal is the use of two Kinect sensor from two different viewpoints. The one-Kinect system is more prone to give poor estimation when occlusion occurs, while the two-Kinect sensor system often gives more depth measurement from the other viewpoint, so that the pose tracking module infers lees cloud points of the occlusion (27). In future improvements of this project, two Kinect sensors could be used in order to erase this problem and acquire more precision.

We also saw that there were some joint orientation data for leaf joints that had their quaternions values set to 0. In these cases, if the 3D model skeleton has been defined to have vertices attached to these joints, then the orientations of these joints' parents should be used. For example, when orienting the right hand tip, then you would use the right wrist orientation instead as the hand tip will be defined as all zeros.

Then, we had to integrate Kinect and Unity by downloading a package from the Microsoft SDK site. In this package, there are some commands that had to be imported to our project in order to work with Kinect and Unity, together, from our C# scripts in Visual Studio. At this point our Unity game was "Kinectized".

Finally, the Kinect orientations had to be applied into the avatar. This part opened a wide range of possible solutions but it was not formally documented. As previously stated, those orientations could be given by Euler angles, with a previous mathematical process from the quaternion, or directly, by applying those quaternion. Some research has been made on this topic. And the conclusions are decisive, computer-based applications work more efficient and deliver better results with quaternion than with Euler angles (26).

Once the body joints orientation quaternion were processed and applied to every avatar joint GameObject, some problems as the Unity skeleton bones were not orientated as the Kinect orientations. So an adaptation had to be made joint per joint in order to match these orientations. Simply, we created auxiliary quaternion to rotate the current orientations, and multiply the current to the auxiliary so a new well oriented quaternion is obtained. Being this problem solved, we transferred the Kinect-captured user motion to the humanoid avatar model and it worked smoothly and correctly.

The data presented in this project show that the main objective of achieving an effective model that integrates Kinect and Unity in real time and animates an avatar has been met.

Economic analysis

The economic study of this project can be divided into two parts: on the one hand, acquisition costs of the devices used and software, and on the other hand, the costs associated with the design process and generation of the code and the time invested with experimentation.

- **Acquisition costs:**

In our case, the Barcelona East School of engineering (EEBE) already had from the beginning of the project, the equipment for its development. Even though, the economic cost that would involve buying them for carrying out similar projects is contemplated. On the one hand, it is necessary to assume the purchase costs of the system of MOCAP Microsoft Kinect V2 along with its adapter for Windows. On the other hand, in regard to the software used, all of them are free and, therefore, their licenses do not imply a cost additional to the project. Unity Personal is the free version of Unity, which is available to use if your income or funds (raised or self-financed) do not exceed $100.000 per year. The result of the acquisition costs is shown in Table 9.

Product	Description	Unit cost (€)	Units	Price (€)
Laboratory equipment	Kinect v2 sensor	85,00	1	85,00
	Windows Adapter	40,00	1	40,00
Software	Unity Personal	0,00	1	0,00
	Microsoft Kinect for windows SDK	0,00	1	0,00
	Visual Studio	0,00	1	0,00
TOTAL				125,00

Table 9 *Acquisition project costs*

- **Realization costs:**

Those costs include the work done by the engineer or researcher that gives them the time dedicated to the implementation and generation of the code and the experimental processes. It is considered a junior engineer. It is taken into account the hours of research and the writing and elaboration of the memory.

It is also taken into account the depreciation associated to the student computer during the duration of the project. It is supposed a product life of 5 years with an annual depreciation of 16% and a residual coefficient value of 20%. The initial cost of the device is € 1.100 and the duration of the project is 4 months (0.3333 years). The costs associated with the energy are minimum. Approximately an average consumption of 0,04kWh due to the computer and the Kinect 0,03kWh. The result is shown in Table 10.

Activity	Description	Unit cost (€)	Units	Price (€)
Employees				
Engineer	Research (h)	30,00	120	3600,00
	Code development (h)	35,00	100	3500,00
	Experimental processes (h)	35,00	40	1400,00
	Thesis writing (h)	30,00	80	2400,00
Equipment				
Energy consumption	PC consumption (kWh)	0,04	340	13,60
	Kinect consumption (kWh)	0,03	100	3,00
PC	Anual depreciation PC	176,00	0,3333	58,66
TOTAL				**10975,26**

Table 10 *Project realization costs*

- **Total costs:**

The total cost of this project including the acquisition costs of the equipment and software used, and the realization costs are showed in Table 11.

Costs	Price (€)
Acquisition costs	125,00
Realization costs	10975,26
TOTAL	11100,26

Table 11 *Total project costs*

Bibliography

1. Norman, J. «Pygmalion's Spectacles,» Probably the First Comprehensive and Specific Fictional Model for Virtual Reality. A: [en línia]. 2006. Disponible a: http://www.historyofinformation.com/expanded.php?id=4543.

2. Lange, B. et al. FAAST: The Flexible Action and Articulated Skeleton Toolkit. A: . 2011,

3. Moeslund,T.B. Hilton,A.,Krüger, V. A survey od advances in vision-based human motion capture and analysis. A: . 2006,

4. Susín, A. i Lligadas, X. *Biomechanical validation of Upper-Body and Lower-Body joint movements of kinect motion capture data for rehabilitation treatments*. 2012.

5. Chan, Y.-J., Chen, S.-F. i Huang, J.-D. A Kinect-based system for physical rehabilitation: A pilot study for young adults with motor disabilities. A: *Research in Developmental Disabilities*. 2011, p. 2566-2570.

6. Chang, Y.-J., Han, W.-Y. i Tsai, Y.-C. A Kinect-based upper limb rehabilitation system to assist people with cerebral palsy. A: *Research in Developmental Disabilities*. 2011, p. 3654-3659.

7. Bolas, M. et al. Development and evaluation of low cost game-based balance rehabilitation tool using the microsoft kinect sensor. A: . 2011,

8. Alexiadis, D., Kelly, P. i Daras, P. Evaluating a dancer's performance using kinect-based skeleton tracking. A: . 2011,p. 659-662.

9. Sergi Foix, G.A. and C.T. Lock-in Time-of-Flight (ToF) Cameras: A Survey. A: *IEEE SENSORS JOURNAL, VOL. 11, NO. 3*. 2011,

10. Remote Sens. Assessment and Calibration of a RGB-D Camera (Kinect v2 Sensor) Towards a Potential use for Close-Range 3D Modeling. A: . 2015,

11. Shotton,J. ,Fitzgibbon, A., Cook,M.,Sharp, T.,Finocchio,M.,Moore,R.,Kipman,A.,Blake, A. Real time human pose recognition in parts from single depth images. A: . 2011,

12. Theodore, M. *The Kinect Up Close: Modifications for Short-Range Depth Imaging*. 2012.

13. Plantard,P.Auvinet, E., Pierres, A.S.L. *Pose Estimation with a Kinect for Ergonomic Studies: Evaluation of the Accuracy Using a Virtual Mannequin*. 2015.

14. Van Oostendorp, H., Jan Beun, Ro. i Van Diggelen, J. *Human-Media Interaction*. 2010.

15. Preciado, J.J. *Diseño y programación con Unity3D de un avatar interactivo con sincronización musical*. 2014.

16. Unity documentation. A: [en línia]. 2017. Disponible a: https://docs.unity3d.com/Manual.

17. Torner, J. et al. *VR_multipose_v1*. 2018.

18. Kinect for Windows. A: [en línia]. Disponible a: https://developer.microsoft.com/en-us/windows/kinect.

19. Goldman, R. Understanding quaternions. A: *Graphical Models*. 2011, p. 21-49.

20. European Parliament. *DIRECTIVE 2012/19 / EU OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 4 July of 2012 on waste electrical and electronic equipment (WEEE)*. 2012. 2012.

21. Jonietz, E. No Title. A: [en línia]. 2007. Disponible a: http://www.techreview.com/special/emerging/.

22. Poelman, R. *A Survey of Augmented Reality Technologies, Applications and Limitations*. Delft University of Technology Jaffalaan, 2010.

23. Piron, L. et al. Virtual Environment Training Therapy for Arm Motor Rehabilitation. A: *Presence Teleoperators Virtual Environ*. 2005, p. 732-740.

24. Lewis, G.N. et al. Virtual reality games for rehabilitation of people with stroke: perspectives from the users. A: *Disability Rehabilitation Assistance Technology*. 2011, p. 453-463.

25. McGoldrick, M.M.M.C.M.C.O. Adaptive Virtual Reality Games for Rehabilitation of Motor Disorders. A: *Universal Access in Human-Computer Interaction. Ambient Interaction*. 2007, p. 681-690.

26. B. Kuipers, J. *Quaternions and rotation sequences*. Calvin College Grand Rapids, 2000.

27. Du, S. et al. *Leveraging Two Kinect Sensors for Accurate Full-Body Motion Capture*. 2015.

Annex A

A1. Kinect specifications

MANUAL DEL PRODUCTO XBOX ONE Y EL SENSOR KINECT

Lea este manual de instrucciones antes del uso y funcionamiento de su consola Xbox One, sensor Kinect para Xbox One y productos accesorios.

INFORMACIÓN IMPORTANTE DE SEGURIDAD Y GARANTÍA DEL PRODUCTO

Este símbolo identifica los mensajes de seguridad y salud en este y en otros manuales de producto
Lea esta guía para obtener información importante de seguridad y salud correspondiente al producto que compró.

ADVERTENCIA: Si este producto no se configura, utiliza y cuida adecuadamente, aumenta el riesgo de lesión grave, muerte, daño a la propiedad, o bien de daño al equipo o sus accesorios. Lee este manual y conserve todas las guías impresas para referencia futura. Para obtener guías de reemplazo, consulte xbox.com/xboxone/support/manual.

CONTRATO DE GARANTÍA LIMITADA DE XBOX ONE Y TÉRMINOS DE LICENCIA DE SOFTWARE
Usted debe aceptar los Términos de uso de Xbox (incluidos los términos del software de Xbox y los términos de licencia de juegos) en xbox.com/live/termsofuse, los términos de licencia de software en xbox.com/xboxone/slt, y la Garantía limitada en xbox.com/xboxone/warranty para usar su consola Xbox One, los accesorios de Xbox y/o el sensor Kinect para Xbox One. Al usar la consola Xbox One, los accesorios de Xbox y/o el sensor Kinect para Xbox One, usted acepta estar sujeto a estos términos. Léalos. Si no acepta estos términos y condiciones, no instale ni utilice la consola Xbox One, los accesorios de Xbox y/o el sensor Kinect para Xbox One. Devuelva el producto Xbox a Microsoft o a su vendedor para que se lo reembolsen.

CONFIGURACIÓN INICIAL DE LA CONSOLA Y ACTUALIZACIONES
Debe conectarse a Internet para realizar la configuración inicial de la consola antes de que pueda jugar sin conexión (se recomienda un ancho de banda de Internet de 1.5 mbps de descarga/768 kbps de subida. Se requiere una cuenta de Microsoft y una cuenta en Xbox Live, en un país/región de Xbox Live compatible con Xbox One para realizar la configuración inicial para algunas características; aplican los costos de su Proveedor de Servicio de Internet). Para obtener respuestas a las preguntas sobre configuración de la consola, instrucciones para la solución de problemas e información de contacto del Servicio de soporte al cliente de Xbox, visite www.xbox.com/support.

USO CON DISPOSITIVOS DE INFRARROJO
El sensor Kinect puede interferir con o disminuir el funcionamiento de dispositivos de infrarrojo, incluidos controles remotos y anteojos en 3D. Si observa interferencias o un rango reducido, intente reubicar el sensor Kinect o el dispositivo de infrarrojo.

SEGURIDAD ELÉCTRICA

ADVERTENCIA
Al igual que con muchos otros dispositivos eléctricos, si no se toman las siguientes precauciones se pueden producir lesiones graves o la muerte por causa de una descarga eléctrica o incendio, o bien daños a la consola Xbox One o el sensor Kinect.

Equipos con energía de CA
Selecciona una fuente de alimentación adecuada para la consola Xbox One:
- Use únicamente la unidad de fuente de alimentación y el cable de CA que vienen con su consola o que recibió de un centro de reparación autorizado. Si no está seguro de tener la unidad de alimentación correcta, compara el número de modelo de la unidad con el número de modelo especificado en la consola. Si necesita una unidad de fuente de alimentación o un cable de CA de reemplazo, puede encontrar información de contacto del Servicio de soporte al cliente de Xbox en xbox.com/xbone/support.
- Confirme que su tomacorriente eléctrico proporciona el tipo de alimentación indicado en la unidad de fuente de alimentación (en términos de voltaje [V] y frecuencia [Hz]). Si no está seguro del tipo de alimentación suministrado en su hogar, consulte con un eléctrico calificado.
- No use fuentes de alimentación no estándar, como generadores o inversores, incluso aunque el voltaje y la frecuencia parezcan aceptables. Use sólo alimentación de CA desde un tomacorriente de pared estándar.
- No sobrecargue el tomacorriente de pared, el cable del alargador, el enchufe múltiple u otros receptáculos eléctricos. Confirme que cuentan con calificación para manejar la corriente total (en amperes [A]) derivada de la consola Xbox One (indicada en la unidad de fuente de alimentación) y cualquier otro dispositivo que esté en el mismo circuito.

PRECAUCIÓN: Cables
- Para reducir los peligros potenciales de tropezar o enredarse, ordene los cables de manera que las personas y las mascotas no se tropiecen o los tiren accidentalmente al moverse o caminar por el área. Tampoco permita que los niños jueguen con los cables.

Para evitar causar daños en los cables y la fuente de alimentación:
- Proteja los cables de alimentación de ser pisados o aplastados.
- Proteja los cables de ser apretados o curvados de manera excesiva, sobre todo por la parte que se conecta al enchufe eléctrico, la unidad de la fuente de alimentación y la consola.
- No sacuda, anude ni doble excesivamente los cables de alimentación ni los utilice de ninguna otra forma inadecuada.
- No exponga los cables de alimentación a fuentes de calor.
- Mantenga a los niños y las mascotas alejados de los cables de alimentación. No permita que los muerdan o mastiquen.
- Al desconectar los cables de alimentación, tire del enchufe, no del cable.

Si un cable de alimentación o unidad de fuente de alimentación se daña de alguna manera, detenga su uso de inmediato. Visite xbox.com/xboxone/support para obtener la información de contacto del Servicio de soporte al cliente de Xbox.

Desconecte la consola Xbox One durante las tormentas eléctricas o cuando no vaya a ser utilizada durante largos períodos.

DISPOSITIVOS CON ALIMENTACIÓN POR BATERÍAS

ADVERTENCIA: Seguridad de las baterías
Se aplican las siguientes precauciones a todos los productos que utilicen baterías desechables o recargables, incluidas las de polímero de litio. El uso inadecuado de baterías podría tener como consecuencia lesiones, la muerte, daño a la propiedad, o bien daño al equipo o sus accesorios como resultado de fugas de

1

líquido, incendio, sobrecalentamiento o explosiones. El líquido derramado es corrosivo y podría ser tóxico. Puede provocar quemaduras en la piel y en los ojos, y resulta nocivo si se ingiere. Para reducir el riesgo de lesiones:

Mantenga las baterías fuera del alcance de los niños.
Extraiga las baterías si están gastadas o antes de guardar el dispositivo durante largos periodos. Siempre extraiga las baterías antiguas, medio descargadas o gastadas inmediatamente y recíclelas o deséchelas de acuerdo con las disposiciones legales locales, regionales y nacionales.
Si sale líquido de una batería, extráigalas todas siguiendo de forma inversa los pasos de instalación de este producto y procure que el líquido derramado no entre en contacto con la piel ni con la ropa. Si el líquido de la batería entra en contacto con la piel o con la ropa, limpie la piel con agua inmediatamente.
Antes de insertar baterías nuevas, limpie exhaustivamente el compartimento de la batería con un paño seco, o bien siga las recomendaciones de limpieza del fabricante de la batería.

- No aplaste, abra, perfore, deforme, caliente por sobre 35 °C (95 °F), aplique calor directo ni arroje las baterías al fuego.
- No mezcle baterías nuevas con viejas o baterías de distintos tipos (por ejemplo, de carbono-cinc y alcalinas).
- No permita que objetos metálicos toquen los terminales de las baterías en el dispositivo; pueden calentarse y provocar quemaduras.
- No transporte o coloque baterías junto con collares, horquillas para el cabello u otros objetos metálicos.
- No deje el dispositivo con alimentación por batería bajo la luz directa del sol durante un periodo prolongado, como por ejemplo sobre el tablero de un automóvil durante el verano.
- No sumerja las baterías en agua ni permita que se mojen.
- No conecte las baterías directamente a tomacorrientes de pared o tomas de encendedores de automóvil.
- No intente conectar a los terminales de las baterías, a menos que utilice un dispositivo de alojamiento aprobado por Microsoft.
- No golpee, arroje, pise o someta las baterías a una descarga eléctrica física intensa.
- No atraviese los recubrimientos de las baterías de ninguna forma.
- No intente desarmar ni alterar las baterías de ninguna forma.
- No recargue las baterías cerca de un incendio o en condiciones extremadamente calurosas.

USO Y CUIDADOS DE SU CONSOLA XBOX ONE

ADVERTENCIA: No intente realizar reparaciones
No intente desmontar, abrir, reparar ni alterar el producto, los accesorios o la fuente de alimentación. Al hacerlo existe el riesgo de descarga eléctrica, incendio u otros peligros, o daño para el sistema Xbox One o el sensor Kinect. Cualquier evidencia de que este dispositivo se haya intentado abrir y/o modificar, incluidas peladuras, perforaciones o retirada de cualquiera de las etiquetas, anulará la Garantía limitada y hará que la Xbox One no pueda optar a ser reparada por un servicio autorizado. Modificar la consola puede provocar su prohibición de acceso permanente a Xbox LIVE, necesario para jugar y disfrutar de otros usos de la consola.

Uso y limpieza

ADVERTENCIA: No permita que la consola o el sensor se mojen. Para reducir el riesgo de incendio o descarga eléctrica, no exponga la consola o el sensor a la lluvia o a otros tipos de humedad.

Úselos de acuerdo con estas instrucciones:
- No lo utilice cerca de ninguna fuente de calor.
- No coloque la consola en posición vertical.
- Usa sólo los accesorios especificados por Microsoft.
- Desconecte la fuente de alimentación de la consola para evitar que se encienda y apague o que el disco se abra durante la limpieza.
- Limpie únicamente el exterior de la Xbox One. Asegúrese de que no haya objetos insertados en las aberturas de ventilación.
- Utilice un paño seco; no utilice estropajos ásperos, detergentes, polvos para estropajos, disolventes (por ejemplo, alcohol, gasolina, diluyente de pintura o benceno), ni otros limpiadores líquidos o en aerosol.
- No utilice aire comprimido.
- No utilice dispositivos limpiadores de lectores de DVD.
- No intente limpiar los conectores.
- Limpie las patas de la consola y la superficie en la que descansa esta con un paño seco.
- Limpie la superficie sobre la cual descansa el sensor con un paño seco.

Evite el humo y el polvo
No utilice la consola en lugares con humo o polvo. El humo y el polvo pueden dañar la consola, en particular la unidad de disco.

Uso del disco
Para evitar que la unidad de disco se atasque y se dañen los discos o la consola:
- Extraiga los discos antes de mover la consola.
- Nunca use discos agrietados. Pueden romperse en el interior de la consola y atascarse o romper piezas internas.
- Devuelva siempre los discos a sus contenedores de almacenamiento cuando no se encuentren en la unidad de disco. No deje los discos bajo la luz directa del sol, cerca de una fuente de calor o sobre la Xbox One. Sujete siempre los discos por los bordes. Para limpiar los juegos y otros discos:
  - Sujete los discos por los bordes; no toque la superficie del disco con los dedos.
  - Limpie los discos con un trapo suave, pasándolo con suavidad desde el centro hacia afuera.
  - No utilice solventes, pueden dañar el disco. No utilice dispositivos para limpiar discos.

Objetos metálicos y adhesivos
No coloque elementos metálicos ni adhesivos cerca o encima de la Xbox One, ya que pueden interferir con el control, la red y los botones de encendido/apagado.

PRECAUCIÓN: Las imágenes fijas en los juegos de video se pueden "quemar" en algunas pantallas de televisor, con lo cual se genera una sombra permanente. Consulte el manual del fabricante del televisor antes de jugar.

ÁREA DE JUEGO

ADVERTENCIA: El juego con el sensor Kinect puede requerir diversas cantidades de movimiento. Para reducir el riesgo de lesiones o daño a la propiedad, tome las siguientes precauciones antes de jugar:

- Asegúrese de tener espacio suficiente para moverse con libertad.
- Mire en todas direcciones (a la derecha, izquierda, adelante, atrás, abajo y arriba). Asegúrese de que no hay nada con lo que pueda tropezar; como, por ejemplo, juguetes, muebles o alfombras sueltas.
- Asegúrese de que el área de juego esté lejos de ventanas, paredes, escaleras, etc.
- Preste atención a los niños y a las mascotas que se encuentren en el área. Si es necesario, aleje los objetos o las personas del área de juego.

2

Mientras juega:

- Mantenga una distancia adecuada del televisor para evitar el contacto.
- Conserve una distancia suficiente de los demás jugadores, observadores y mascotas. Esta distancia puede variar entre juegos, por lo tanto, tenga en cuenta la forma en la que juega cuando determine lo lejos que debe estar.
- Esté alerta frente a objetos o personas a las cuales podría golpear o con las que podría tropezarse. Durante el juego, es posible que se acerquen personas y objetos al área, por lo tanto, esté alerta a su alrededor.
- Asegúrese de usar un calzado adecuado mientras juega:
- Juegue a nivel del suelo con tracción suficiente para las actividades de juegos.
- Asegúrese de usar un calzado adecuado, o bien de estar descalzo mientras juega, si es adecuado. No use tacones, sandalias, etc.

No realice esfuerzos excesivos

El juego con el sensor Kinect puede requerir diversas cantidades de actividad física.

Consulte a un médico antes de usar el sensor si tiene alguna condición médica o problema que afecte su capacidad de realizar actividad física de manera segura, o bien si:

- está embarazada o puede estarlo,
- tiene un problema cardiaco, respiratorio, de la espalda, articulaciones o de carácter ortopédico,
- sufre de presión alta,
- tiene dificultad con el ejercicio físico, o
- si se le ha indicado que restrinja su actividad física.

Consulte a su médico antes de comenzar cualquier rutina de ejercicios o régimen de ejercicios que incluya el uso del sensor Kinect. No juegue bajo la influencia de alcohol o drogas y asegúrese de que su equilibrio y capacidades físicas sean suficientes para los movimientos que realiza mientras juega.

Descanse periódicamente

Deje de jugar si sus músculos, articulaciones u ojos se cansan o comienzan a doler.

Si experimenta una fatiga excesiva, náuseas, falta de aliento, opresión en el pecho, mareos, incomodidad o dolor, DETENGA EL USO DE INMEDIATO y consulte a un médico.

JUEGOS SALUDABLES

⚠ ADVERTENCIA: Advertencias de salud importantes sobre el uso de videojuegos

Crisis de fotosensibilidad

Un porcentaje muy pequeño de personas puede experimentar un ataque al exponerse a ciertas imágenes visuales, incluidas luces o patrones parpadeantes que pueden aparecer en juegos de video. Incluso personas sin historial de ataques o epilepsia pueden tener un problema no diagnosticado que puede provocar estos "ataques epilépticos fotosensibles" mientras observa juegos de video.

Estos ataques pueden tener una variedad de síntomas, incluidos vértigo, visión alterada, espasmos en los ojos o la cara, sacudidas o temblores de brazos o piernas, desorientación, confusión o pérdida momentánea de la conciencia. Los ataques también pueden provocar pérdida de la conciencia o convulsiones que pueden derivar en lesiones por caída o por chocar con objetos cercanos.

Deje de jugar inmediatamente y consulte a un médico si experimenta alguno de estos síntomas. Los padres deben estar atentos o preguntar a sus hijos acerca de los síntomas mencionados, los niños y adolescentes son más propensos a experimentar estos ataques que los adultos. El riesgo de ataques epilépticos fotosensibles se puede reducir si se toman las siguientes precauciones:

- Siéntese o aléjese más de la pantalla del televisor.
- Utilice una pantalla de televisión más pequeña.
- Juegue en una habitación con buena iluminación.
- No juegue cuando tenga sueño o esté cansado.
- Si alguno de sus familiares o usted mismo ha sufrido crisis o epilepsia con anterioridad, consulte a su médico antes de jugar.

Trastornos musculoesqueléticos

El uso de controles de juego, teclados, ratones u otros dispositivos de entrada electrónicos puede estar relacionado con lesiones y afecciones graves.

Al jugar juegos de video, como con muchas actividades, es posible que experimente incomodidad en las manos, los brazos, los hombros, el cuello u otras partes del cuerpo. Sin embargo, si experimenta síntomas como malestar persistente o repetido, dolor, punzadas, hormigueos, entumecimiento, sensación de quemazón o rigidez NO PASE POR ALTO ESTAS SEÑALES DE ADVERTENCIA. ACUDA DE INMEDIATO A UN PROFESIONAL DE LA SALUD CALIFICADO, aunque los síntomas aparezcan cuando no esté jugando. Este tipo de síntomas puede estar asociado con lesiones o afecciones dolorosas de los nervios, músculos, tendones, vasos sanguíneos y otras partes del cuerpo, que en ocasiones pueden ocasionar incapacidad permanente. Estos trastornos musculoesqueléticos (MSD) incluyen síndrome de túnel carpiano, tendinitis, tenosinovitis, síndromes de vibración y otras afecciones.

Aunque los investigadores aún no pueden responder muchas preguntas acerca de los MSD, existe un consenso general de que puede haber muchos factores ligados a su aparición, incluidas afecciones médicas o físicas, tensión y cómo se lidia con ella, salud en general y cómo una persona ubica y usa el cuerpo durante el trabajo y otras actividades (incluidos los juegos de video). Algunos estudios sugieren que la cantidad de tiempo que una persona realiza una actividad puede ser un factor.

En la Guía de juegos saludables en xbox.com/xboxone/playhealthy **puede encontrar algunas pautas que le pueden ayudar a trabajar y jugar de manera más cómoda y que posiblemente reduzcan el riesgo de experimentar un MSD.** Estas pautas abordan temas como:

- Colocación del cuerpo para utilizar posturas cómodas y normales.
- Relajación de manos, dedos y otras partes del cuerpo.
- Pausas entre juegos.
- Desarrollo de un estilo de vida saludable.

Si tiene alguna pregunta sobre la relación que pudieran tener su estilo de vida, actividades y condiciones médicas o físicas con los MSD, acuda a un profesional de la salud calificado.

⚠ ADVERTENCIA: Riesgo de asfixia

Este dispositivo puede contener piezas pequeñas que supongan un riesgo de asfixia para niños menores de 3 años. Mantenga las piezas pequeñas fuera del alcance de los niños.

Asegúrese de que los niños jueguen de forma segura

Asegúrese de que los niños que utilicen el accesorio de Xbox One junto con la consola Xbox One y el sensor Kinect jueguen de manera segura y dentro de sus límites y que comprendan el uso correcto del sistema.

No use accesorios sin licencia u objetos de utilería no autorizados u otros objetos con el sensor Kinect.

El uso de estos accesorios u objetos puede generar como resultado lesiones a usted u otras personas y/o daños al sensor o a otra propiedad. El uso de accesorios no autorizados infringe la Licencia de software y puede anular la Garantía limitada.

Evite el reflejo

Para minimizar las molestias que pueden generar los reflejos, pruebe con lo siguiente:

A2. Body Source View script

```
1.  using UnityEngine;
2.  using System.Collections.Generic;
3.  using Kinect = Windows.Kinect;
4.  using Windows.Kinect;
5.  using JointsAngles;
6.
7.  using System;
8.  using System.IO;
9.
10. public class BodySourceView : MonoBehaviour
11. {
12.     public Transform[] articulacions;
13.
14.     public Material BoneMaterial;
15.     public GameObject BodySourceManager;
16.
17.     public double[,] orientationsarray;
18.
19.     UnityEngine.Quaternion or1_unity;
20.     UnityEngine.Quaternion or2_unity;
21.     UnityEngine.Quaternion or3_unity;
22.     UnityEngine.Quaternion or4_unity;
23.     UnityEngine.Quaternion or5_unity;
24.     UnityEngine.Quaternion or6_unity;
25.     UnityEngine.Quaternion or7_unity;
26.     UnityEngine.Quaternion or8_unity;
27.     UnityEngine.Quaternion or9_unity;
28.     UnityEngine.Quaternion or10_unity;
29.     UnityEngine.Quaternion or11_unity;
30.     UnityEngine.Quaternion or12_unity;
```

```
31.     UnityEngine.Quaternion or13_unity;
32.     UnityEngine.Quaternion or14_unity;
33.     UnityEngine.Quaternion or15_unity;
34.     UnityEngine.Quaternion or16_unity;
35.     UnityEngine.Quaternion or17_unity;
36.     UnityEngine.Quaternion or18_unity;
37.     UnityEngine.Quaternion or19_unity;
38.     UnityEngine.Quaternion or20_unity;
39.     UnityEngine.Quaternion or21_unity;
40.     UnityEngine.Quaternion or22_unity;
41.
42.
43.     private Dictionary<ulong, GameObject> _Bodies = new Dictionary<ulong,
   GameObject>();
44.     private BodySourceManager _BodyManager;
45.
46.     private Dictionary<Kinect.JointType, Kinect.JointType> _BoneMap = new
   Dictionary<Kinect.JointType, Kinect.JointType>()
47.     {
48.         { Kinect.JointType.FootLeft, Kinect.JointType.AnkleLeft },
49.         { Kinect.JointType.AnkleLeft, Kinect.JointType.KneeLeft },
50.         { Kinect.JointType.KneeLeft, Kinect.JointType.HipLeft },
51.         { Kinect.JointType.HipLeft, Kinect.JointType.SpineBase },
52.
53.         { Kinect.JointType.FootRight, Kinect.JointType.AnkleRight },
54.         { Kinect.JointType.AnkleRight, Kinect.JointType.KneeRight },
55.         { Kinect.JointType.KneeRight, Kinect.JointType.HipRight },
56.         { Kinect.JointType.HipRight, Kinect.JointType.SpineBase },
57.
58.         { Kinect.JointType.HandTipLeft, Kinect.JointType.HandLeft },
59.         { Kinect.JointType.ThumbLeft, Kinect.JointType.HandLeft },
60.         { Kinect.JointType.HandLeft, Kinect.JointType.WristLeft },
61.         { Kinect.JointType.WristLeft, Kinect.JointType.ElbowLeft },
```

```
62.         { Kinect.JointType.ElbowLeft, Kinect.JointType.ShoulderLeft },
63.         { Kinect.JointType.ShoulderLeft, Kinect.JointType.SpineShoulder
   },
64.
65.         { Kinect.JointType.HandTipRight, Kinect.JointType.HandRight },
66.         { Kinect.JointType.ThumbRight, Kinect.JointType.HandRight },
67.         { Kinect.JointType.HandRight, Kinect.JointType.WristRight },
68.         { Kinect.JointType.WristRight, Kinect.JointType.ElbowRight },
69.         { Kinect.JointType.ElbowRight, Kinect.JointType.ShoulderRight },
70.         { Kinect.JointType.ShoulderRight, Kinect.JointType.SpineShoulder
   },
71.
72.         { Kinect.JointType.SpineBase, Kinect.JointType.SpineMid },
73.         { Kinect.JointType.SpineMid, Kinect.JointType.SpineShoulder },
74.         { Kinect.JointType.SpineShoulder, Kinect.JointType.Neck },
75.         { Kinect.JointType.Neck, Kinect.JointType.Head },
76.     };
77.
78.
79.
80.     public void Update()
81.     {
82.
83.
84.         if (BodySourceManager == null)
85.         {
86.             return;
87.         }
88.
89.         _BodyManager =
   BodySourceManager.GetComponent<BodySourceManager>();
90.         if (_BodyManager == null)
91.         {
```

```
92.             return;
93.         }
94.
95.         Kinect.Body[] data = _BodyManager.GetData();
96.         if (data == null)
97.         {
98.             return;
99.         }
100.
101.            List<ulong> trackedIds = new List<ulong>();
102.            foreach (var body in data)
103.            {
104.                if (body == null)
105.                {
106.                    continue;
107.                }
108.
109.                if (body.IsTracked)
110.                {
111.                    trackedIds.Add(body.TrackingId);
112.                }
113.            }
114.
115.            List<ulong> knownIds = new List<ulong>(_Bodies.Keys);
116.
117.            // First delete untracked bodies
118.            foreach (ulong trackingId in knownIds)
119.            {
120.                if (!trackedIds.Contains(trackingId))
121.                {
122.                    Destroy(_Bodies[trackingId]);
123.                    _Bodies.Remove(trackingId);
```

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

```
124.                    }
125.                }
126.
127.                foreach (var Body in data)
128.                {
129.                    if (Body == null)
130.                    {
131.                        continue;
132.                    }
133.
134.                    if (Body.IsTracked)
135.                    {
136.                        if (!_Bodies.ContainsKey(Body.TrackingId))
137.                        {
138.                            _Bodies[Body.TrackingId] =
    CreateBodyObject(Body.TrackingId);
139.                        }
140.
141.                        RefreshBodyObject(Body, _Bodies[Body.TrackingId]);
142.
143.         // ************************** KINECT JOINT
    ORIENTATIONS****************************************//
144.
145.
146.
147.
148.                        var orientation1 =
    Body.JointOrientations[JointType.Head].Orientation;
149.
150.                        var orientation2 =
    Body.JointOrientations[JointType.Neck].Orientation;
151.
152.                        var orientation3 =
    Body.JointOrientations[JointType.SpineMid].Orientation;
```

```
153.

154.                    var orientation4 =
    Body.JointOrientations[JointType.ShoulderLeft].Orientation;

155.

156.                    var orientation5 =
    Body.JointOrientations[JointType.ShoulderRight].Orientation;

157.

158.                    var orientation6 =
    Body.JointOrientations[JointType.ElbowLeft].Orientation;

159.

160.                    var orientation7 =
    Body.JointOrientations[JointType.ElbowRight].Orientation;

161.

162.                    var orientation8 =
    Body.JointOrientations[JointType.WristLeft].Orientation;

163.

164.                    var orientation9 =
    Body.JointOrientations[JointType.WristRight].Orientation;

165.

166.                    var orientation10 =
    Body.JointOrientations[JointType.HipLeft].Orientation;

167.

168.                    var orientation11 =
    Body.JointOrientations[JointType.HipRight].Orientation;

169.

170.                    var orientation12 =
    Body.JointOrientations[JointType.KneeLeft].Orientation;

171.

172.                    var orientation13 =
    Body.JointOrientations[JointType.KneeRight].Orientation;

173.

174.                    var orientation14 =
    Body.JointOrientations[JointType.SpineBase].Orientation;

175.

176.                    var orientation15 =
    Body.JointOrientations[JointType.AnkleLeft].Orientation;

177.
```

```
178.                     var orientation16 =
    Body.JointOrientations[JointType.AnkleRight].Orientation;

179.

180.                     var orientation17 =
    Body.JointOrientations[JointType.FootLeft].Orientation;

181.

182.                     var orientation18 =
    Body.JointOrientations[JointType.FootRight].Orientation;

183.

184.                     var orientation19 =
    Body.JointOrientations[JointType.HandLeft].Orientation;

185.

186.                     var orientation20 =
    Body.JointOrientations[JointType.HandRight].Orientation;

187.

188.                     var orientation21 =
    Body.JointOrientations[JointType.ThumbLeft].Orientation;

189.

190.                     var orientation22 =
    Body.JointOrientations[JointType.ThumbRight].Orientation;

191.

192.

193.

194.

195.                     //*********************************APPLYING
    ROTATIONS TO AVATAR***********************************//

196.

197.

198.                     // HEAD

199.                     or1_unity.x = orientation1.X;

200.                     or1_unity.y = orientation1.Y;

201.                     or1_unity.z = orientation1.Z;

202.                     or1_unity.w = orientation1.W;

203.

204.                     Quaternion or1_unity_aux;
```

```
205.                        Quaternion or1_unity_aux2;
206.                        Quaternion or1_new;
207.
208.                        or1_unity_aux = Quaternion.AngleAxis(90,
    Vector3.back);
209.                        or1_unity_aux2 = Quaternion.AngleAxis(180,
    Vector3.up);
210.                        or1_new = or1_unity_aux * or1_unity_aux2 *
    or1_unity;
211.                        articulacions[0].rotation = or1_new;
212.
213.                        //NECK
214.                        or2_unity.x = orientation2.X;
215.                        or2_unity.y = orientation2.Y;
216.                        or2_unity.z = orientation2.Z;
217.                        or2_unity.w = orientation2.W;
218.                        Quaternion or2_unity_aux;
219.                        Quaternion or2_unity_aux2;
220.                        Quaternion or2_new;
221.
222.                        or2_unity_aux = Quaternion.AngleAxis(90,
    Vector3.back);
223.                        or2_unity_aux2 = Quaternion.AngleAxis(180,
    Vector3.up);
224.                        or2_new = or2_unity_aux * or2_unity_aux2 *
    or2_unity;
225.                        articulacions[1].rotation = or2_new;
226.
227.                        //SPineMid
228.                        or3_unity.x = orientation3.X;
229.                        or3_unity.y = orientation3.Y;
230.                        or3_unity.z = orientation3.Z;
231.                        or3_unity.w = orientation3.W;
232.
233.                        Quaternion or3_unity_aux;
```

```
234.                        Quaternion or3_unity_aux2;
235.                        Quaternion or3_new;
236.
237.                        or3_unity_aux = Quaternion.AngleAxis(90,
    Vector3.back);
238.                        or3_unity_aux2 = Quaternion.AngleAxis(180,
    Vector3.up);
239.                        or3_new = or3_unity_aux * or3_unity_aux2 *
    or3_unity;
240.                        articulacions[2].rotation = or3_new;
241.
242.                        //ShoulderLeft
243.                        or4_unity.x = orientation4.X;
244.                        or4_unity.y = orientation4.Y;
245.                        or4_unity.z = orientation4.Z;
246.                        or4_unity.w = orientation4.W;
247.
248.                        Quaternion or4_unity_aux2;
249.                        Quaternion or4_unity_aux3;
250.                        Quaternion or4_new;
251.
252.                        or4_unity_aux2 = Quaternion.AngleAxis(180,
    Vector3.up);
253.                        or4_unity_aux3 = Quaternion.AngleAxis(90,
    Vector3.back);
254.                        or4_new = or4_unity * or4_unity_aux3;
255.                        articulacions[3].rotation = or4_new;
256.
257.
258.                        //ShoulderRight
259.                        or5_unity.x = orientation5.X;
260.                        or5_unity.y = orientation5.Y;
261.                        or5_unity.z = orientation5.Z;
262.                        or5_unity.w = orientation5.W;
```

```
263.
264.                        Quaternion or5_unity_aux;
265.                        Quaternion or5_unity_aux3;
266.                        Quaternion or5_new;
267.
268.                        or5_unity_aux = Quaternion.AngleAxis(180,
    Vector3.right);
269.                        or5_unity_aux3 = Quaternion.AngleAxis(90,
    Vector3.forward);
270.                        or5_new = or5_unity_aux * or5_unity *
    or5_unity_aux3;
271.                        articulacions[4].rotation = or5_new;
272.
273.                        //ElbowLeft
274.                        or6_unity.x = orientation6.X;
275.                        or6_unity.y = orientation6.Y;
276.                        or6_unity.z = orientation6.Z;
277.                        or6_unity.w = orientation6.W;
278.
279.                        Quaternion or6_unity_aux;
280.                        Quaternion or6_unity_aux2;
281.                        Quaternion or6_unity_aux3;
282.                        Quaternion or6_new;
283.
284.                        or6_unity_aux = Quaternion.AngleAxis(180,
    Vector3.back);
285.                        or6_unity_aux2 = Quaternion.AngleAxis(180,
    Vector3.up);
286.                        or6_unity_aux3 = Quaternion.AngleAxis(180,
    Vector3.right);
287.                        or6_new = or6_unity_aux * or6_unity_aux2 *
    or6_unity_aux3 * or6_unity;
288.                        articulacions[5].rotation = or6_new;
289.
290.
```

```
291.                        //ElbowRight
292.                        or7_unity.x = orientation7.X;
293.                        or7_unity.y = orientation7.Y;
294.                        or7_unity.z = orientation7.Z;
295.                        or7_unity.w = orientation7.W;
296.
297.                        Quaternion or7_unity_aux;
298.                        Quaternion or7_unity_aux2;
299.                        Quaternion or7_unity_aux3;
300.                        Quaternion or7_new;
301.
302.                        or7_unity_aux = Quaternion.AngleAxis(30,
    Vector3.back);
303.                        or7_unity_aux2 = Quaternion.AngleAxis(180,
    Vector3.up);
304.                        or7_unity_aux3 = Quaternion.AngleAxis(0,
    Vector3.right);
305.                        or7_new = or7_unity_aux * or7_unity_aux2 * or7_unity
    * or7_unity_aux3;
306.                        articulacions[6].rotation = or7_new;
307.
308.                        //WristLeft
309.                        or8_unity.x = orientation8.X;
310.                        or8_unity.y = orientation8.Y;
311.                        or8_unity.z = orientation8.Z;
312.                        or8_unity.w = orientation8.W;
313.
314.                        Quaternion or8_unity_aux;
315.                        Quaternion or8_unity_aux2;
316.                        Quaternion or8_new;
317.
318.                        or8_unity_aux = Quaternion.AngleAxis(90,
    Vector3.back);
319.                        or8_unity_aux2 = Quaternion.AngleAxis(180,
    Vector3.up);
```

```
320.                            or8_new = or8_unity_aux * or8_unity_aux2 *
    or8_unity;
321.                            articulacions[7].rotation = or8_new;
322.
323.
324.                            //WristRight
325.                            or9_unity.x = orientation9.X;
326.                            or9_unity.y = orientation9.Y;
327.                            or9_unity.z = orientation9.Z;
328.                            or9_unity.w = orientation9.W;
329.
330.                            Quaternion or9_unity_aux;
331.                            Quaternion or9_unity_aux2;
332.                            Quaternion or9_new;
333.
334.                            or9_unity_aux = Quaternion.AngleAxis(90,
    Vector3.back);
335.                            or9_unity_aux2 = Quaternion.AngleAxis(180,
    Vector3.up);
336.                            or9_new = or9_unity_aux * or9_unity_aux2 *
    or9_unity;
337.                            articulacions[8].rotation = or9_new;
338.
339.                            //HipLeft
340.                            or10_unity.x = orientation10.X;
341.                            or10_unity.y = orientation10.Y;
342.                            or10_unity.z = orientation10.Z;
343.                            or10_unity.w = orientation10.W;
344.
345.                            Quaternion or10_unity_aux;
346.                            Quaternion or10_unity_aux2;
347.                            Quaternion or10_unity_aux3;
348.                            Quaternion or10_new;
```

```
349.
350.                  or10_unity_aux = Quaternion.AngleAxis(180,
    Vector3.forward);
351.                  or10_unity_aux2 = Quaternion.AngleAxis(180,
    Vector3.right);
352.                  or10_unity_aux3 = Quaternion.AngleAxis(180,
    Vector3.down);
353.                  or10_new = or10_unity * or10_unity_aux *
    or10_unity_aux2;
354.                  articulacions[9].rotation = or10_new;
355.
356.                  //HipRight
357.                  or11_unity.x = orientation11.X;
358.                  or11_unity.y = orientation11.Y;
359.                  or11_unity.z = orientation11.Z;
360.                  or11_unity.w = orientation11.W;
361.
362.
363.                  Quaternion or11_unity_aux;
364.                  Quaternion or11_unity_aux2;
365.                  Quaternion or11_unity_aux3;
366.                  Quaternion or11_new;
367.
368.                  or11_unity_aux = Quaternion.AngleAxis(90,
    Vector3.up);
369.                  or11_unity_aux2 = Quaternion.AngleAxis(90,
    Vector3.forward);
370.                  or11_unity_aux3 = Quaternion.AngleAxis(180,
    Vector3.right);
371.                  or11_new = or11_unity * or11_unity_aux *
    or11_unity_aux2;
372.                  articulacions[10].rotation = or11_new;
373.
374.                  //KneeLeft
375.                  or12_unity.x = orientation12.X;
```

```
376.                        or12_unity.y = orientation12.Y;
377.                        or12_unity.z = orientation12.Z;
378.                        or12_unity.w = orientation12.W;
379.
380.                        Quaternion or12_unity_aux;
381.                        Quaternion or12_unity_aux2;
382.                        Quaternion or12_unity_aux3;
383.                        Quaternion or12_new;
384.
385.                        or12_unity_aux = Quaternion.AngleAxis(180,
    Vector3.forward);
386.                        or12_unity_aux2 = Quaternion.AngleAxis(180,
    Vector3.right);
387.                        or12_unity_aux3 = Quaternion.AngleAxis(180,
    Vector3.down);
388.                        or12_new = or12_unity * or12_unity_aux *
    or12_unity_aux2;
389.                        articulacions[11].rotation = or12_new;
390.
391.                        //KneeRight
392.                        or13_unity.x = orientation13.X;
393.                        or13_unity.y = orientation13.Y;
394.                        or13_unity.z = orientation13.Z;
395.                        or13_unity.w = orientation13.W;
396.
397.                        Quaternion or13_unity_aux;
398.                        Quaternion or13_unity_aux2;
399.                        Quaternion or13_unity_aux3;
400.                        Quaternion or13_new;
401.
402.                        or13_unity_aux = Quaternion.AngleAxis(90,
    Vector3.up);
403.                        or13_unity_aux2 = Quaternion.AngleAxis(90,
    Vector3.forward);
```

```
404.                        or13_unity_aux3 = Quaternion.AngleAxis(180,
   Vector3.right);
405.                        or13_new = or13_unity * or13_unity_aux *
   or13_unity_aux2;
406.                        articulacions[12].rotation = or13_new;
407.
408.                        //SpineBase
409.                        or14_unity.x = orientation14.X;
410.                        or14_unity.y = orientation14.Y;
411.                        or14_unity.z = orientation14.Z;
412.                        or14_unity.w = orientation14.W;
413.
414.                        Quaternion or14_unity_aux;
415.                        Quaternion or14_unity_aux2;
416.                        Quaternion or14_new;
417.
418.                        or14_unity_aux = Quaternion.AngleAxis(90,
   Vector3.back);
419.                        or14_unity_aux2 = Quaternion.AngleAxis(180,
   Vector3.up);
420.                        or14_new = or14_unity_aux * or14_unity_aux2 *
   or14_unity;
421.                        articulacions[13].rotation = or14_new;
422.
423.                        //AnkleLeft
424.                        or15_unity.x = orientation15.X;
425.                        or15_unity.y = orientation15.Y;
426.                        or15_unity.z = orientation15.Z;
427.                        or15_unity.w = orientation15.W;
428.
429.                        Quaternion or15_unity_aux;
430.                        Quaternion or15_unity_aux2;
431.                        Quaternion or15_unity_aux3;
432.                        Quaternion or15_new;
```

```
433.
434.                        or15_unity_aux = Quaternion.AngleAxis(180,
    Vector3.forward);
435.                        or15_unity_aux2 = Quaternion.AngleAxis(180,
    Vector3.right);
436.                        or15_unity_aux3 = Quaternion.AngleAxis(180,
    Vector3.down);
437.                        or15_new = or15_unity * or15_unity_aux *
    or15_unity_aux2;
438.                        articulacions[14].rotation = or15_new;
439.
440.                        //AnkleRight
441.                        or16_unity.x = orientation16.X;
442.                        or16_unity.y = orientation16.Y;
443.                        or16_unity.z = orientation16.Z;
444.                        or16_unity.w = orientation16.W;
445.
446.                        Quaternion or16_unity_aux;
447.                        Quaternion or16_unity_aux2;
448.                        Quaternion or16_unity_aux3;
449.                        Quaternion or16_new;
450.
451.                        or16_unity_aux = Quaternion.AngleAxis(90,
    Vector3.up);
452.                        or16_unity_aux2 = Quaternion.AngleAxis(90,
    Vector3.forward);
453.                        or16_unity_aux3 = Quaternion.AngleAxis(180,
    Vector3.right);
454.                        or16_new = or16_unity * or16_unity_aux *
    or16_unity_aux2;
455.                        articulacions[15].rotation = or16_new;
456.
457.                        //FootLeft
458.                        or17_unity.x = orientation17.X;
459.                        or17_unity.y = orientation17.Y;
```

```
460.                        or17_unity.z = orientation17.Z;
461.                        or17_unity.w = orientation17.W;
462.
463.
464.                        Quaternion or17_unity_aux;
465.                        Quaternion or17_unity_aux2;
466.                        Quaternion or17_unity_aux3;
467.                        Quaternion or17_new;
468.
469.                        or17_unity_aux = Quaternion.AngleAxis(180,
    Vector3.forward);
470.                        or17_unity_aux2 = Quaternion.AngleAxis(180,
    Vector3.right);
471.                        or17_unity_aux3 = Quaternion.AngleAxis(180,
    Vector3.down);
472.                        or17_new = or17_unity * or17_unity_aux *
    or17_unity_aux2;
473.                        articulacions[16].rotation = or17_new;
474.
475.                        //FootRight
476.                        or18_unity.x = orientation18.X;
477.                        or18_unity.y = orientation18.Y;
478.                        or18_unity.z = orientation18.Z;
479.                        or18_unity.w = orientation18.W;
480.
481.                        Quaternion or18_unity_aux;
482.                        Quaternion or18_unity_aux2;
483.                        Quaternion or18_unity_aux3;
484.                        Quaternion or18_new;
485.
486.                        or18_unity_aux = Quaternion.AngleAxis(90,
    Vector3.up);
487.                        or18_unity_aux2 = Quaternion.AngleAxis(90,
    Vector3.forward);
```

```
488.                    or18_unity_aux3 = Quaternion.AngleAxis(180,
   Vector3.right);

489.                    or18_new = or18_unity * or18_unity_aux *
   or18_unity_aux2;

490.                    articulacions[17].rotation = or18_new;

491.

492.                    //HandLeft

493.                    or19_unity.x = orientation19.X;

494.                    or19_unity.y = orientation19.Y;

495.                    or19_unity.z = orientation19.Z;

496.                    or19_unity.w = orientation19.W;

497.

498.                    Quaternion or19_unity_aux2;

499.                    Quaternion or19_unity_aux3;

500.                    Quaternion or19_new;

501.

502.                    or19_unity_aux2 = Quaternion.AngleAxis(180,
   Vector3.down);

503.                    or19_unity_aux3 = Quaternion.AngleAxis(180,
   Vector3.right);

504.                    or19_new = or19_unity_aux2 * or19_unity_aux3 *
   or19_unity;

505.                    articulacions[18].rotation = or19_new;

506.

507.                    //HandRight

508.                    or20_unity.x = orientation20.X;

509.                    or20_unity.y = orientation20.Y;

510.                    or20_unity.z = orientation20.Z;

511.                    or20_unity.w = orientation20.W;

512.

513.                    Quaternion or20_unity_aux;

514.                    Quaternion or20_unity_aux3;

515.                    Quaternion or20_new;

516.
```

```
517.                    or20_unity_aux = Quaternion.AngleAxis(180,
    Vector3.forward);
518.                    or20_unity_aux3 = Quaternion.AngleAxis(180,
    Vector3.right);
519.                    or20_new = or20_unity_aux * or20_unity *
    or20_unity_aux3;
520.                    articulacions[19].rotation = or20_new;
521.
522.                    //ThumbLeft
523.                    or21_unity.x = orientation21.X;
524.                    or21_unity.y = orientation21.Y;
525.                    or21_unity.z = orientation21.Z;
526.                    or21_unity.w = orientation21.W;
527.
528.                    Quaternion or21_unity_aux2;
529.                    Quaternion or21_unity_aux3;
530.                    Quaternion or21_new;
531.
532.                    or21 unity aux2 = Quaternion.AngleAxis(180,
    Vector3.down);
533.                    or21_unity_aux3 = Quaternion.AngleAxis(180,
    Vector3.right);
534.                    or21_new = or21_unity_aux2 * or21_unity_aux3 *
    or21_unity;
535.                    articulacions[20].rotation = or21_new;
536.
537.                    //ThumbRight
538.                    or22_unity.x = orientation22.X;
539.                    or22_unity.y = orientation22.Y;
540.                    or22_unity.z = orientation22.Z;
541.                    or22_unity.w = orientation22.W;
542.
543.                    Quaternion or22_unity_aux;
544.                    Quaternion or22_unity_aux3;
545.                    Quaternion or22_new;
```

```
546.
547.                    or22_unity_aux = Quaternion.AngleAxis(180,
   Vector3.forward);
548.                    or22_unity_aux3 = Quaternion.AngleAxis(180,
   Vector3.right);
549.                    or22_new = or22_unity_aux * or22_unity *
   or22_unity_aux3;
550.                    articulacions[21].rotation = or22_new;
551.
552.                    //************************************Orientations
   array**********************************//
553.
554.                    orientationsarray = new double[22, 4]{{
   or1_new.x,or1_new.y,or1_new.z,or1_new.w},
555.                                                {
   or2_new.x,or2_new.y,or2_new.z,or2_new.w },
556.                                                {
   or3_new.x,or3_new.y,or3_new.z,or3_new.w },
557.                                                {
   or4_new.x,or4_new.y,or4_new.z,or4_new.w },
558.                                                {
   or5_new.x,or5_new.y,or5_new.z,or5_new.w },
559.                                                {
   or6_new.x,or6_new.y,or6_new.z,or6_new.w },
560.                                                {
   or7_new.x,or7_new.y,or7_new.z,or7_new.w },
561.                                                {
   or8_new.x,or8_new.y,or8_new.z,or8_new.w },
562.                                                {
   or9_new.x,or9_new.y,or9_new.z,or9_new.w },
563.                                                {
   or10_new.x,or10_new.y,or10_new.z,or10_new.w },
564.                                                {
   or11_new.x,or11_new.y,or11_new.z,or11_new.w },
565.                                                {
   or12_new.x,or12_new.y,or12_new.z,or12_new.w },
566.                                                {
   or13_new.x,or13_new.y,or13_new.z,or13_new.w },
567.                                                {
   or14_new.x,or14_new.y,or14_new.z,or14_new.w },
```

```
568.                                              {
    or15_new.x,or15_new.y,or15_new.z,or15_new.w },

569.                                              {
    or16_new.x,or16_new.y,or16_new.z,or16_new.w },

570.                                              {
    or17_new.x,or17_new.y,or17_new.z,or17_new.w },

571.                                              {
    or18_new.x,or18_new.y,or18_new.z,or18_new.w },

572.                                              {
    or19_new.x,or19_new.y,or19_new.z,or19_new.w },

573.                                              {
    or20_new.x,or20_new.y,or20_new.z,or20_new.w },

574.                                              {
    or21_new.x,or21_new.y,or21_new.z,or21_new.w },

575.                                              {
    or22_new.x,or22_new.y,or22_new.z,or22_new.w }

576.                    };

577.                    //************************************* TXT file
    creation ***********************************//

578.

579.                    for (int i = 0; i < 22; i++)

580.                    {

581.                        for (int j = 0; j < 4; j++)

582.                        {

583.                            double datos1 =
    Math.Round(orientationsarray[i, j], 4);

584.

585.                            string datos = datos1.ToString();

586.                            File.AppendAllText(@"C:\Users\USUARIO\Docume
    nts\TFG\virtualMirror_3108\orientationsarray.txt", datos);

587.

588.                        }

589.                        File.AppendAllText(@"C:\Users\USUARIO\Documents\
    TFG\virtualMirror_3108\orientationsarray.txt", Environment.NewLine);

590.                    }

591.                    File.AppendAllText(@"C:\Users\USUARIO\Documents\TFG\
    virtualMirror_3108\orientationsarray.txt", Environment.NewLine);

592.
```

```
593.                    }
594.
595.
596.                }
597.
598.
599.            }
600.
601.            private GameObject CreateBodyObject(ulong id)
602.            {
603.                GameObject body = new GameObject("Body:" + id);
604.
605.                for (Kinect.JointType jt = Kinect.JointType.SpineBase; jt <=
    Kinect.JointType.ThumbRight; jt++)
606.                {
607.                    GameObject jointObj =
    GameObject.CreatePrimitive(PrimitiveType.Cube);
608.
609.                    LineRenderer lr = jointObj.AddComponent<LineRenderer>();
610.                    lr.SetVertexCount(2);
611.                    lr.material = BoneMaterial;
612.                    lr.SetWidth(0.05f, 0.05f);
613.
614.                    jointObj.transform.localScale = new Vector3(0.3f, 0.3f,
    0.3f);
615.                    jointObj.name = jt.ToString();
616.                    jointObj.transform.parent = body.transform;
617.                }
618.
619.                return body;
620.            }
621.
622.
```

```
623.
624.
625.
626.         private void RefreshBodyObject(Kinect.Body body, GameObject
   bodyObject)
627.         {
628.             for (Kinect.JointType jt = Kinect.JointType.SpineBase; jt <=
   Kinect.JointType.ThumbRight; jt++)
629.             {
630.                 Kinect.Joint sourceJoint = body.Joints[jt];
631.                 Kinect.Joint? targetJoint = null;
632.
633.                 if (_BoneMap.ContainsKey(jt))
634.                 {
635.                     targetJoint = body.Joints[_BoneMap[jt]];
636.                 }
637.
638.                 Transform jointObj =
   bodyObject.transform.Find(jt.ToString());
639.                 jointObj.localPosition =
   GetVector3FromJoint(sourceJoint);
640.
641.                 LineRenderer lr = jointObj.GetComponent<LineRenderer>();
642.                 if (targetJoint.HasValue)
643.                 {
644.                     lr.SetPosition(0, jointObj.localPosition);
645.                     lr.SetPosition(1,
   GetVector3FromJoint(targetJoint.Value));
646.                     lr.SetColors(GetColorForState(sourceJoint.TrackingSt
   ate), GetColorForState(targetJoint.Value.TrackingState));
647.                 }
648.                 else
649.                 {
650.                     lr.enabled = false;
```

```
651.                     }
652.                 }
653.             }
654.
655.             private static Color GetColorForState(Kinect.TrackingState
    state)
656.             {
657.                 switch (state)
658.                 {
659.                     case Kinect.TrackingState.Tracked:
660.                         return Color.green;
661.
662.                     case Kinect.TrackingState.Inferred:
663.                         return Color.red;
664.
665.                     default:
666.                         return Color.black;
667.                 }
668.             }
669.
670.             private static Vector3 GetVector3FromJoint(Kinect.Joint joint)
671.             {
672.                 return new Vector3(joint.Position.X * 10, joint.Position.Y *
    10, joint.Position.Z * 10);
673.             }
674.
675.
676.
677.
678.
679.
```

A3. Body Source Manager script

```
1.  using UnityEngine;
2.  using System.Collections;
3.  using Windows.Kinect;
4.
5.  public class BodySourceManager : MonoBehaviour
6.  {
7.      private KinectSensor _Sensor;
8.      private BodyFrameReader _Reader;
9.      private Body[] _Data = null;
10.
11.     public Body[] GetData()
12.     {
13.         return _Data;
14.     }
15.
16.     void Start ()
17.     {
18.         _Sensor = KinectSensor.GetDefault();
19.
20.         if (_Sensor != null)
21.         {
22.             _Reader = _Sensor.BodyFrameSource.OpenReader();
23.
24.             if (!_Sensor.IsOpen)
25.             {
26.                 _Sensor.Open();
27.             }
28.         }
29.     }
30.
31.     void Update ()
32.     {
```


UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

```
33.         if (_Reader != null)
34.         {
35.             var frame = _Reader.AcquireLatestFrame();
36.             if (frame != null)
37.             {
38.                 if (_Data == null)
39.                 {
40.                     _Data = new Body[_Sensor.BodyFrameSource.BodyCount];
41.                 }
42.
43.                 frame.GetAndRefreshBodyData(_Data);
44.
45.                 frame.Dispose();
46.                 frame = null;
47.             }
48.         }
49.     }
50.     void OnApplicationQuit()
51.     {
52.         if (_Reader != null)
53.         {
54.             _Reader.Dispose();
55.             _Reader = null;
56.         }
57.         if (_Sensor != null)
58.         {
59.             if (_Sensor.IsOpen)
60.             {
61.                 _Sensor.Close();
62.             }
63.
64.             _Sensor = null;
65.         }
66.     }
```