GESTURE RECOGNITION USING LEAP MOTION

MSc PROJECT

IOANNIS ZIFOS

SUPERVISOR

Dr IRINA VOICULESCU

DEPARTMENT OF COMPUTER SCIENCE,
UNIVERSITY OF OXFORD

2015

Abstract

This project created a virtual environment to be used during the rehabilitation process of people who suffer from apraxia: a post stroke implication. Patients suffering from apraxia have lost of the ability to co-ordinate and perform skilled, purposeful movements and gestures with normal accuracy.

In order to create this program, two scoring functions were created, each of which takes under consideration different criteria. Both scoring functions appear to be equally suitable. The game was tested on a number of different healthy controls with great success and neither was preferred by a majority group.

The program is based on several parameters which offer the ability to adapt the overall score to the needs of each user. Empirical results indicate that the scores are proportional to the matching of the user's gesture with the goal gesture.

The outcome is a user-friendly game-like environment which can be used easily by both patients and healthy people.

Contents

1. Introduction

No one can deny that technology has been progressing sharply these last decades. Computers play a crucial role in our everyday life. We use our laptops, tablets and mobile phones to perform tasks that would just seem impossible a few years ago. We can communicate with a person on the other side of the Earth just by touching the screen of a device smaller than our palm.

Ground breaking technologies are constantly being created. Some of them allow us to interact with our computers using movements, speech or even brain waves. Microsoft Kinect [11], Wii [12] or Playstation Move [13] are some of the technologies that allow the user to use remote controls that are operated through motion, or use his whole body as a controller.

Computer Science has also a profound effect on health care. New technologies are created and used for medical purposes, ranging from Nanotechnology to Robotic devices. Apart from the diagnosis and treatment of an illness, the rehabilitation process is certainly essential. The existence of a rehabilitation program for every medical condition is of vital importance.

Our purpose in this project is to create an environment which can help the recovery of patients who suffer from limb apraxia. We will use the Leap Motion sensor to keep track of the user's hand. We will give the patient several gesture challenges and he will try to mimic them[1]. Depending on how well he has performed, he will receive a score. Numerous requirements will have to be met and several parameters will have to be created in order to score each gesture. All of which will be explained in detail in the following sections.

1.1 Motivation

A Stroke, or "brain attack", is when poor blood flow to the brain results in cell death. Every year, 15 million people worldwide suffer a stroke. Stroke is the second leading cause of disability, after dementia [9]. Specifically every year in the UK, around 110,000 people have a stroke and it is the third largest cause of death, after heart disease and cancer [10].

Nearly six million die and another five million are left permanently disabled worldwide. When brain cells die during a stroke, abilities controlled by that area of the brain such as memory and muscle control are lost. One of the possible disabilities that can follow a stroke is Apraxia.

Apraxia is a motor disorder caused by damage to the brain, in which someone has difficulty with the motor planning to perform tasks or movements when asked, even if the request is understood, the patient is willing to perform the task or the muscles needed to perform the task work properly.

[1] The third personal pronoun is used in its masculine gender throughout the text for simplicity.

There are several forms of apraxia: Gait apraxia, Apraxia of speech, Buccofacial or orofacial apraxia etc. In this project we are concerned in limb apraxia and especially in Ideomotor apraxia. The patients who suffer from this specific form of apraxia have deficits in their ability to plan or complete motor actions that rely on semantic memory. They are able to explain how to perform an action, but unable to "imagine" or act out a movement. The ability to perform an action automatically when cued, however, remains intact. This is known as automatic-voluntary dissociation. For example, they may not be able to pick up a phone when asked to do so, but can perform the action without thinking when the phone rings.

During the rehabilitation process, patients were asked to mimic gestures which were shown to them by a demonstrator. The patients were then scored according to several standard criteria. But this process has vast disadvantages. The fact that the final score is determined by a person, means that it is highly subjective. It is also imprecise because each human is not able to analyse in great detail every possible parameter that is taken under consideration, for example the exact angle of each finger. Finally, there is no way for data (angles, distances etc.) of each gesture of the patient to be saved for later use.

In this project we will try to tackle this problem. We will create an objective and consistent way of scoring the patient's gestures, store his progress and everything will be implemented in a plain and understandable environment.

1.2 Static hand Gestures

This project is concerned with static hand gestures. The gestures that we are going to use were provided by the Department of Experimental Phycology and they are presented below (figures 1.1 – 1.10).

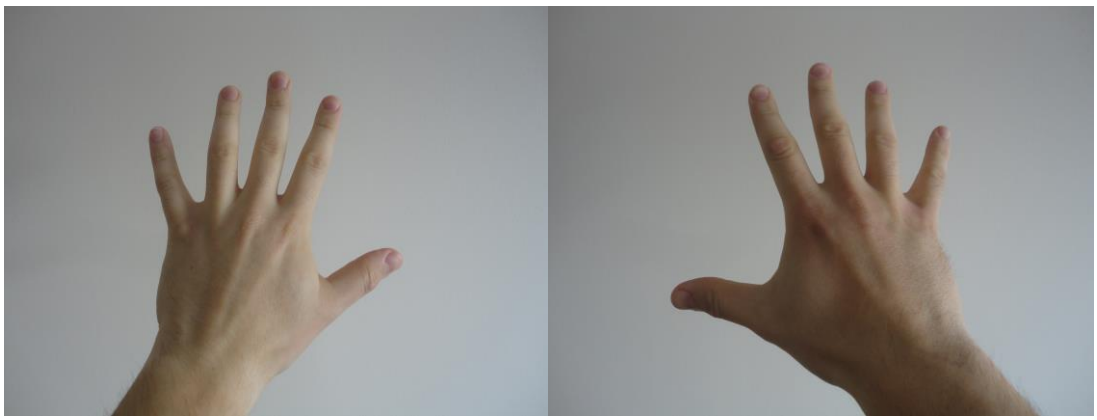


Figure 1.1: Gesture 1 left and right

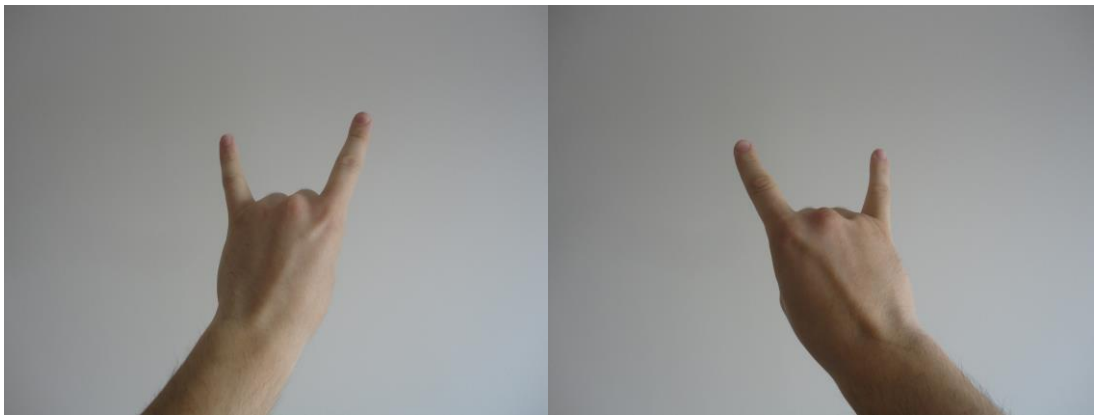


Figure 1.2: Gesture 2 left and right)

Figure 1.3: Gesture 3 left and right

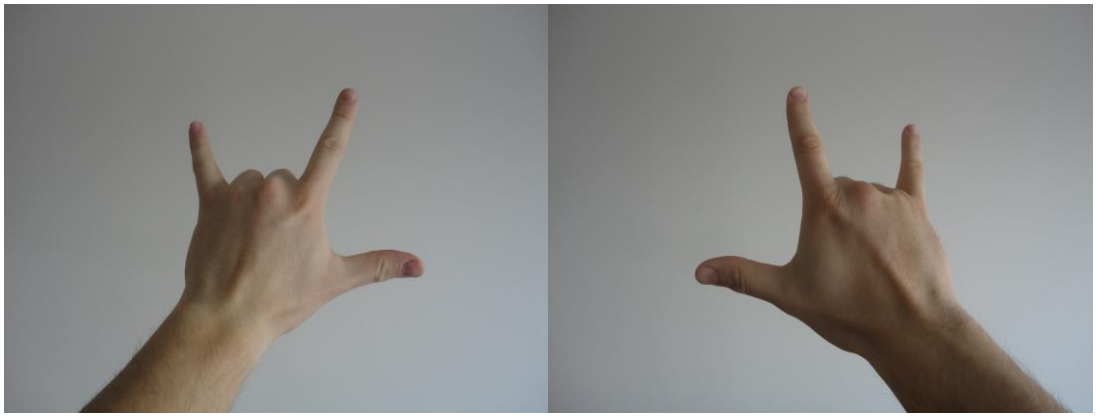


Figure 1.4: Gesture 4 left and right

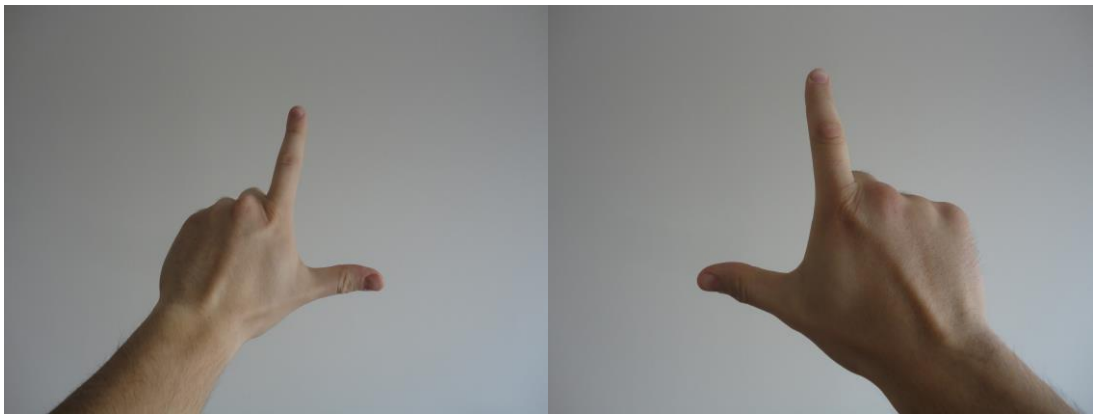


Figure 1.5: Gesture 5 left and right

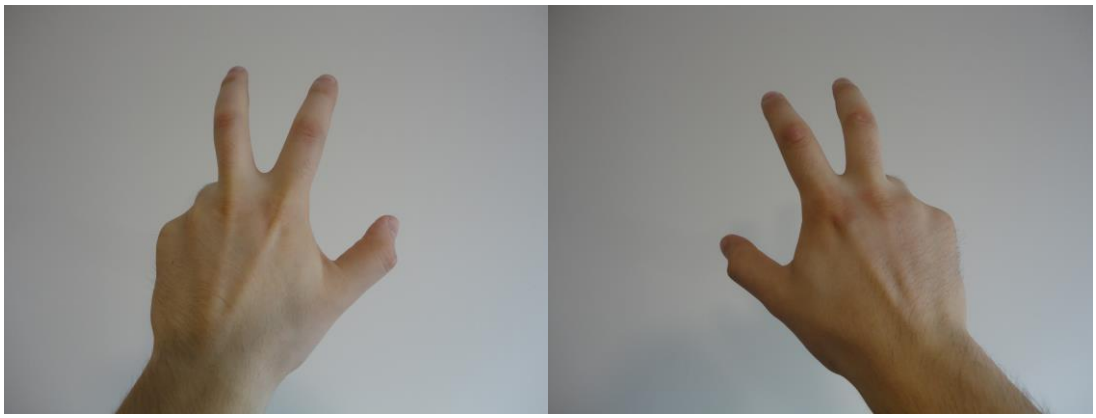


Figure 1.6: Gesture 6 left and right

Figure 1.7: Gesture 7 left and right

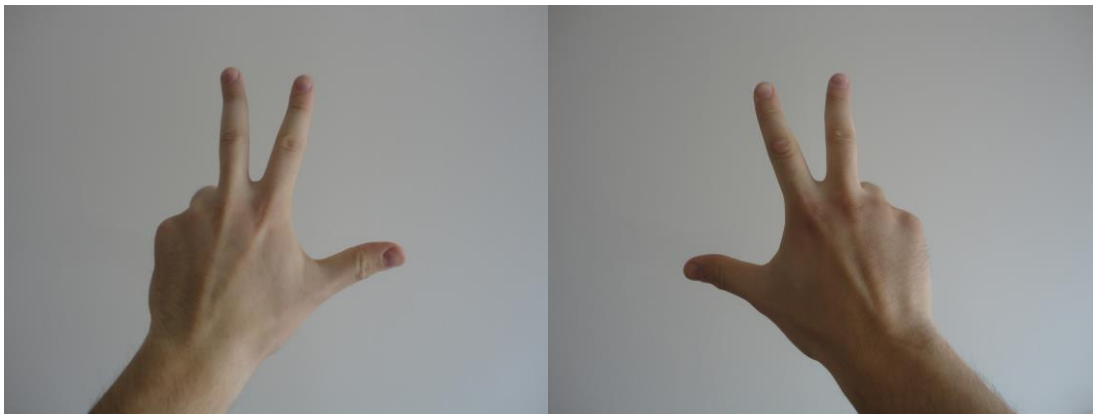


Figure 1.8: Gesture 8 left and right

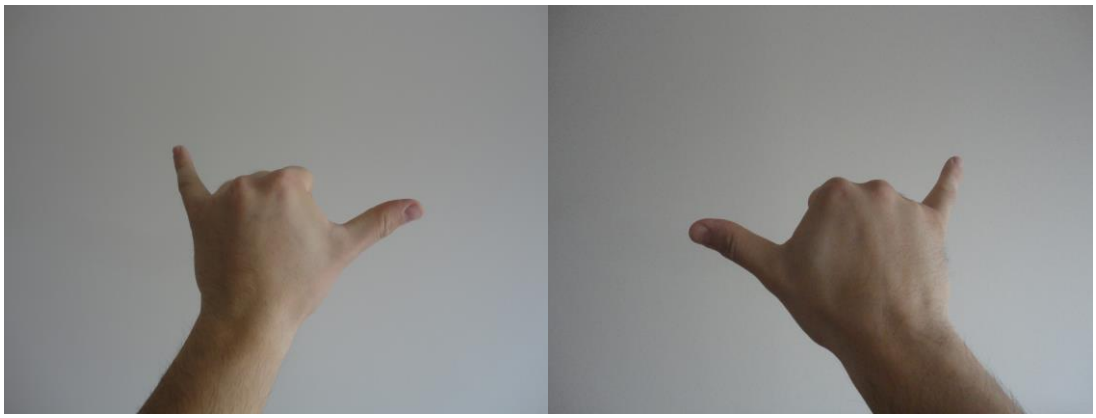


Figure 1.9: Gesture 9 left and right

Figure 1.10: Gesture 10 left and right

1.3 Outline

The main requirements that our program will have to meet are the following:

• A library of gestures will have to be put together. We will work with ten left-hand gestures and the respective right-hand ones. So twenty photographs will be taken and each gesture will be recorded using Leap Motion in order to be used later in the program. These gestures will be called *goal gestures*.

• A scoring function will have to be developed. In our case we will describe two different scoring functions, each one will be totally independent from the other.

• The user will be able to see his score at any time. So, he will be able to improve it before he stops trying.

• Apart from the score, there has to be some kind of visual aid to indicate what is wrong with the user's gesture. Red, yellow and green colours will be used to indicate how well the gesture is performed.

• The time that the patient needed to achieve his high score, as well as the total time that he worked on a gesture will be recorded.

• Each high score and recorded time will be stored in files after the user completes his exercise.

• Data will have to be gathered from healthy users in order to make sure that our program and more specifically our scoring functions work properly. If we notice any disadvantage in our scoring functions we will adjust them to work better.

• The program will have to be used from both patients and healthy controls. For that reason different levels and difficulties will have to exist. The program will have to adjust its functionality accordingly to the user's condition.

• The user should be able to see his progress over time. A visual way of representing his improvement will have to be found.

• All the above will have to be implemented in a user-friendly environment.

1.4 Goals

• The creation of a properly working tool that can be used by patients.

• The design and implementation of sensible scoring criteria.

2. Background

Similar projects have been done by Piotr Kozlowski [14], Kamil Chmurzynski [15] and Emil Culic [16]. Kamil worked with web camera and Piotr used the Microsoft Kinect device which enables humans to interact naturally with computers and provides full-body 3D motion capture, facial recognition and voice recognition. Emil used Leap Motion and he employed two machine learning algorithms which provided a classifier for each target gesture in order to take the score of the user's gestures.

This project follows a different way of implementation. The usage of Leap Motion combined with Unity (both of which are described below), produced a game that covers our requirements. But the most important advancement is the implementation of different scoring functions.

The sections below are a brief introduction to the technologies used.

2.1 Leap Motion

Leap Motion is a small portable device that connects with the computer through a USB port and it is used for tracking the hand movements. Using two monochromatic IR cameras and three infrared LEDs, the device observes a practically hemispherical area, to a distance of about 1 metre (3.28084 feet). The LEDs generate pattern-less IR light and the cameras generate approximately 300 frames per second of reflected data, which is then sent through a USB cable to the host computer, where it is analysed by the Leap Motion controller software, synthesizing 3D position data by comparing the 2D frames generated by the two cameras. [1]

The user places his hands above the machine and the Leap recognizes his palms, wrists, fingers, bones etc. Thus, he is able to interact with his computer just by moving his hands inside the observation area of the machine. The following picture was taken while the user tested the game that was created:

Figure 2.1: User testing the program

The device comes with a variety of apps:

Figures 2.2 and 2.3: Leap Motion apps

There are more that can be downloaded from Leap Motion's app store called Airspace where apps made by developers are being sold.

Version 2.0 of the Leap Motion API introduces a new skeletal tracking model that provides additional information about hands and fingers and also improves overall hand tracking data.

A range of programing languages can be used to develop applications that use Leap Motion: Javascript, C#, Java, C++, Python and Objective-C. Leap Motion also cooperates with game engines like Unity [3] and Unreal [17]. There exists a detailed documentation for each language in the Leap Motion website [6].

The Leap Motion Unity assets[2] provide an easy way to get motion-controlled hands into a Unity game. The Leap Motion Unreal plugin provides an easy, blueprint-friendly way to add motion-controlled 3D hands to an Unreal game. The use of these environments provides the developer with the opportunity to use the benefits of both Leap Motion and Unity (or Unreal) to create a fascinating game.

2.1.1 API Overview

The Leap Motion system employs a right-handed Cartesian coordinate system (figure 2.4). The origin is centred at the top of the Leap Motion Controller. The x- and z-axes lie in the horizontal plane, with the x-axis running in parallel to the long edge of the device. The y-axis is vertical, with positive values increasing upwards (in contrast to the downward orientation of most computer graphics coordinate systems). The z-axis has positive values increasing toward the user.

[2] We will see what an asset is in section 2.2.3

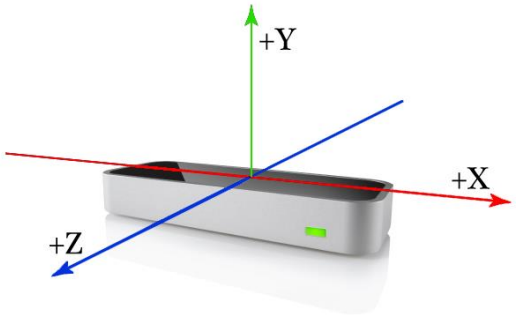


Figure 2.4: Leap Motion coordinate system.

As the Leap Motion controller tracks hands, fingers, and tools in its field of view, it provides updates as a set – or frame – of data. Each **Frame** object representing a frame contains lists of tracked entities, such as hands, fingers, and tools, as well as recognised gestures and factors describing the overall motion in the scene. The Frame object is essentially the root of the Leap Motion data model.

The hand model provides information about the identity, position, and other characteristics of a detected hand, the arm to which the hand is attached, and lists of the fingers associated with the hand. Hands are represented by the **Hand** class.

An **Arm** is a bone-like object that provides the orientation, length, width, and end points of an arm. When the elbow is not in view, the Leap Motion controller estimates its position based on past observations as well as typical human proportion.

The Leap Motion controller provides information about each finger on a hand. If all or part of a finger is not visible, the finger characteristics are estimated based on recent observations and the anatomical model of the hand. Fingers are identified by type name, i.e. *thumb*, *index*, *middle*, *ring*, and *pinky*. Fingers are represented by the **Finger** class.

Here is an example of how Leap Motion can be used to receive the position of our palm in C# :

```
1  Controller controller = new Controller();
2  // add a Controller object to the program - which serves as our
3  // connection to the Leap Motion service/daemon.
4  Frame frame = controller.Frame ();
5  // All the tracking data in the Leap Motion system arrives through
6  // the Frame object.
7  HandList hands = frame.Hands;
8  // Get the hands of the user
9  firstHand = hands.Leftmost;
10 // Get the leftmost hand
11 Vector v = firstHand.PalmPosition;
12 // Get the position of the palm of the leftmost hand
```

We can gain information on the position of our fingertips, the position of the joints, the length of each bone or the angle between bones in a similar way. All this information is vital in creating our final program.

Leap Motion can also identify tools if the object is simple and by that we mean they have to be thin and cylindrical, like a pencil. Tools are represented by the **Tool** class.

Figure 2.5: Tool.

Leap motion handles distances in millimetres, time in microseconds and angles in radians.

2.2 Unity

Unity [3] is a cross-platform game engine developed by Unity Technologies and used to develop video games for PC, consoles, mobile devices and websites. Through Unity, the developer can create multiplatform 3D and 2D games and interactive experiences. Using it is easy enough for a beginner and powerful enough for expert use.

There are two versions available: Unity Pro and Unity Personal [2]. For this project version 5 of Unity Personal was used.

In the official website of Unity [3], a number of ways are offered for someone to familiarize himself with the platform. There are video and article based contents, there is also Documentation which is a complete written manual and scripting reference, and finally there are live Q&A sessions.

Unity supports C#, Javascript and Boo. C# was used for this project but despite the fact that C# is more advanced, optimized and robust than JS and Boo, C# does have its disadvantages. Perhaps the most notable drawback is that most Unity tutorials are written for Javascript.

There are numerous games created using Unity and Leap Motion [4]. Figure 2.6 shows our game during execution:

Figure 2.6: Our game during execution

2.2.1 Why Unity?

Apart from the fact that Unity is one of the greatest game development engines, Unity was a successful choice because it has the necessary functionality for reaching all the goals that we have set, for example the 3D environment that it offers matches perfectly with our needs. Unity allows us to export easily our project on PC, Mac & Linux standalone file, which means that it is runnable to every computer without having installed the platform itself. We can use it for free and it provides us with countless options, abilities and most importantly, it allows easy extensions for further future work.

2.2.2 Unity Panels

The project panel is where all the assets within a project are stored. When assets are imported, they will first appear here. The hierarchy panel is where assets are organized in a scene. Assets from the Project panel can be dragged into the Hierarchy panel to add them to the current scene. The Inspector panel lets the developer inspect and adjust all the attributes of a selected asset. Everything from its position and rotation, to whether it's affected by gravity or able to cast a shadow.

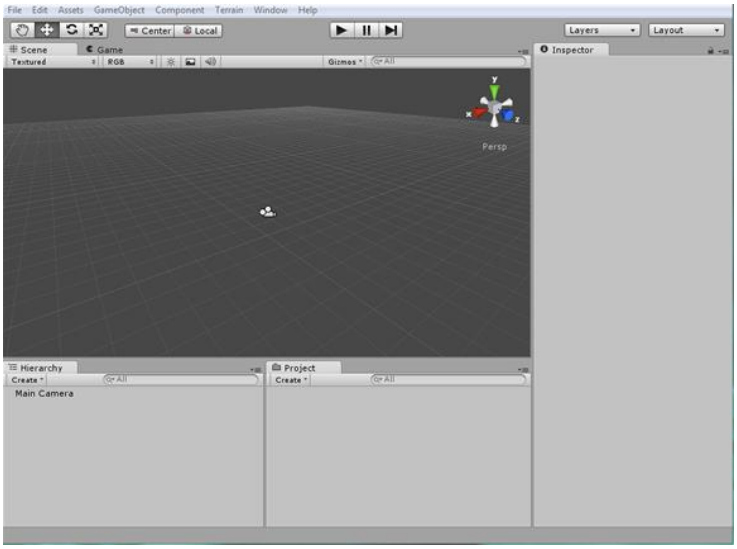


Figure 2.7: Unity panels.

2.2.3 Unity Assets

Assets are any resource that a game uses. This includes 3D models, materials, textures, audio, scripts, and fonts and numerous others. Apart from some simple objects such as cubes and spheres (spheres will be used extensively in this project), Unity can't actually create most of these assets. Instead, they must be created externally using 3D modelling applications and painting tools and then imported into Unity. Leap Motion offers an asset package that contains assets such as model hands or others. In order to create a Leap Motion – Unity game we will have to use the Leap Motion asset package but we will explain this more detail in Section 4.

It is of great importance and convenience that Unity's asset importing is robust and intelligent. Unity, in contrast with most 3D game engines, will accept all popular 3D file formats including Maya, 3D Studio Max, Blender and FilmBox with all the rigging, materials and textures intact. Unity also supports all common image file formats, including PNG, JPEG, TIFF and even layered PSD files directly from Photoshop. Unity supports audio files such as WAV and AIF, which are ideal for sound effects, and MP3 and OGG for music. For this project, JPEG images were used, although, the game created could be extended to contain sound effects.

Unity has an Asset Store where the developer can purchase or in some cases download for free, 3D models, characters, textures, sound effects, music, tools, and even scripts.

2.2.4 Unity Scenes and Functions

Scenes contain the objects of the game. They can be used to create a main menu, individual levels, and anything else. In each Scene, environments, obstacles, and decorations can be placed. Essentially a scene is a piece of the game that has to be designed and built. Scenes are where the developer can drag in project assets and arrange them to make levels and game screens. The Hierarchy panel represents the contents of the current scene in a tree-like format.

Scripting is an essential ingredient in all games. Even the simplest game will need scripts to respond to input from the player and arrange for events in the gameplay to happen when they should. Beyond that, scripts can be used to create graphical effects, control the physical behaviour of objects or even implement a custom AI system for characters in the game.

Scripts are known in Unity as behaviours, let you take assets in your scene and make them interactive. Multiple scripts can be attached to a single object, allowing for easy code reuse. They can be used to manipulate objects that exist in the scene, such as cameras.

In this project the camera's zooming, the movement of the user's hand, the buttons that exist on the screen as well as every calculation that takes place, are all coded in C#.

A typical C# script will look like this:

```
1  using UnityEngine;
2  using System.Collections;
3  public class MainPlayer : MonoBehaviour {
4      // Use this for initialization
5      void Start () {
6
7      }
8      // Update is called once per frame
9      void Update (){
10
11     }
12 }
```

The `Update` (line 9) function is the place to put code that will handle the frame update for the GameObject.

The `Start` (line 5) function will be called by Unity before gameplay begins (ie, before the Update function is called for the first time) and is an ideal place to do any initialization.

Another important function in Unity is: `void OnGUI ()`

`OnGUI` is called for rendering and handling GUI events. It is responsible for any buttons or messages that appear on the screen.

2.3 Unity and Leap Motion

The Leap Motion asset package includes plugin files for using the Leap Motion device on Windows and Mac computers. This package includes Hand prefabs, scripts, and demo scenes to help the programmer develop Leap Motion apps quickly.

In addition to the prefabs and scripts included in the asset package, the user can write his own scripts to access tracking data from the Leap Motion API. The Leap Motion classes are defined in the Leap namespace. A basic *MonoBehavior* class that accesses the Leap Motion API will look something like the following:

```
1  using UnityEngine;
2  using System.Collections;
3  using Leap;
4
5  public class MainPlayer : MonoBehaviour {
6
7      Controller controller;
8
9      void Start () {
10         controller = new Controller();
11     }
12
13     void Update (){
14         Frame frame = controller.Frame();
15         // use the tracking data that Leap offers
16     }
17 }
```

Since a Unity application has a natural frame rate, the `Update()` function can get the current frame of data from the Leap Motion controller when the function is called by the Unity engine.

3. The Hand Model In Unity

3.1 Human Hand Through Leap Motion

The next image shows the finger bones in the human hand. It also shows the position of the palm and the wrist. Figure 3.1 shows the Leap Motion inner representation of the human hand.

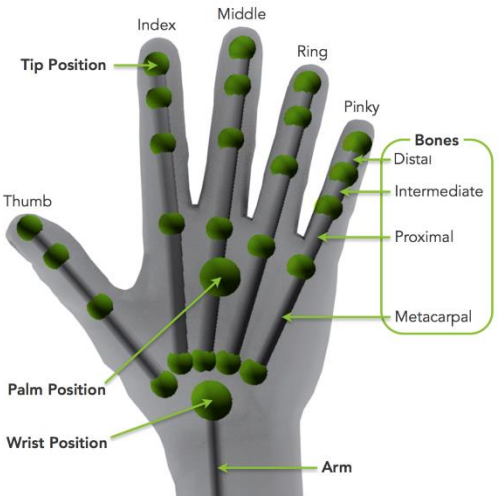


Figure 3.1: The Leap Motion representation of the human hand

If we have initialised the controller and the frame as mentioned in the Unity section, we can use the following commands to gain information about the user's hand:

```
1  // The palm position as a vector
2  Vector v = firstHand.PalmPosition;
3  // take the arm
4  Arm arm = firstHand.Arm;
5  // Print the first coordinate of the vector that contains the
6  position of the wrist
7  print (arm.WristPosition.x);
8  // And now for the bones:
9  // take the fingers
10  FingerList fingers = firstHand.Fingers;
11  // for each finger
12  foreach(Finger finger in fingers){
13    Bone bone;
14     // for each bone
15    foreach (Bone.BoneType boneType in
16  (Bone.BoneType[]) Enum.GetValues(Typeof(Bone.BoneType))){
17          // store bone
18          Bone = finger.Bone(boneType);
19          // ... use bone ...
20    }
21 }
```

Bones are ordered from base to tip, indexed from 0 to 3. Additionally, the bone's Type enum may be used to index a specific bone anatomically. The thumb does not have a base metacarpal bone and therefore contains a valid, zero length bone at that location.

3.2 Representing The User's Hand

In order to use Leap Motion through Unity, we have to download and install the latest asset package from Leap Motion website. After we create our project and we import the package, we still have to drag in our scene the prefab called *HandController*. Then we are ready to create our game.

The asset package contains numerous interesting things such as *materials*, *shaders* or *textures* that we could use in our game. It also contains several hand models, such as the ones represented in figures 3.2 and 3.3:

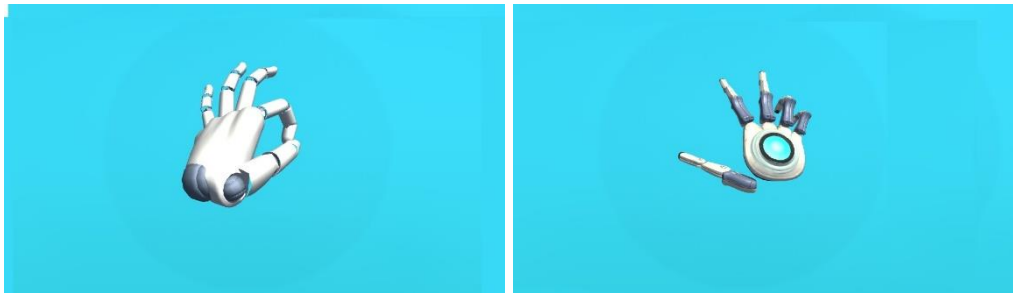


Figure 3.2 and 3.3: Hand models CleanRobotLeftHand and GlowRobotRightHand

It would be possible to use one of the model hands that are provided, but there are some problems that occur if these hands are used. A major problem is that the user cannot see his fingers if they are behind the palm. So, it would be simpler for the user and maybe more instructive if there weren't any unnecessary hand parts in the scene, just whatever is important to make the gesture correctly. And the chosen way to do that is the following:

Twenty-two spheres at each frame are created and they are placed at the position of the last joint of each bone (`bone.NextJoint`). So, there is one sphere representing the wrist, one for the palm and 4 for each finger (figure 3.4). Note that the joint at the beginning of each finger, except for the thumb, does not appear on the screen. The reason for that is because they wouldn't be useful. These joints positions depend solely on the position of the wrist. The user wouldn't gain any additional information by seeing these four spheres. On the other hand, the fact that the beginning (which is also the end) of the metacarpal bone of the thumb appears on the screen is more useful because it is easier for the user to see how his palm is rotated.

Figure 3.4: Our representation of the human hand.

By representing the user's hand this way, the user is able to see exactly where each of his finger joins are, which is the most important part, and it also helps the way that its scoring and visualization take place.

Spheres were chosen because they are simple geometrical objects and they are sufficient to represent the core information of the human palm. Spheres are just the underlying model. Other graphic representations would be easy to display.

After representing the user's hand, we will have to figure out a way to represent the *goal gesture* as well. This will be the steady gesture that the user sees on the screen and tries to mimic it. In order to be able to represent the goal gestures, we first have to record them.

3.3 Recording The Gestures/ The Goal Gesture

Because this project is concerned only with static hand gestures, it is sufficient to record the positions of each joint of each gesture on a single frame. In order to record the gestures, I had my hand above Leap Motion and I performed the gesture that I wanted to record. The essential information were stored on a file that was specified by this command:

```
using (System.IO.StreamWriter file = new
      System.IO.StreamWriter(@"C:pathname\filename.txt", true)) { … }
```

The palm position, the wrist position as the position of the beginning and the end of the metacarpal bones, and finally the position of the last joint of the proximal, intermediate and distal bones were stored in that file. The following commands appends to the `file` the palm position:

```
file.WriteLine(""+ firstHand.PalmPosition);
```

Note that the end of the metacarpal bones is the same as the beginning of the proximal bones, so after saving the coordinates mentioned above, we will have a file containing the coordinates of the beginning and the ends of every bone in every finger.

So, after having completed the above procedure, we have a file named *filename.txt* in the *pathname* directory, which contains the positions of each joint that we are interested in. Despite the fact that we will not be showing where the start of the metacarpal bones is, we will need their positions for later use.

We now have to implement a way to use these data. If we read each gesture from a txt file during execution, we always run the danger that the user might accidentally alter some of them. If that happened then the program would not run properly. So, in order to avoid this disadvantage, we have to store the coordinates in the code. Then whenever the program needed them, it would call a function and load the information.

In order to represent the goal gesture, we will follow a similar procedure that we followed for the user's hand. Twenty-two additional spheres were created through Unity, and each one was placed where the recorded information indicated. When the user chooses the gesture he wants to perform, in the *gesture and scoring function menu*, the respective coordinates are loaded, so he is able to see the goal gesture on the screen.

What we will do next is explain how the scoring takes place. Giving a score to the user is a challenging task. We will explain the criteria that were taken under consideration and possible improvements that may give a more accurate score.

4. Scoring Functions

Two ways of scoring were examined in this project; first one is based on angles, and the second on distances. The user is able to choose which way he prefers to be scored with and he is able to see all his scores gathered once he has finished. Each way is independent and has different advantages and disadvantages. We will first examine angle scoring function.

4.1 Angle Scoring

The user has done the gesture correctly if the angles between his finger bones match with the respective angles between the bones of the goal gesture. If the goal and the user finger angles are the same, then the user receives a perfect score. Otherwise, his score is calculated depending on the difference between the two angles.

4.1.1 Calculating The Angles

We will treat each bone as a vector. An array will contain these vectors. The information that we have in our disposal is the coordinates of the beginning and the end of each bone.

Let A(x_1, y_1, z_1) and B(x_2, y_2, z_2) be the points in space that represent the beginning and the end of a single bone. The vector that connects A and B is this:

$$\overrightarrow{AB} = (x_2 - x_1, y_2 - y_1, z_2 - z_1)$$

We will create an array that contains these differences for the user's hand and one for the goal gesture, now we will use these vectors to find the angle between them. There is a built-in command that allows us to compute this angle in radians:

```
vector1.angleTo(vector2)
```

In this project, we worked with degrees, so we had to convert all angles.

We are now ready to continue with the scoring function but to do so, we will have to keep in mind firstly, that the user has chosen a level of difficulty and secondly, the difficulty that he has moving each of his fingers at the beginning of the game.

4.1.2 The Scoring Function

The main part of this scoring function is presented below.

```
1  // for each finger
2  for (int i = 0; i<fingerDifficultyLeft.Length; i++) {
3     numerator1 += fingerDifficultyLeft [i] *
4  (100 - (Math.Abs (GoalBonesAngles [i, 0] - UserBonesAngles [i, 0])));
5     numerator2 += fingerDifficultyLeft [i] *
6  (135 - (Math.Abs (GoalBonesAngles [i, 1] - UserBonesAngles [i, 1])));
7     numerator3 += fingerDifficultyLeft [i] *
8  (90 - (Math.Abs (GoalBonesAngles [i, 2] - UserBonesAngles [i, 2])));
9  }
10
11 for (int i = 0; i < fingerDifficultyLeft.Length; i++) {
12    denumerator += fingerDifficultyLeft [i];
13 }
```

The variables `denumerator`, `numerator1`, `2` and `3` are initialized to 0. The array `fingerDifficultyLeft` contains the integers that the user chose at the beginning for his difficulties for his left hand (we work similarly if the gesture is a right hand one)[3]. These integers are multiplied by a difference. The subtraction that takes place in lines 4, 6 and 8, is between a constant number and another difference. The second difference is the one that we mentioned in the previous section. So, the smaller the difference between the user's angles and the goal angles, the bigger the numerator.

By convention we have called:
The angle between the metacarpal bone and the proximal bone: *Angle1*
The angle between the proximal bone and the intermediate bone: *Angle2*
The angle between the intermediate bone and the distal bone: *Angle3*

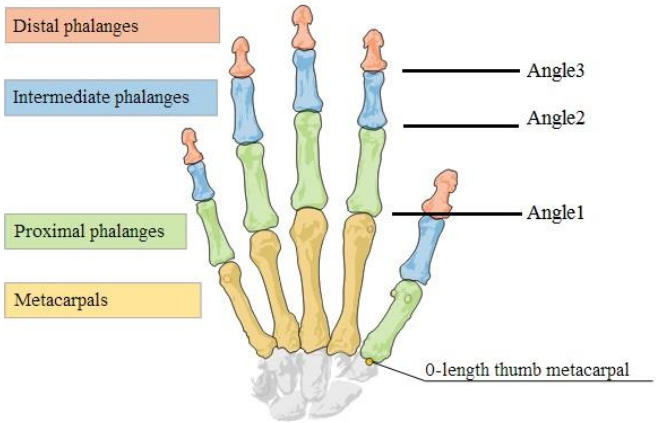


Figure 4.1: Angles between the finger bones.

Note that *Angle1* of the thumb will always be 0.

The constant values 100, 135 and 90 play an important role in the final score. The bigger these values are, the bigger the final score.

The first constant, as well as the whole variable `numerator1` (lines 3, 4), are addressed to *Angle1*. The second is addressed to *Angle2* (lines 5, 6) and the last to *Angle3* (lines 7, 8). These specific values were chosen because the final score should depend less on *Angle3*, because the way this part of the finger bends, is more dependent on the way the previous joint is bent. So, the fact that the user managed to bend his second joint correctly should be rewarded more than the third joint which follows the second. *Angle1* receives a higher constant than *Angle3* for the same reason. The fact that the constants of *Angle1* and *Angle2* are different is because of empirical reasons. Tests show that having a bigger constant in *Angle2* provides a bigger score than having a big constant in *Angle1*.

But apart from the reasons described above, these specific numbers make sense because *Angle2* can bend approximately 135 degrees maximum and *Angle3* about 90 degrees maximum. *Angle1* can bend more than 100 degrees, but giving a bigger

[3] The finger difficulties will be examined in detail in section 5.2

constant there would give a much larger score than the one required. The number 100 is a balanced number between the maximum bending of the angle and the score difficulty that we want to achieve.

But this is obviously not the final score. The final score is produced by the following formula:

```
1  angleScore [gesture - 1] =
2        100*((Convert.ToSingle(numerator1 / denumerator)/100) *
3        (Convert.ToSingle(numerator2 / denumerator)/135) *
4        (Convert.ToSingle(numerator3 / denumerator)/90));
```

The result that we get from this formula is a percentage. The term `numerator1 / denumerator` (line 2) is 100 if *Angle1* of the user and *Angle1* of the goal gesture are the same. So the factor:

```
(Convert.ToSingle(numerator1 / denumerator)/100)
```

Will be equal to 1 if the user has bent this angle perfectly. The same behaviour have the rest of the terms. So the outcome will be a number between 0 and 1.

The function `Convert.ToSingle` (lines 2, 3, 4) converts a number to float. The outcome of the formula `(numerator1 / denumerator)/100` is a double, we will have to convert it because the array `angleScore` is defined as float so that there are not many unnecessary decimals in the outcome.

After gathering data from healthy controls, it emerged from the data that the maximum score achieved by most of them was around 95 - 97%.

Most people achieved scores close to 90% in most gestures and close to 80-85% in the more difficult ones.

An additional observation was that the worst possible score that anyone could achieve was close to 13%. Fitting the possible scores in the interval [0,100] was inevitable.

4.1.3 “Stretching” The Score

The following code “stretches” the score in a way where 97% becomes 100% and 13% becomes 0%.

```
1  // if the score is less than 13, set it to 13
2  if (angleScore < 13) {
3        angleScore = 13;
4  }
5  // stretch the score
6  angleScore = Convert.ToSingle(2 / Math.PI *
7        Math.Atan((angleScore - 13) / 100) * 225);
8  // if the produced score is above 100, set it to 100
9  if (angleScore > 100) {
10       angleScore = 100;
11 }
```

The table below illustrates some of the changes that occur.

Before	After
12%	0%
14%	1.42%
20%	10.01%
30%	24.12%
40%	37.77%
50%	50.76%
60%	62.93%
70%	74.20%
80%	84.55%
90%	93.99%
95%	98.37%
97%	100%

Table 4.1: Before and after the Stretching function.

With a quick look we can see that scores below 50% fall and above 50% rise. That is considered a good behaviour because 50% is the ideal threshold. A bad behaviour would be if the user had achieved a score of 65% and his new score was 60%, or even worse if the score was higher.

The graph below is an optical representation of the above table.

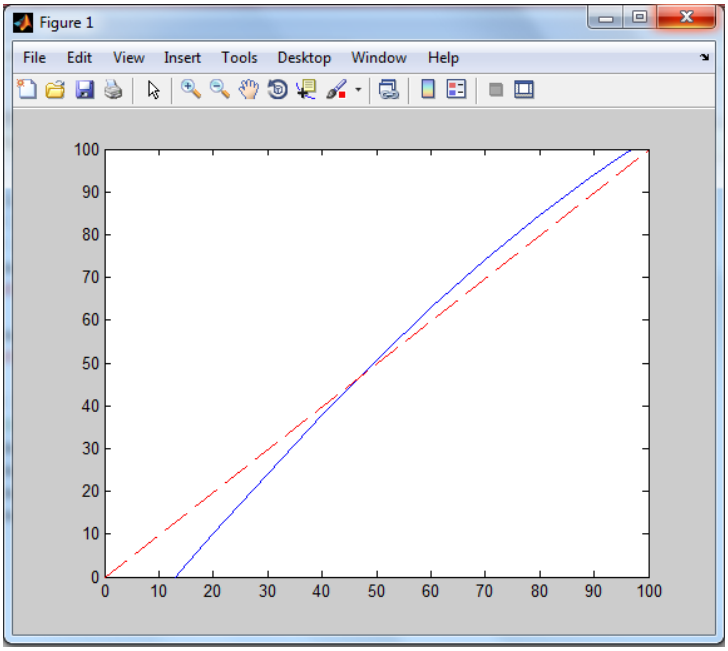


Figure 4.2: Optical representation of table 4.1

The red-dashed line is the representation of the function f(x) = x. The blue line is the function that takes as input the numbers on the left column of the table and returns the respective numbers on the right one.

It is clear that our new score will be smaller for inputs less than 50% and larger for the rest. We can also notice that the amount of increase is larger for inputs close to 70 or 80%. When our score goes near 90%, the amount of increase is smaller.

The main element of the stretching function is the *arctan* function. Its domain is the real numbers, it takes values from $-\frac{\pi}{2}$ up to $\frac{\pi}{2}$ and it is an increasing function. If we divide it by $\frac{\pi}{2}$ our new function will get values between –1 and +1 (line 6). The fact that we are only concerned about positive inputs, means that our function will give us values from 0 to +1. Now if we subtract 13 from our input (line 7), it will mean that if our input is 13 our function will give 0 as a result. That is why the if-statement is necessary before the definition of the function (lines 2 – 4); it will convert every value below 13 to 13, so that our function will give 0.

The other constants are found in an empirical way. We wanted 97% to be the "new" 100% so we had to try different constants to find a threshold.

This concluded the way the angle scoring takes place.

4.1.4 Colouring the Spheres

We could assume that giving a score to the user would be enough. But people respond more effectively to visual feedback. For that reason, we will give colours to the user's spheres. It is as simple as it sounds, but the outcome is much elegant than without colours. There are two ways to achieve that.

Firstly, the spheres that represent the wrist, the palm and the ends of the metacarpal bones will always be coloured green and the reason for that is because the angles between these joints are always the same. The rest of the spheres will be coloured depending on the angle that they represent. The end of the proximal bones represent *Angle1*, the end of the intermediate bones represent *Angle2* and the end of the distal bones represent *Angle3*. If the difference between *Angle1* of the user and *Angle1* of the goal gesture is bigger than 15 degrees then the respective sphere will be coloured red, if the difference is between 10 and 15 degrees, the sphere will be coloured yellow and if the difference is smaller than 10 degrees, the sphere will be coloured green (figure 4.3).

These thresholds were once more chosen empirically. Their purpose is to show the users what they are not doing correctly so that they can improve it before they end the exercise. A fragment of code that colours a sphere green is this:

```
1 if (( Math.Abs(GoalAngle - UserAngle) < 10)) {
2     userSphereRend.material.color = Color.green;
3 }
```

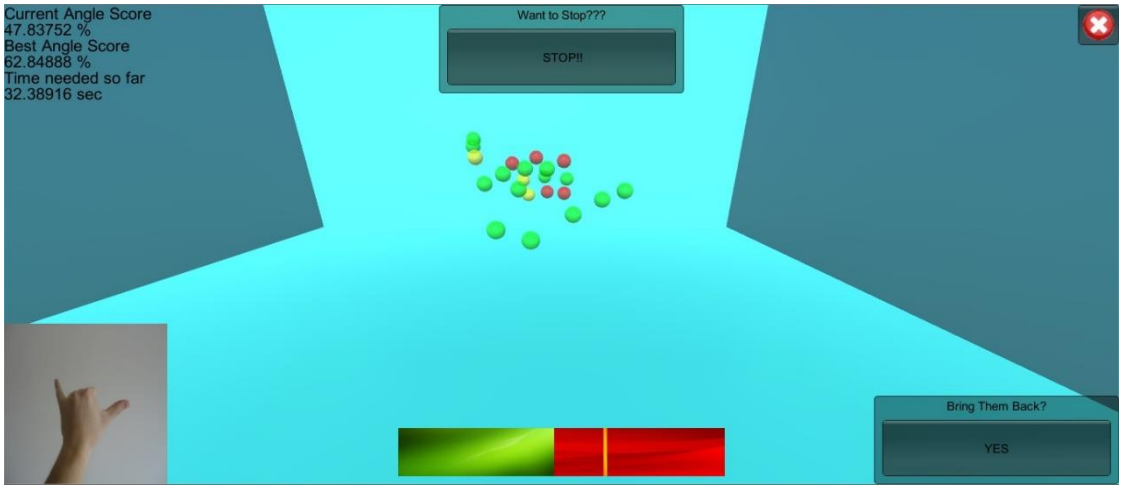


Figure 4.3: First way of colouring the user's spheres during angle scoring.

A Second way, and maybe a more precise is the following: We will use a function that is provided to us by Unity.

```
Color.Lerp(Color a, Color b, float t);
```

With the use of this function we can interpolate between colours `a` and `b` by `t`, `t` is clamped between 0 and 1. When `t` is 0 returns `a`. When `t` is 1 returns `b`. But this way is not perfect either. When the user comes very close to his target, it is really hard to tell the difference between shades of green, so it is really hard to tell what is wrong with his gesture. But this disadvantage exists in the previous way as well, so it is safe to say the second way has more advantages than the first one, it is more precise and more clear to the user. This way was the one that was chosen for the final version of this game.

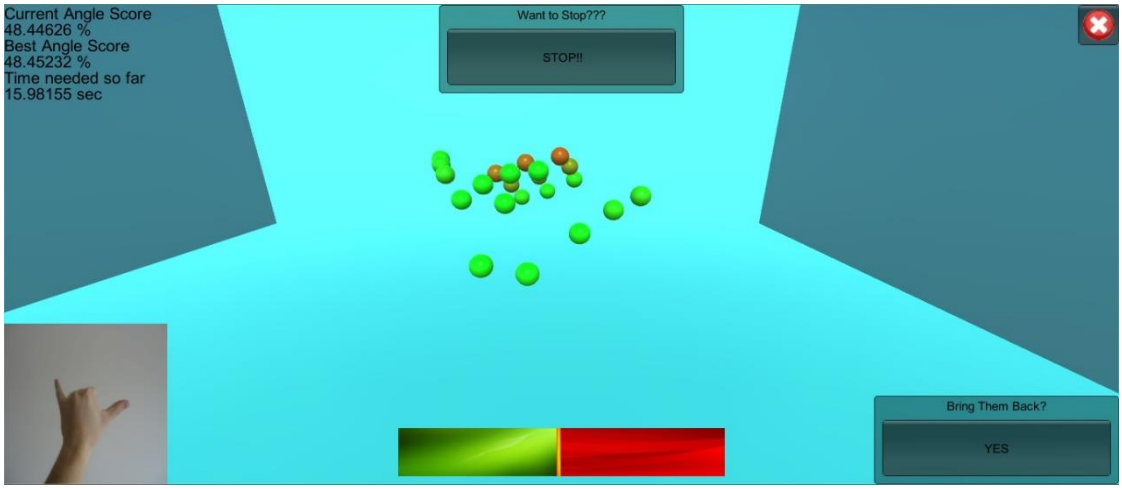


Figure 4.4: Second way of colouring the user's spheres during angle scoring.

4.2 Position Scoring

The Position Scoring function is solely based on the distance between the user and the goal spheres.

4.2.1 Calculating The Distances

At every frame, the distance between each one of the user's sphere with its respective goal gesture sphere is calculated. So twenty-two distances will be calculated.

```
1  void DistanceUserAndGoalSphere(){
2  // for each sphere
3     for (int I = 0; i<numberOfSpheres; i++){
4           // the distance between the user and the goal sphere
5           distFingers[i] = (userFingers[i].transform.position -
6           goalFingers[i].transform.position).magnitude;
7     }
8  }
```

The array `userFingers` is defined as:

```
public GameObject[] userFingers = new GameObject[numberOfSpheres];
```

GameObject is Base class for all entities in Unity scenes. When we add the line above to our script, an empty array will be created in Unity editor, and we will have to fill it with spheres. So by the command:

```
userFingers[i].transform.position
```

We are able to access the position of this object in Unity. The above command will return a Vector3 which comes with the built-in command `.magnitude`, which will return the magnitude of the vector, in other words the Euclidian Distance between its endpoints.

So if the tip of the user's index is located at the coordinates (x,y,z) and the tip of the index of the goal gesture is located at the coordinates (a,b,c) then the above function will calculate:

$$\sqrt{(x-a)^2+(y-b)^2+(z-c)^2}$$

4.2.2 The Function

Thus, when these distances are created, the user receives a score depending on whether or not his spheres collide with the respective goal ones. Let us consider the distance between the sphere that represent the user's palm and the sphere that represent the goal palm.

i) If that distance is less than the radius of the spheres then the user takes the maximum score for that sphere and the sphere is coloured green.

ii) If it is between the radius and $\frac{4}{3}$ of the radius, then the user receives half score for that sphere and the sphere is coloured yellow.

iii) If the distance is more than $\frac{4}{3}$ of the radius the user takes zero score for that sphere and the sphere is coloured red.

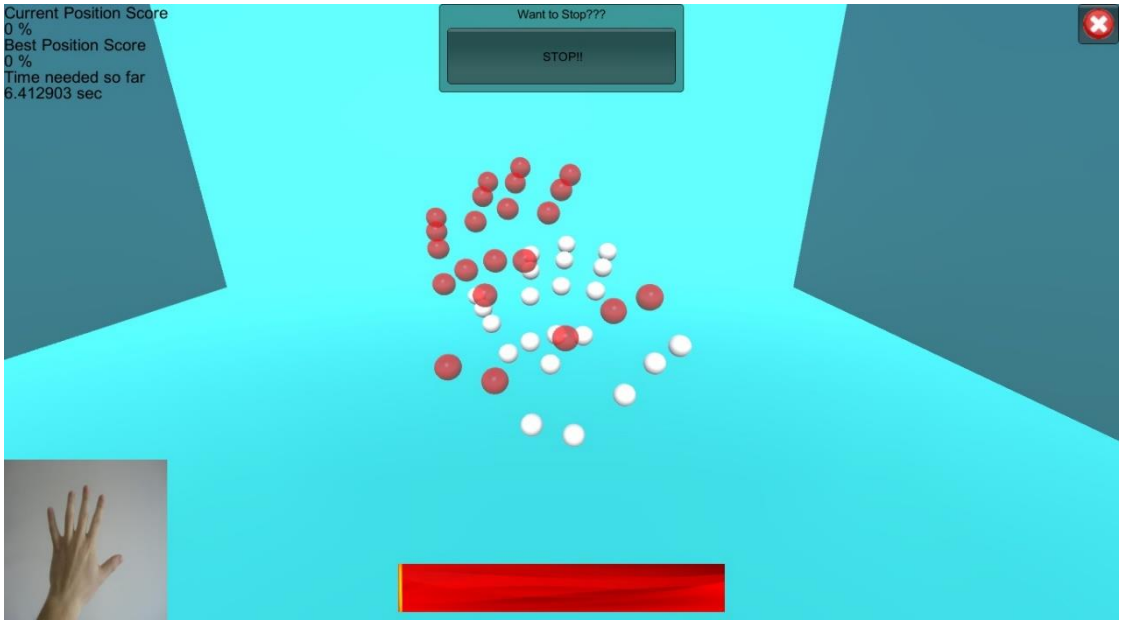


Figure 4.5: All distances are more than $\frac{4}{3}$ of the radius

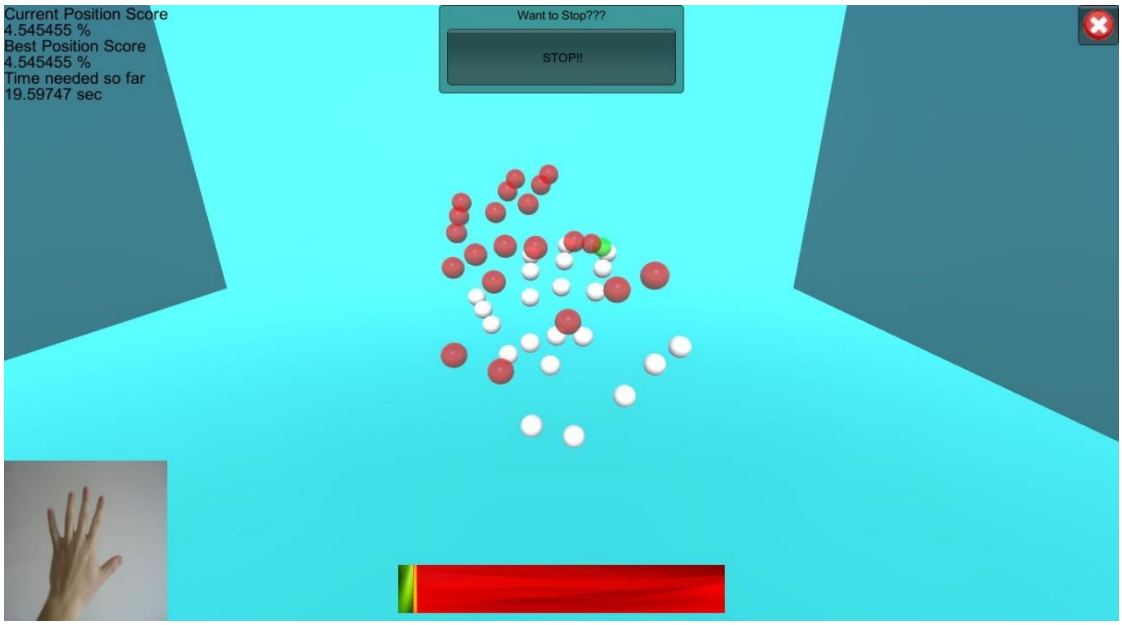


Figure 4.6: Only one distance is less than the radius of the spheres

Figure 4.7: Some of the distances are less than the radius of the spheres, some between the radius and $\frac{4}{3}$ of the radius and others more than $\frac{4}{3}$ of the radius.

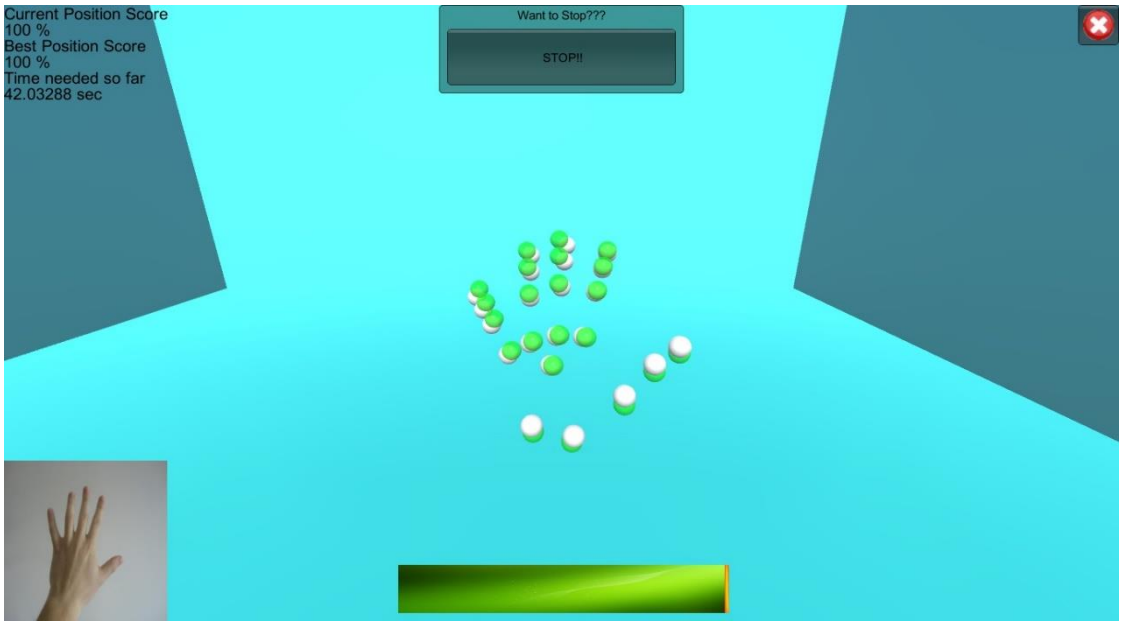


Figure 4.8: All distances are less than the radius of the spheres

The final score that the user sees is produced by the method described below.

An integer variable sum is created which is increased by the following if the above cases occur:

i) `2 * fingerDifficultyLeft [j]`

ii) `1 * fingerDifficultyLeft [j]`

iii) it is not increased.

Here j indicates the finger that is examined. In case we are examining the palm or the wrist, then the method is the same just without the multiplication with the difficulties.

The denominator *totalDiff* is also created and it is increased by two for the palm and 2 for the wrist and then it is increased by `2 * fingerDifficultyLeft [j]` for every finger and joint.

```
1  int totalDiff = 4;
2  // for every joint
3  for (int j = 0; j<4; j++) {
4      // for every finger
5     for (int i = 0; i<fingerDifficultyLeft.Length; i++) {
6           totalDiff += 2 * fingerDifficultyLeft [i];
7     }
8  }
```

The algorithm works similarly if we are working with the right hand.
So the score is:

```
posScore = 100*((sum*1.0f)/totalDiff);
```

This is a percentage and it is possible to achieve both 0% and 100%.

But, the above function needs further adjustment in order to allow different hand sizes. What happens if the user's hand is smaller or bigger than the hand he sees on the screen, in other words my hand? That would mean that the user would never be able to achieve a 100% no because he doesn't perform the gesture correctly but because it is practically impossible.

Note that this matter does not bother us in the angle scoring case, because the angles are not affected by the length of the bones or the size of the palm in general.

The way to make this adjustment is described in the section below.

4.2.3 Adjust Gesture To The User's Hand

The main idea for resolving this issue has been mentioned in section 4.1.1. We will have to treat each bone in the goal gesture as a vector. That is possible because we have the coordinates of the beginning and the end of each bone[4].

Firstly we will read the position of the wrist and put the adjusted wrist sphere in the same position. The position of the wrist will stay the same because it is not affected by the size of the user's palm.

Now we will deal with the position of the palm. The necessary steps that need to be followed are listed below:

• The recorded position of the palm is read.
• We create a *Vector3* with these coordinates as parameters, except the last one has to

[4] This is the reason why we kept the coordinates of the beginnings of the metacarpal bones when we recorded the gestures.

be multiplied by (–1). That is because Unity and Leap Motion calculate the z-coordinates in opposite directions. So if we didn't make this alteration, the goal hand that would appear on the screen would be a reflection of the hand that was supposed to be shown.

• Let *myPalmWristDist* be the distance between the recorded wrist and palm sphere (the above vector is used).

• Let *wristToPalm* be the vector that connects the recorded wrist with the recorded palm, and it has the direction wrist → palm[5].

• Now we will use the user's hand to calculate the distance between his wrist and palm, let this distance be *newPalmWristDist*.

• Now we are ready to find the adjusted position of the palm sphere:

```
1  Vector3 newPalmPosition = new Vector3 ();
2  newPalmPosition.x = Convert.ToSingle(wrist.transform.position.x +
3          (wristToPalm.x / myPalmWristDist) * newPalmWristDist);
```

The piece of code above describes the way to find the first coordinate that we are looking for, which then needs to be rewritten in the other two coordinates.

`wrist` (line 2) is the vector that contains the coordinates of the recorded wrist sphere.

So, let's explain the above formula.

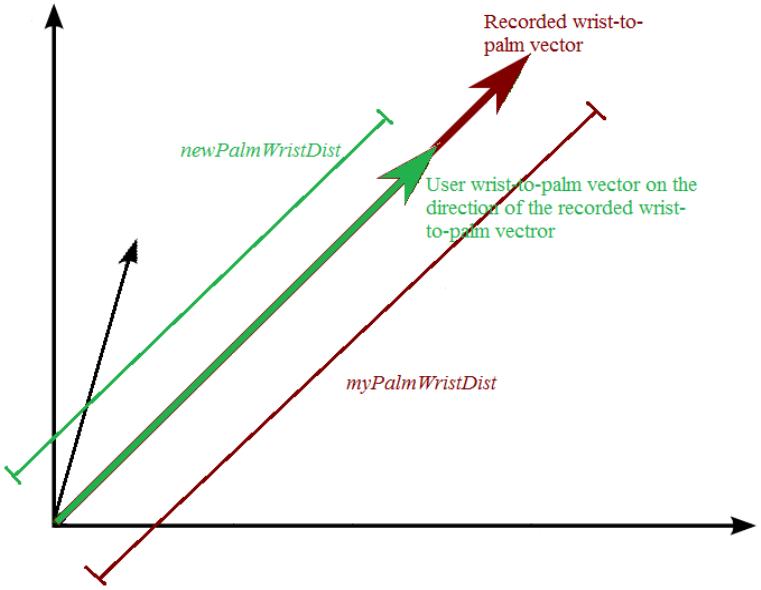


Figure 4.9: Optical representation of our vectors.

What we want to do is: find the direction of the vector that connects the recorded wrist and palm (red arrow in figure 4.8), divide it by its length so that we have a vector with magnitude equal to 1, multiply it by the length of the user bone, or in this case

[5] We have the coordinates of both wrist and palm and the method to calculate it was described in section 4.1.1

wrist-to-palm distance (green arrow), and add it to the position of the recorded wrist so that the new vector will be at the correct place in space.

We have now described the way that the palm will be put in its new position. We now have to do the same for the metacarpal bones, just this time, the part of the wrist that stays intact will be played by the beginnings of the metacarpal bones and so on.

Note: The program was tested on various users with a high range of palm sizes, and all of them managed to achieve 100% in at least one gesture under position scoring. This solely proves that the method described above performs its task.

Figures 4.10 and 4.11 represent the palms of two users with different palm size. The zooming is the same for both images. We can see that the first palm is larger than the other one. It is also clear that the goal spheres are adjusted to the palm size in each case. The fact that in both cases the users managed to achieve 100% means that their palm matched perfectly with the goal palm.

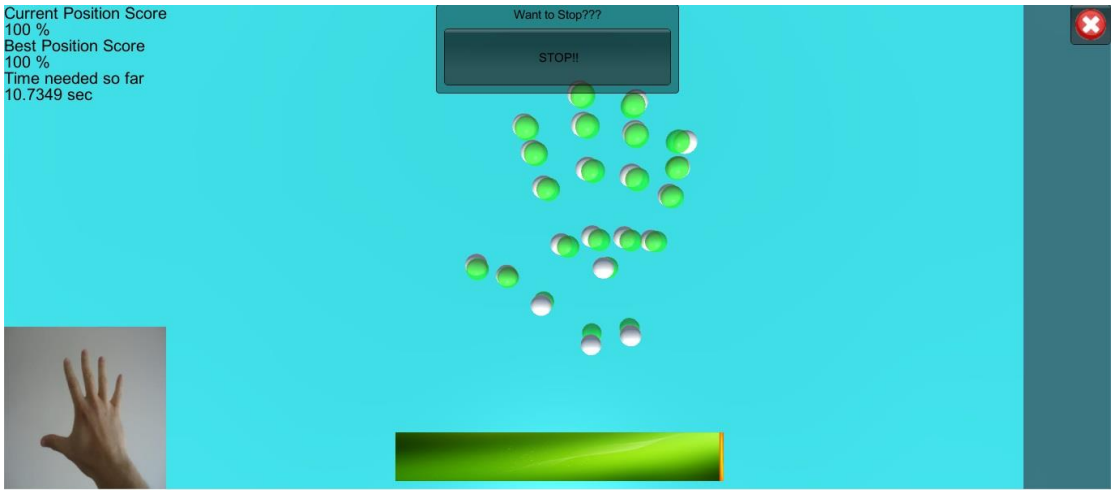


Figure 4.10: Larger palm

Figure 4.11: Smaller palm

4.2.4 Goals Revisited: Position Or Angle?

There are advantages and disadvantages in both scoring functions. Let's consider each case.

A great benefit that the *angle scoring* has is that the score that the user receives does not depend on the position of his hand in space, as long as it is in the Leap's field of view. That is of great importance because it won't matter if the user's hand is located 10 centimetres above the Leap, or 15. It won't matter whether his hand is a bit more to the left than the one on the screen, it only matters how well he is performing his task and that is the goal of this project.

The *position scoring* has the exact opposite property. It matters whether the user's hand is a bit higher or lower than the goal gesture. It affects if the one hand is a bit more front than the other. The user's hand has to be exactly at the position that my hand was when I recorded that gesture.

Luckily, things are not as bad as they sound. It takes a bit of practice to familiarize yourself with Leap Motion and how to move your hand above it, but once done, the user can achieve a good score in both *position* and *angle scoring function*.

Cameras play a crucial role in *position scoring*. The user will have to find the correct place in space for his hand, so he most probably will need to see, from a different point of view, his spheres in order to locate the exact position.

Note: Most users preferred the front and top view. Figure 4.12 shows the top view:

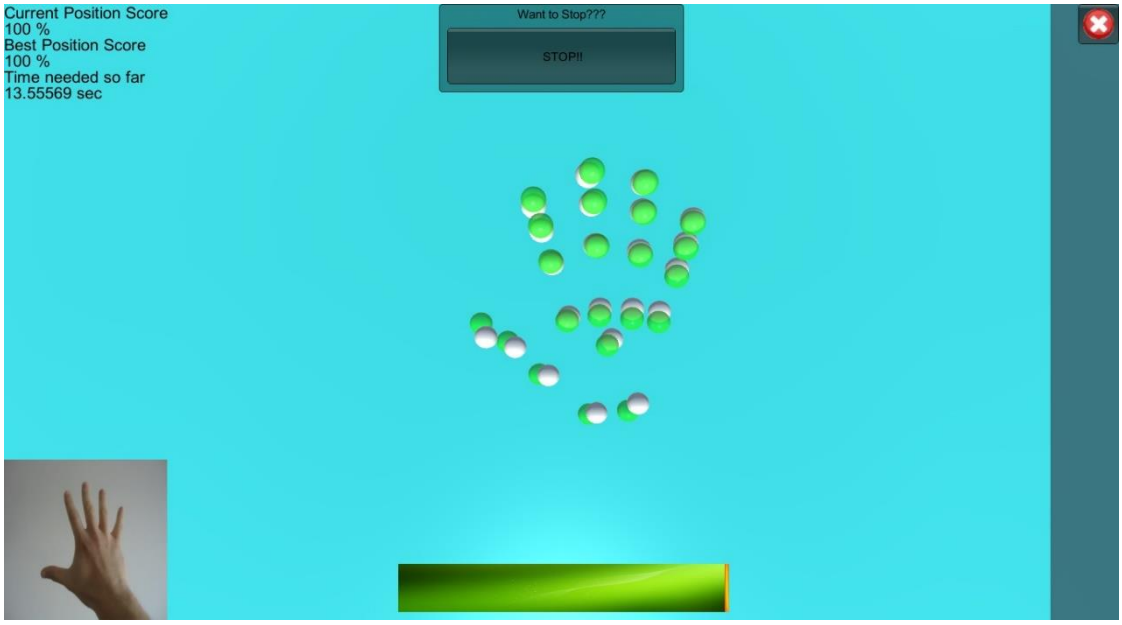


Figure 4.12: top view of the *position scoring function*

An advantage that *position scoring* has as opposed to *angle scoring* is that the user is able to see exactly what is wrong with his gesture at any time. What is meant by that is that if the user sees a red sphere, he knows that this sphere is not in the correct position. Which tells him exactly what he has to do to improve his score. The fact that *angle scoring* counts difference between float numbers, makes it clear that it is really

difficult for the user to know exactly what is wrong with his gesture. Colours help a lot in this situation, but the fact that a sphere is coloured green if a float number is less than a threshold, or that it "looks" like it is green, tells us that if a user sees a green sphere, it doesn't necessarily mean that his score is perfect, it means that is really good, but it may need improvements. In position scoring, seeing a green sphere means that it is precisely where is has to be. Stretching the angle scoring makes things better but yet again, the disadvantage remains.

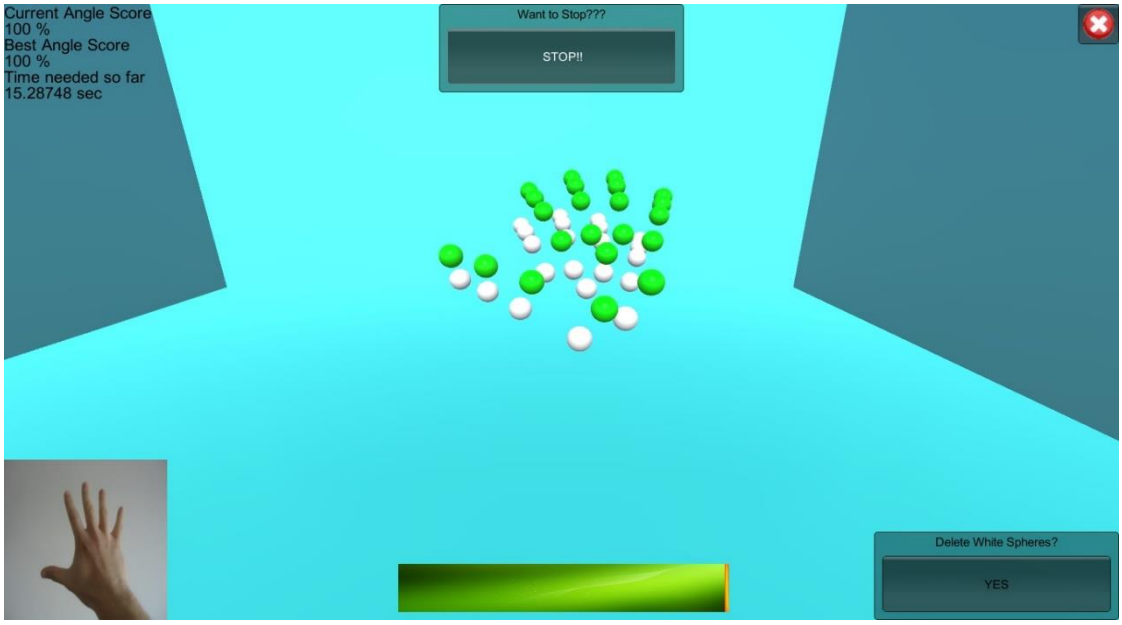


Figure 4.13: 100% in *angle scoring function.*

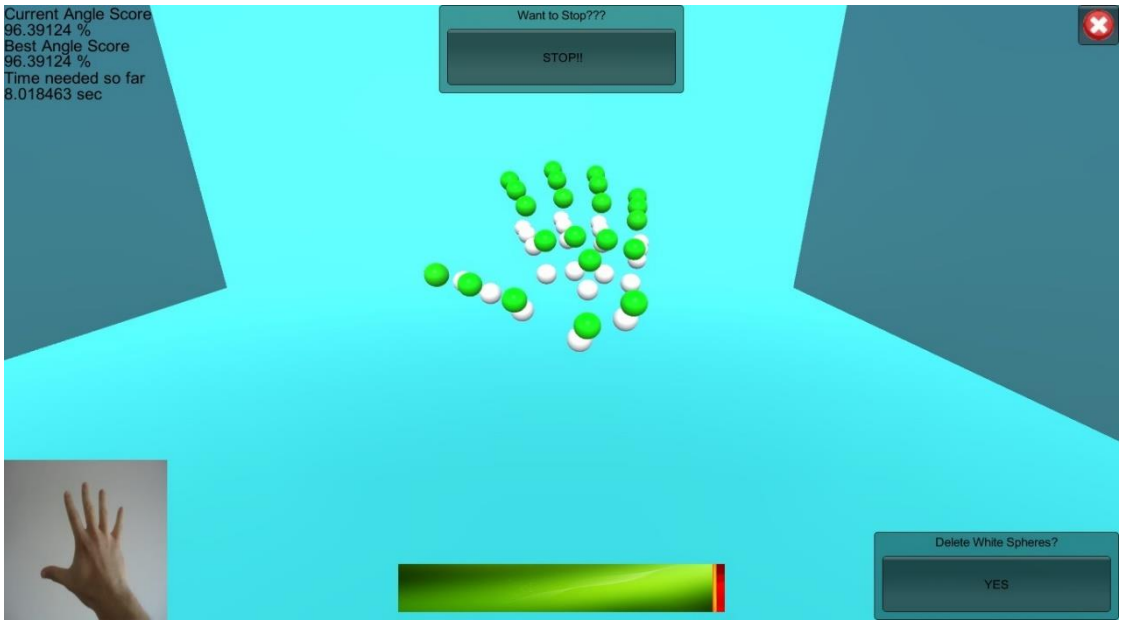


Figure 4.14: 96.39% in *angle scoring function*, every sphere seems totally green.

Note: It is hard to tell which picture is the perfect one just by looking at the colours in the case of the *angle scoring function*. But in the *position scoring function*, if every sphere is green, then the score is 100%.

An additional difference is that during *angle scoring*, the user can perform the gestures only by trying to mimic the image that is shown to him. That means that he can focus only on his fingers and their colours, not on the goal spheres. For that reason the button that deletes the goal spheres from the screen was created, so that the user can concentrate only on his hand, if that helps him. During *position scoring*, the user has to be focused on both his and his goal's spheres. This task increases the difficulty of completing the exercise.

Testing the game with various people showed that one way is not more preferred than the other. There were people that preferred the *angle scoring function* and others that preferred the *position scoring function*. The ones that preferred the first way mentioned that is more objective and easier, and the people that preferred the second way said that it was more challenging and straightforward.

5. Other Features Of The Game

The environment is divided between different parts. The user can navigate in the game by clicking buttons that appear on the screen. The sections of the game are described below.

5.1 Choosing Level

When the user runs the program, the first thing he sees is a menu which asks him to choose the level of the game difficulty. The way each level is different from another is simple and there will be a detailed description in the *Scoring Function* section.

This menu is important in the rehabilitation process. This application is designed to be used from the same person over a long period of time. It is hoped that the patient will make progress over time. This is why we need a way to be able to help the patient in every stage of his recovery.

A patient who suffers from a severe form of apraxia, will have a vast difficulty moving his fingers. For that reason, it would not be fair to give him a score that would discourage him from continuing the exercises. If that person has difficulty achieving a score of 40 or 50% when a healthy person achieves 98%, then giving him a low score may affect his confidence and his will to go on, especially if his score doesn't improve much over time. We do not expect from him to be able to produce a perfect match, this is why we accept a lower score and encourage his effort. The way this is implemented is by having different levels.

The number of levels can vary. Three levels were chosen for this game. In *level 1* it is easier to achieve a good score than in the other two. Respectively, *level 2* is easier than *level 3*. A healthy user would find challenging achieving a very good score in *level 3*. For that reason it is advised, people with motor difficulties to begin from *level 1*, and increase the level of difficulty, once they have achieved high scores in their exercises.

In Section 4.1.3 we defined a stretching function. This function plays a crucial role in the existence of each level. The difference between each level is that the constant values that were found empirically were adjusted so that the "new" 100% is 70% for the first level, or 85% for the second level. The level that was described in Section 4.1.3 was the third and hardest level.

So, if a user that chooses to work on the first level, and suffers from a severe type of apraxia -he can barely move his fingers- achieves a raw score of 40%, his new score will be 51%. If he achieves 50% he will receive a score of 68.8% and so on. This makes clear the importance of the stretching function and how it can be adjusted to every user.

Figure 5.1: Choosing Level.

5.2 Choosing Finger Difficulty

After having chosen a level, the user is asked to choose the difficulty that he has faced moving his fingers.

The purpose of this menu is simple but very important: to personalize the program to each user. It wouldn't be fair to score each user the same way. Especially if we consider that a person may have difficulty controlling all of his fingers, and another person has trouble bending the first two of his. It wouldn't be descent to score a healthy person the same way that a patient would be scored. So, apart from the fact that the user chooses the general level of difficulty for the game, allowing him to choose these numbers, makes the game more adjustable to the different kinds of users.

In essence, if the user chooses a left-hand-index difficulty to be 3 and all the others to be 1, his scores will be adjusted in such a way where, getting the index of his left hand in the correct shape or position, would give him higher score than any other finger. This choice can have both positive and negative results. Let's assume that the user chooses the above difficulties: *case 1*, and we calculate the score of that he achieves with all the difficulties equal 1: *case 2*. On the one hand, if the user manages to get his left-hand-index in the correct stance, his score will be higher in *case 1* than in *case 2*. On the other hand, if he does not manage to get his finger in the correct stance, his *case 2*-score will be higher. So this gives an extra motivation to try harder to move his problematic finger correctly. His effort will be awarded according to the difficulty he had chosen.

After the patient has chosen the numbers that represent the difficulty he has moving his fingers, a button appears at the top of the screen that allows him to move to the next stage. These numbers, which are integers from 1 to 5, are stored to arrays and they will play a leading role later when the scoring function will be described.

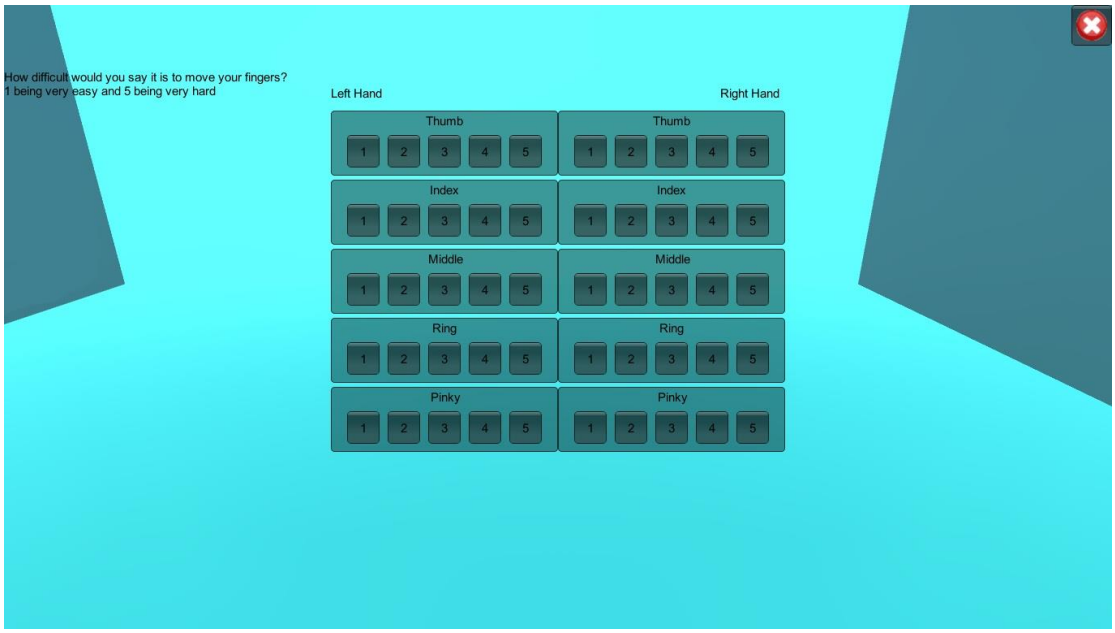


Figure 5.2: Choosing Finger Difficulty

5.3 The Result Catalogue

The screen that appears after the user presses the continue button when he has chosen the appropriate difficulties, is a stage where he can see his scores gathered after each exercise. The high scores, as well as the time needed to perform this score are both displayed. In contrast to the menus before, this list will appear after every exercise. Its context will be updated after every challenge.

At that stage, there are also instructions on how to use the cameras. Cameras are very useful, particularly if the user wants to achieve a high score, he may need to see his gesture from a different point of view to see how to improve the position of his fingers. Cameras are the devices that capture and display the world to the player. We can have an unlimited number of cameras in a scene. They can be set to render in any order, at any place on the screen, or only certain parts of the screen.

There are two ways to switch between cameras. One is with the arrows, and one with the buttons “w”, “a”, “s” and “d”. These two ways were chosen because the user can use either one depending on which hand he has above the Leap. The arrows are the convenient way if he works with his left hand and the other way is better if he works with his right hand. The user is also able to zoom in and out by scrolling up and down. The way to define a camera through a C# script is this:

```
public Camera mainCamera;
```

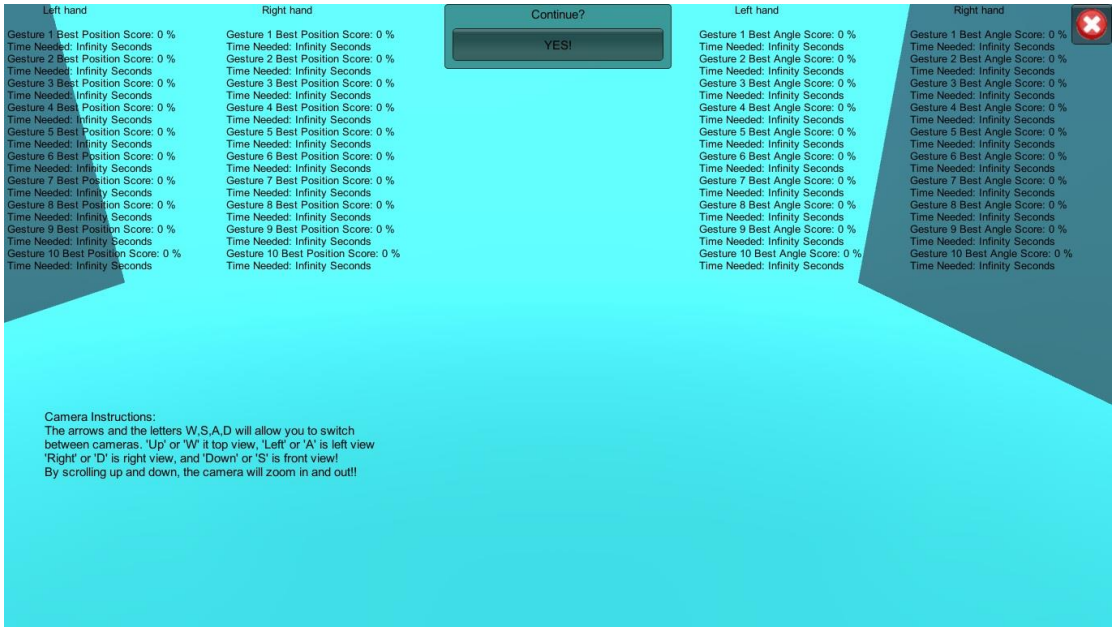


Figure 5.3: The result catalogue.

5.4 The Gesture And Scoring Function Menu

This is the main menu of the game: the gesture and scoring function menu. Two columns with buttons, that represent the left and respectively the right hand gestures, exist. When the user presses a gesture button, an image will appear at the button left corner of the screen. The image shows the gesture that the user has to perform, but in order to continue with his exercise, he will have to choose between two scoring functions which will be described in detail in later chapter. After the two choices are made, the menu disappears and the user is asked if he is ready to begin. The timing will start immediately after he presses that button so it is important that he is ready to start the exercise and that he has his hand above the Leap. It is recommended that the user always has his palm open when he starts an exercise. The reason is because Leap Motion sometimes doesn't recognise the hand immediately if some fingers are bent. If the hand is open when the exercise starts, then the user will not have any issues later.

At this stage, the user has the opportunity to set a time limit. At the bottom of the screen there is a button that if it is pressed, eight more buttons appear and each one contains a number: 10, 20, 30, 40, 50, 60, 90, 120, each representing a time limit in seconds.

Choosing a time limit is not suggested to people who haven't worked with the program before. It is a good exercise if the user wants to test his limits. Having chosen a time limit, the task becomes more challenging and strenuous.

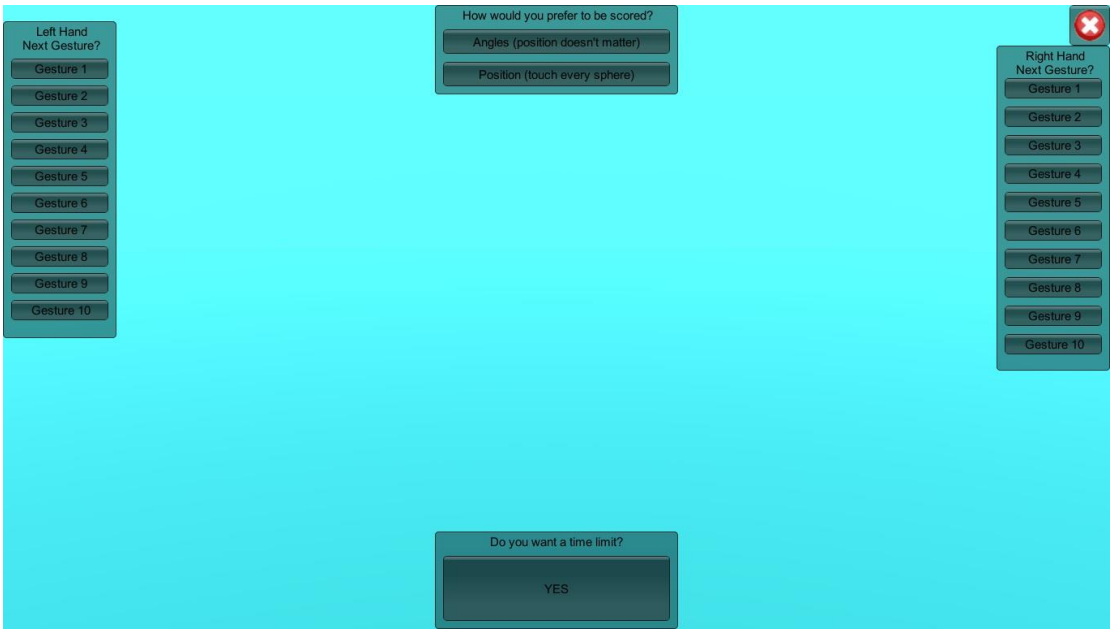


Figure 5.4: Gesture and scoring function menu before having chosen anything.

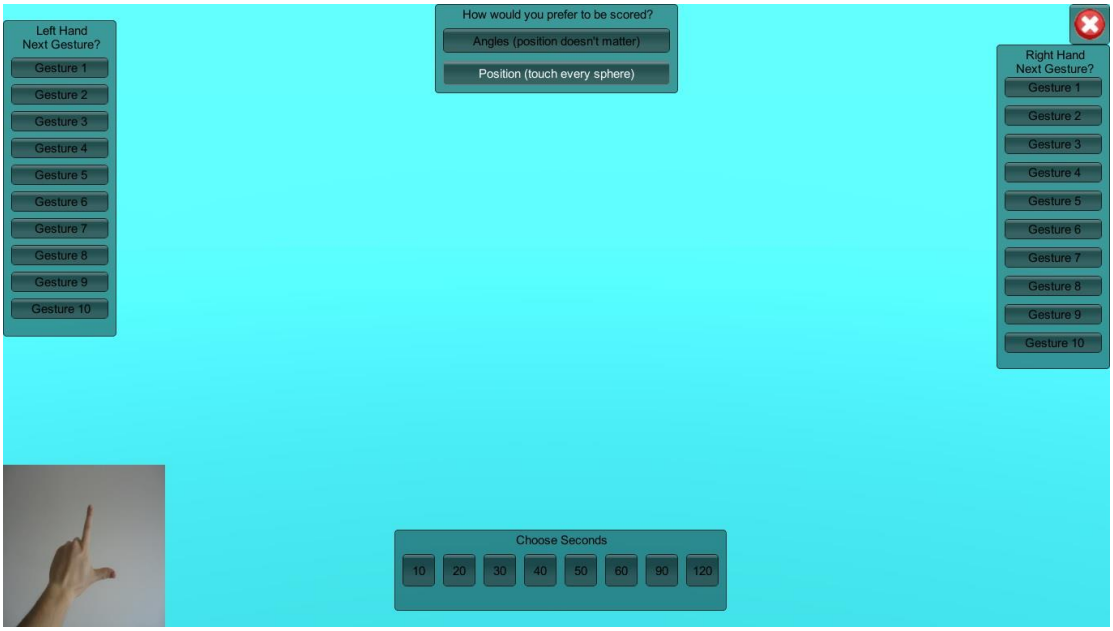


Figure 5.5: Gesture and scoring function menu after having chosen time limit and gesture.

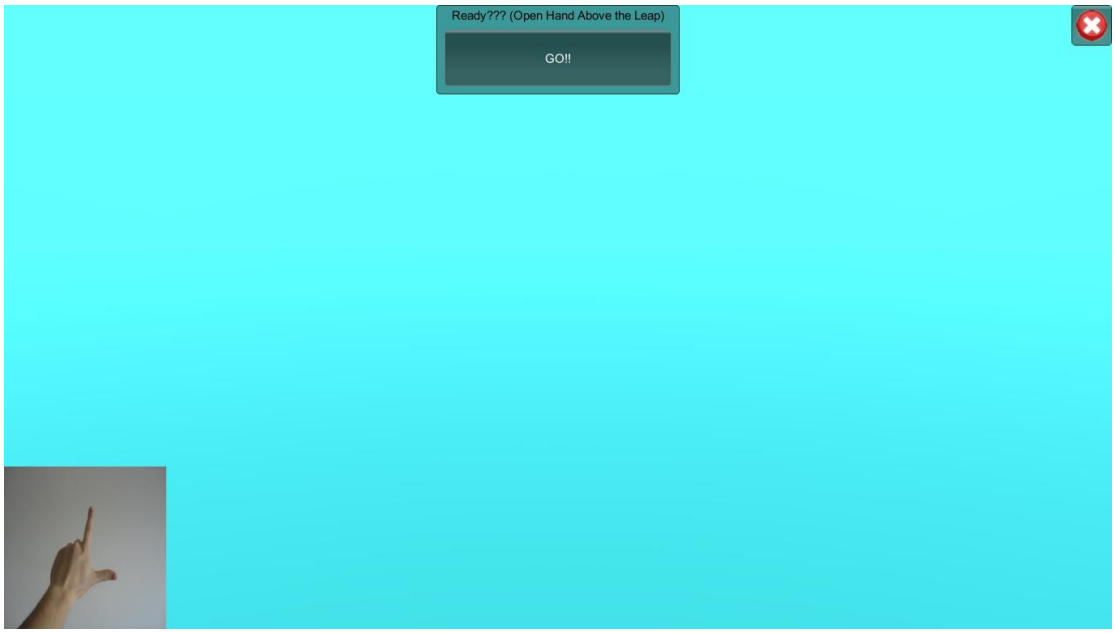


Figure 5.6: Gesture and scoring function menu after having chosen time limit, gesture and scoring function.

5.5 After Pressing Start

The moment that the user presses the *start* button the exercise will commence. Time is counted in seconds and shown on the top left corner of the screen along with the current and high score, which are updated in every frame.

It is vital that the user can see at any time what his current score is, that is because he will be able to know how far he is from his high score and he is able to continue trying until he has passed it. If the score only appeared at the end of his exercise, he wouldn't be able to improve it during the task. The exercise becomes more challenging that way. The colour of the spheres that represent the hand are also a fundamental way in which the user can improve his score, but this will be described later.

Building on to the previous statement, a progress bar is also shown. The bar is completely red if the user's score is 0% and fills up with a green bar as his score rises. If it reaches 100%, the bar is completely green. An orange stripe is placed along the red bar and it indicates the user's high score. So if his high score rises, the orange stripe will follow the end of the green bar and stay the closest possible to the end of the red one.

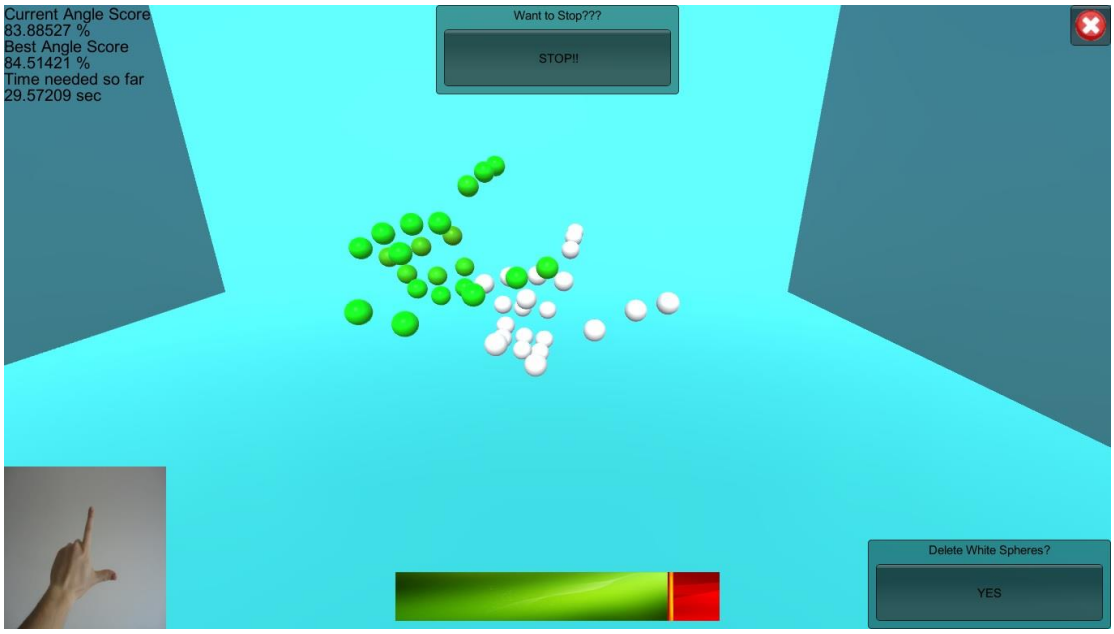


Figure 5.7: After having pressed the Start.

When the user feels that he is done with his task, he can stop the exercise by pressing the stop button. That will lead him to the catalog with his high scores and times. At that point he is able to see which gestures he hasn't tried yet, and which ones he could improve if he tries again.

5.6 Closing The Program

When the user has finally done his exercises, he is able to close the program by pressing the close button on the top right corner of the screen or by pressing the "ESC" button on his keyboard.

When he does this, two folders will be created, the "Scores" folder and the "LineGraph" folder. Both folders will be created inside a folder named "Data" which lies in the same path as our game. The first folder will contain the text file, named "dd_mm_yyyy" (current day), that file will contain his high scores, best times and the total time that he worked on each gesture. If the user deletes by mistake the folder, a new one will be created without any problem. If the user opens and closes the game more than once the same day, the previous scores of that day will not be deleted, the txt file will contain the information of both efforts. That way, the user is able to keep track of his progress over time. An example of how a row of that txt file will look like is this:

Gesture 2 Best Angle Score: 93.02872 % Time Needed: 16.58966 Seconds Total time worked on this gesture: 25.89545 seconds

The user will know exactly how well he has done, for how long he needed to perform the high score and how long he was working on each gesture in total.

The next folder will contain files that contain only the high scores. This folder will have the same properties, although it is recommended that the user doesn't interact with these files, because they will be used to create the line graph that shows his progress over time. If he deletes by accident this folder or a file, then the data will be

lost and the line graph will not be completed. The way this line graph is created will be described in later section.

5.6.1 Saving The User's Raw Data

The close button was used to perform a task which was not essential in the way the game worked but was essential for this project's purposes. While the game was tested on different people, after they were done with their efforts, their data had to be gathered in a file so that they could be reused later if needed. What is meant by that is that, similarly to the *Scores* folder that is created whenever the user shuts the game down, a folder called "RawData" was created that contained a txt file which contained all the data that existed the file in the *Scores* folder, but additionally, it also contained the exact positions, of the user palm when each high score was achieved, it also contained the angles between his fingers when the high score was achieved.

Having this data, played a key role in the final collection of information. Because of the fact that one of the scoring functions was subject to some alterations, the scores of the people that used the game before wouldn't be the same with the score that they would get after those changes. So the final count of them wouldn't be possible. But having saved all the information needed to regenerate the gesture that gave the user his high score, made it possible to recalculate each user's score by using the final scoring function.

6. Recording Longitudinal Progress

As mentioned in earlier section, after the user closes the program, a folder named "LineGraph" will be created. This folder will contain only the user's high scores. Each time the program is closed, 40 lines will be appended to a txt file which is named after the current date in the format "yyyy_mm_dd", the first 20 correspond to the position scores and the next 20 to the angle scores. So if the user works on his exercises 3 times a day, the corresponding txt file will contain 3·40=120 lines with one number each row.

After the user has been working on the program for more than a week, he may want to see his progress. He can always see the files in the folder "Scores" but having to check every score of every gesture in detail, especially for a long period of time, will be very time consuming. So the need for an automated way of gathering the data is created. In addition, being able to see your progress in a more optical way, such as a line graph, has better results.

Two Java classes were made to complete this task. The program was exported as an executable Jar file.

The first class will use the path to the desired file to count the number of lines in that file and read each one of them which will be put in a String array and returned to the main function.

So now, in our main function, after we have read the names of all the txt files that exist in our folder, we will read all the non-zero position and angle scores and produce two means, one for each scoring function. We don't take under consideration the zeros, because if a score is absolute zero, that means that the user didn't try that gesture at all, so it wouldn't be fair to count it as a score.

Then, the line graph is produced. It will count days depending on the number of txt files that are read and it will produce a line graph for the position scoring and one for the angle scoring.

Note that we could have created a line graph for each gesture, but then it would be chaotic to see what you are interested in. Note also that by using the format "yyyy_mm_dd" our files will always be sorted in our folder, so when our program reads them, they will be in the correct order, the oldest first.

6.1 An Experiment

I performed gestures 4 and 10 for the left hand and gesture 7 for the right hand for a one-week period. So, each day, I executed the above gestures for angle scoring only. The scores that I got were high, close to 99% for gestures left-4 and right-7 and close to 93% for gesture left-10. After I ran the program this was the outcome:

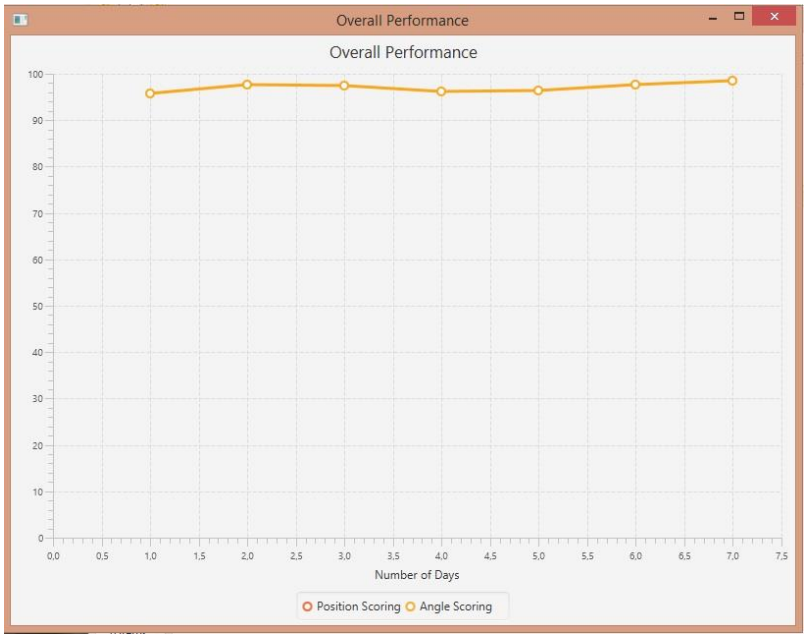


Figure 6.1: Line graph of one week's performance.

We can see that despite the fact that there are no grand fluctuations, there was a relatively poor initiation on day one, then followed a dip on day four and a gradual increase of the score followed on the last day of the experimenting period.

Note that there is no line representing the Position Scoring. That is because no gesture was performed under this scoring function.

Another example, with a higher variety of results over a four day period is the following:

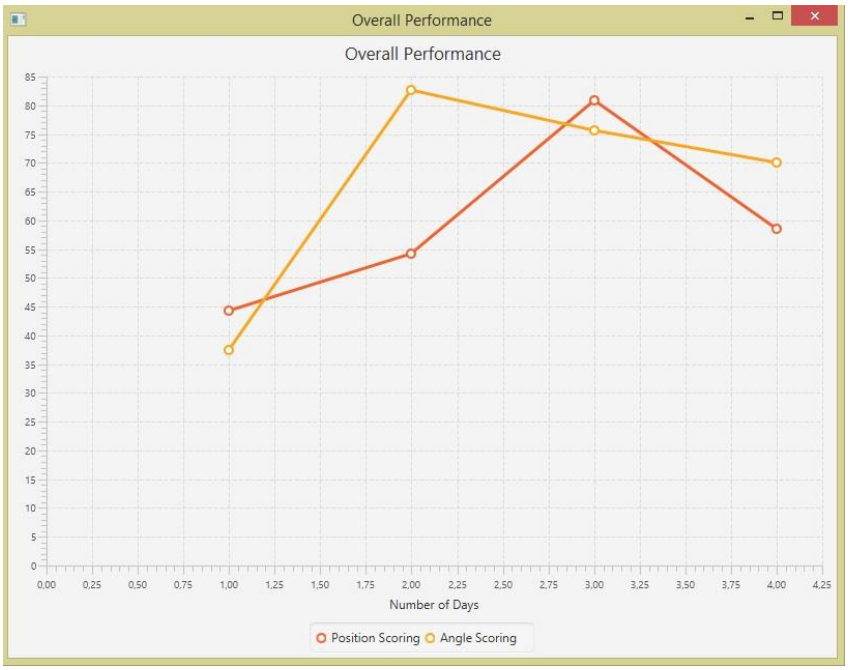


Figure 6.2: Line graph with high variety of results over a four day period

The scores in this example were chosen at random so that there can be more steep fluctuations, for the sake of argument.

Thus, it is made clear why having a way to represent the user's results is necessary. Visualization of the data brings a clearer outcome.

7. Limitation Of The Leap Motion

7.1 Self – Occlusion

In the official website of Leap Motion [5], in the section titled: "Introducing the Skeletal Tracking Model" the writer states:

"By modelling a human hand, the Leap Motion software can better predict the positions of fingers and hands that are not clearly in view. Five fingers are always present for a hand and hands can *often* cross over each other and still be tracked. Of course, the controller still needs to be able to see a finger or hand in order to accurately report its position. Keep this in mind when designing the interactions used by your application. *Avoid requiring complex hand "poses" or subtle motions, especially those involving non-extended fingers*."

We have kept this in mind when creating and running our program because there are numerous gestures that demand the user to bend his fingers in a way so that they cross over each other, or they are located below the palm. There may be cases where Leap Motion does not understand the exact location of some fingers and the gesture shown in the computer does not correspond to the one the user is performing.

The Leap Motion software uses an internal model of a human hand to provide predictive tracking even when parts of a hand are not visible. The hand model always provides positions for five fingers, although tracking is optimal when the silhouette of a hand and all its fingers are clearly visible. The software uses the visible parts of the hand, its internal model, and past observations to calculate the most likely positions of the parts that are not currently visible.

Knowing all this, we use the game created by bearing in mind that it is highly likely to fall on a case where the game doesn't work properly. The user should always be informed that if he finds himself in a situation like the one described above, he should open his palm, so that the Leap understands his gesture completely, and then continue his effort.

A situation like that may occur most likely in the gestures 2, 3, 7 and 10:

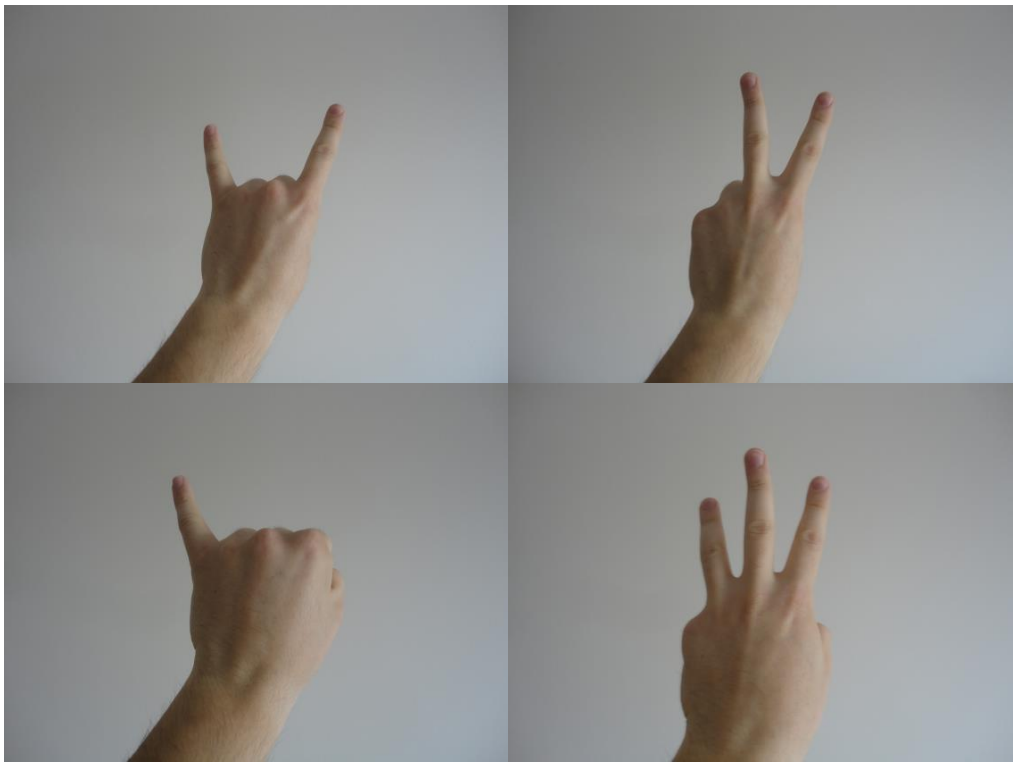


Figure 7.1: Self-occlusion.

Because these are the gestures that have fingers which bend one below the other.

An example of how a “wrong” gesture may look like is this:

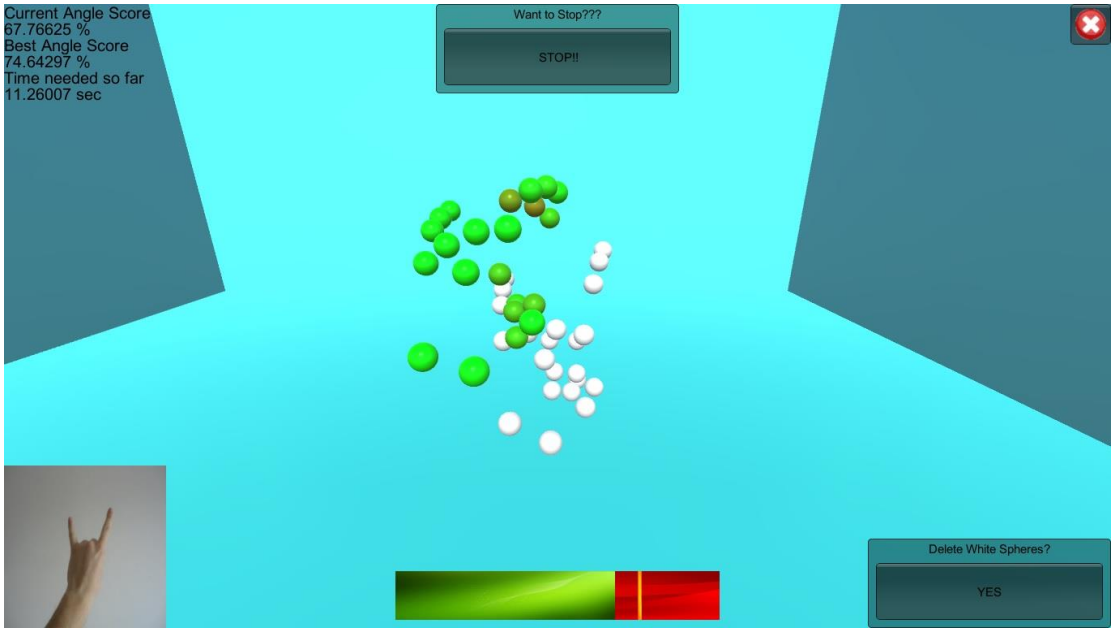


Figure 7.2: A “wrong” gesture.

When this picture was taken, my hand was exactly as the one on the image on the bottom left corner of the screen (gesture 2). But the Leap placed my middle finger higher than it really was.

7.2 Field Of View

Apart from the above disadvantage of Leap Motion, the user may also cause such a malfunction of the program. If, for example, he places his hand too close to the Leap, or his hand is at the boundaries of Leap's field of view. It is very common for a new user to place his hand very close to the machine, believing that Leap would perform better that way.

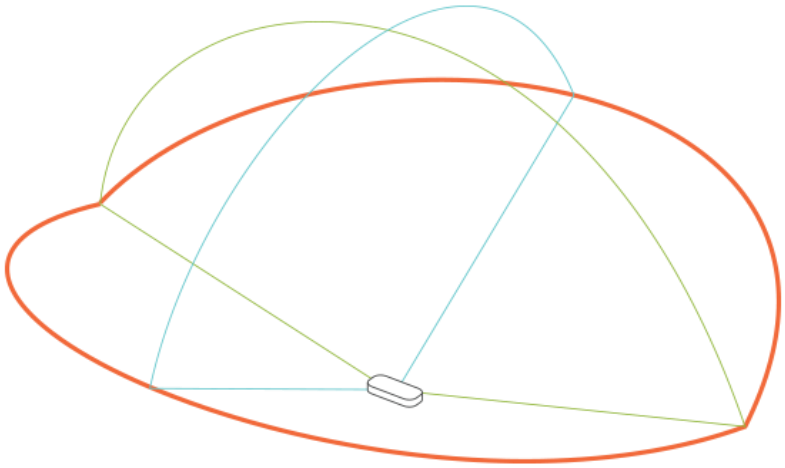


Figure 7.3: The field of view of Leap Motion: 150° field of view with roughly 8 cubic feet of interactive 3D space.

So, to sum up, a new user should always be informed that such cases may occur, he may not see his hand on the screen, his hand may disappear, or the gesture that he sees on the screen is not the same as the one he performs. When each of these cases occurs, he should simply open his palm and try again. That is the reason why the user should start each of his exercises with an open palm, so that the program understands from the beginning all the details of his hand.

These are problems that come with Leap Motion, sadly there is nothing we can do to fix them, apart from informing the user for their existence.

8. Contribution

As mentioned in previous section, we will have to keep in mind the imperfections of Leap Motion. Some scores would be higher if the machine worked in more detail on occasions like Gesture 2 or Gesture 3.

We should also remember that it may take some time for the user to get used to the way the machine works and how exactly the user should bend his fingers in order to improve his score.

The process of data collection has been time consuming. As a result, some users may have stopped their effort because they didn't possess the time required, so their scores were lower than they would normally be. For the same reason, some users did not work on all the gestures, so there may be fluctuations between the total number of people that worked on each gestures.

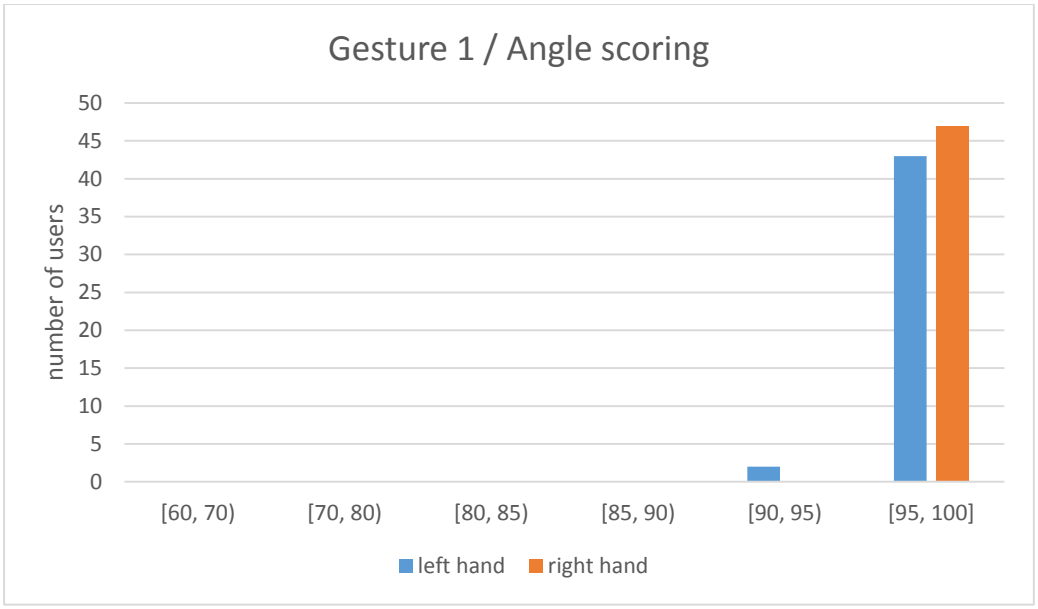
Because of the high amounts of time required to complete the whole game, it was chosen that the users would perform most gestures according to the *angle scoring function*. Nevertheless, most of them also worked on some gestures using the *position scoring function*.

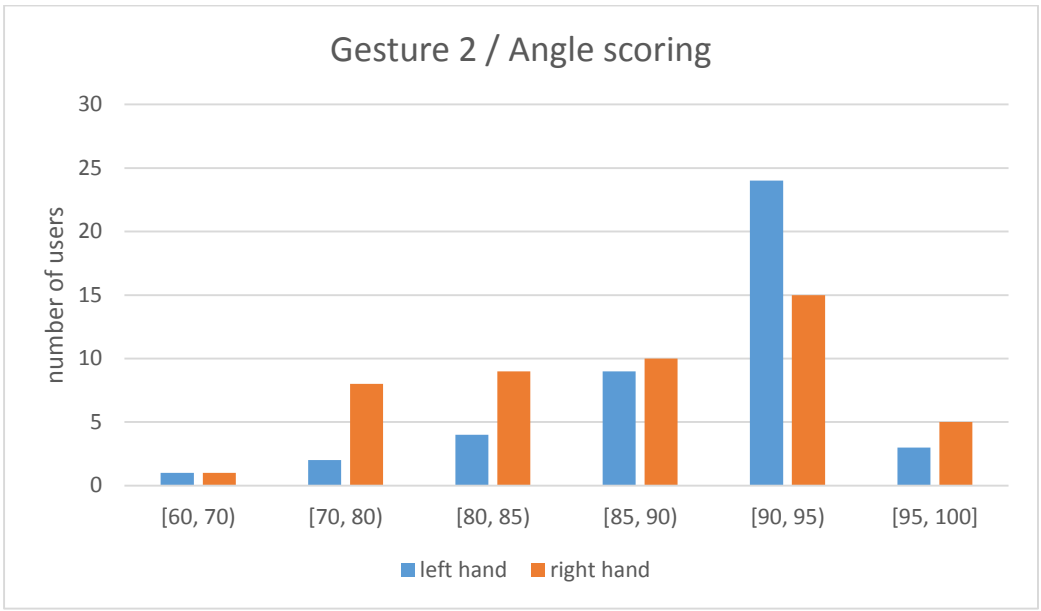
Finally, we should keep in mind that the data below were gathered mostly from healthy controls whose age varied from 20 to 25, with few exceptions. Those few exceptions were older, but did not perform worse than the others. Gathering more data from a wider range of ages would produce stronger results.

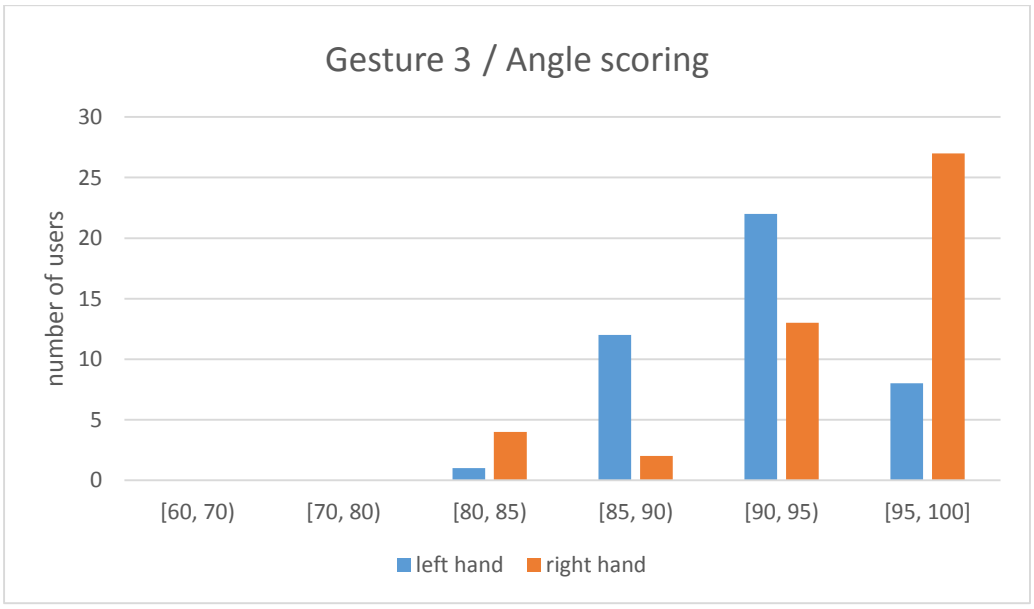
8.1 Results

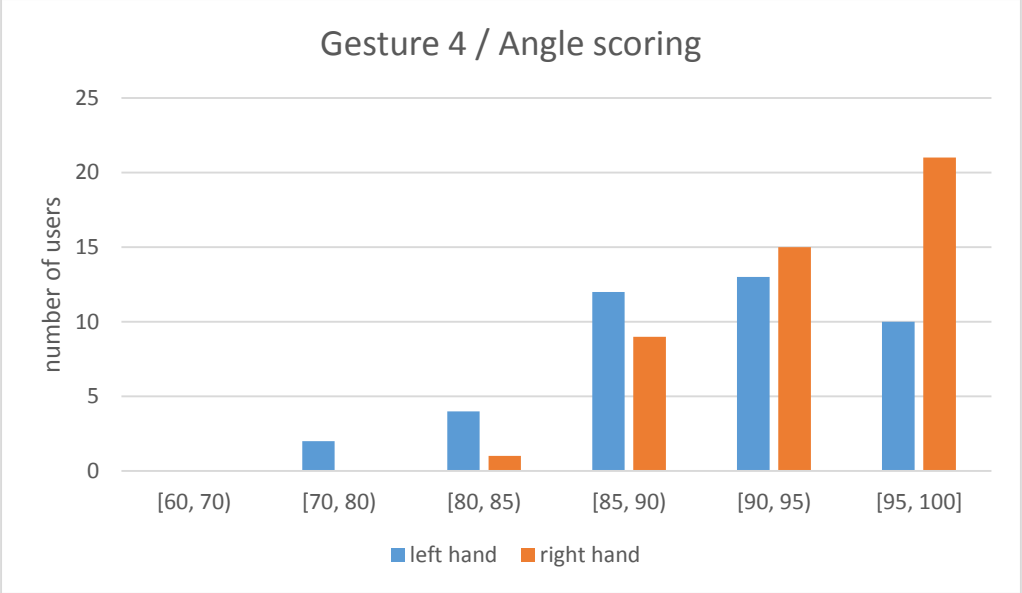
Data was gathered from 55 healthy controls and it describes the *angle scoring function* (we will present the *position scoring function* results in a less detailed way). Their scores were taken in *level 3* and all finger difficulties equal to 1.

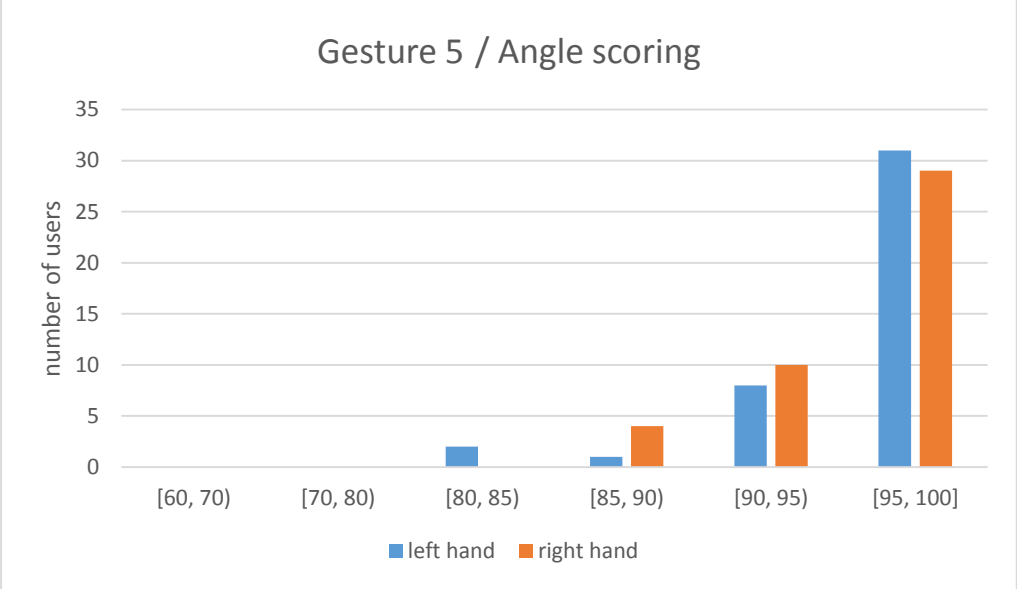
We will create a bar graph for every gesture. We will divide our data into the following intervals: [60, 70), [70, 80), [80, 85), [85,90), [90,95), [95, 100] and we will count the number of users whose respective scores were contained into these intervals.

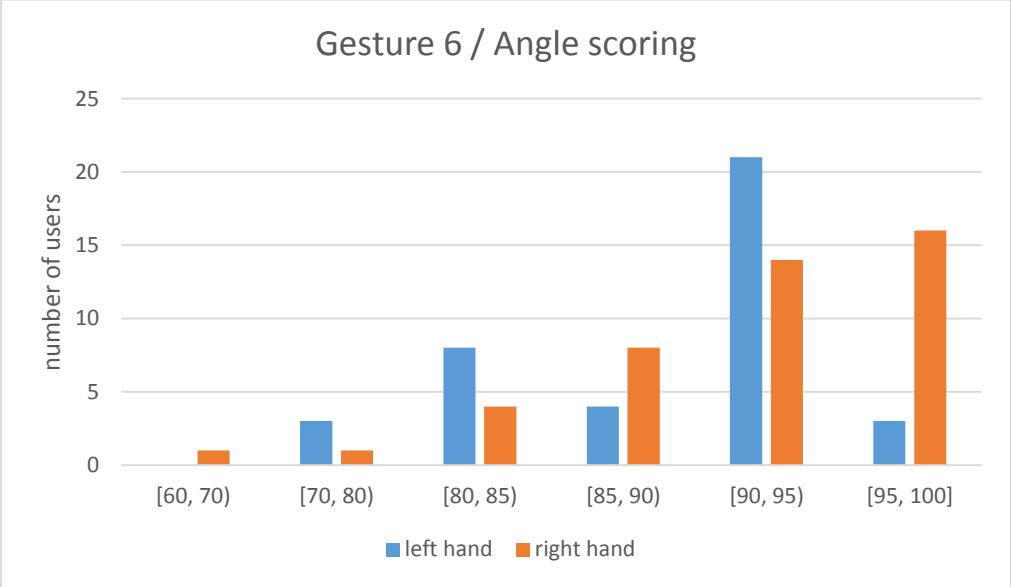

Gesture 1 / Angle scoring
number of users
50
45
40
35
30
25
20
15
10
5
0
[60, 70)
[70, 80)
[80, 85)
[85, 90)
[90, 95)
[95, 100]
left hand
right hand

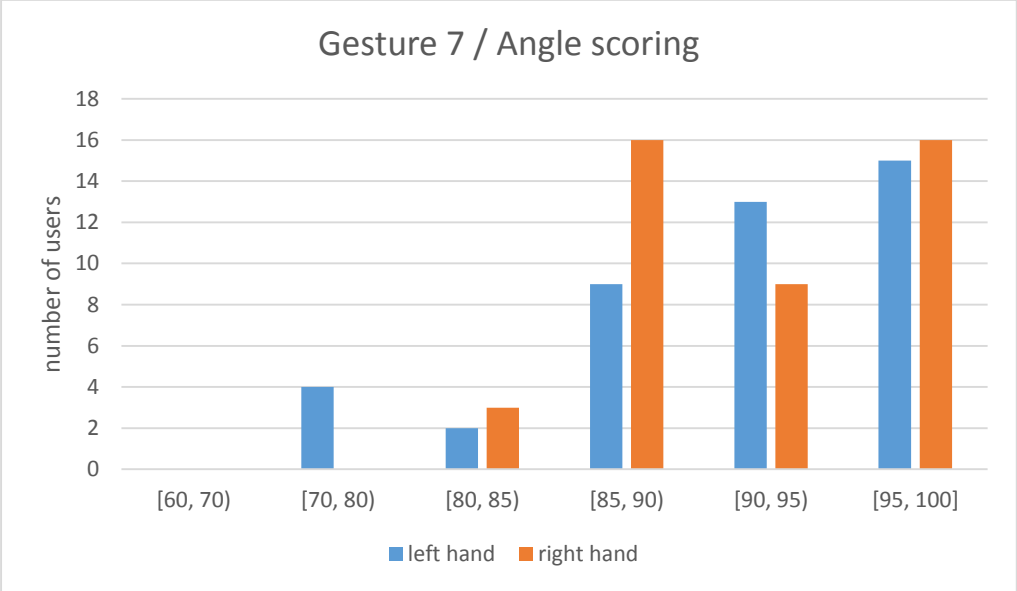

Gesture 2 / Angle scoring
number of users
30
25
20
15
10
5
0
[60, 70)
[70, 80)
[80, 85)
[85, 90)
[90, 95)
[95, 100]
left hand
right hand

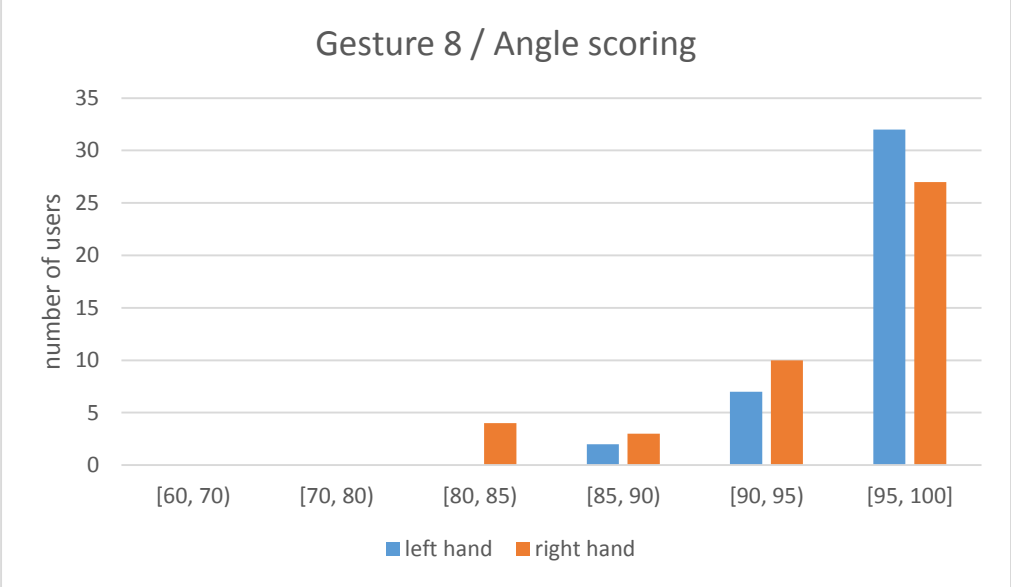

Gesture 3 / Angle scoring
number of users
30
25
20
15
10
5
0
[60, 70)
[70, 80)
[80, 85)
[85, 90)
[90, 95)
[95, 100]
left hand
right hand

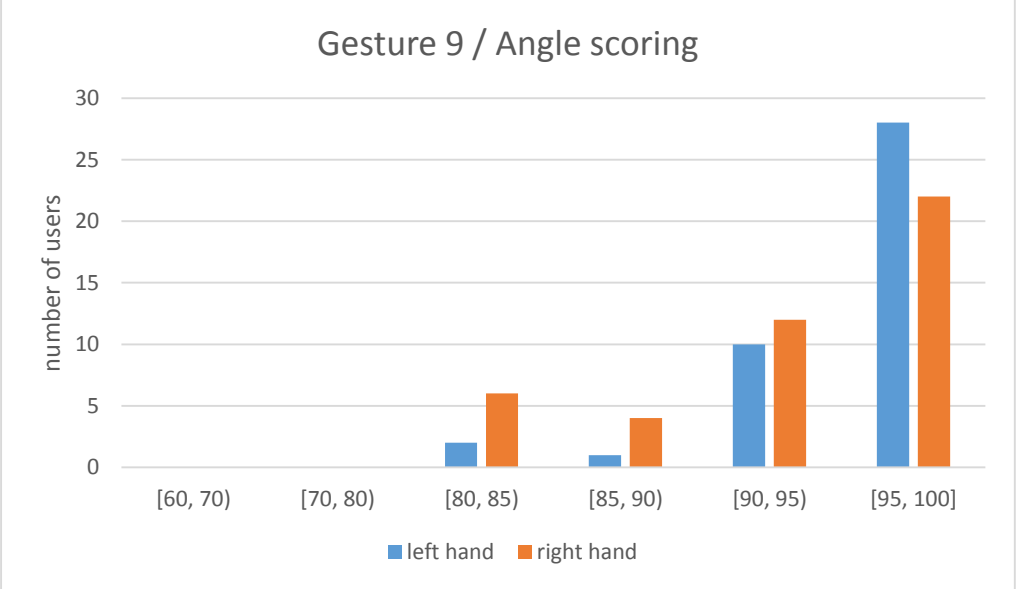

Gesture 4 / Angle scoring
number of users
25
20
15
10
5
0
[60, 70)
[70, 80)
[80, 85)
[85, 90)
[90, 95)
[95, 100]
left hand
right hand

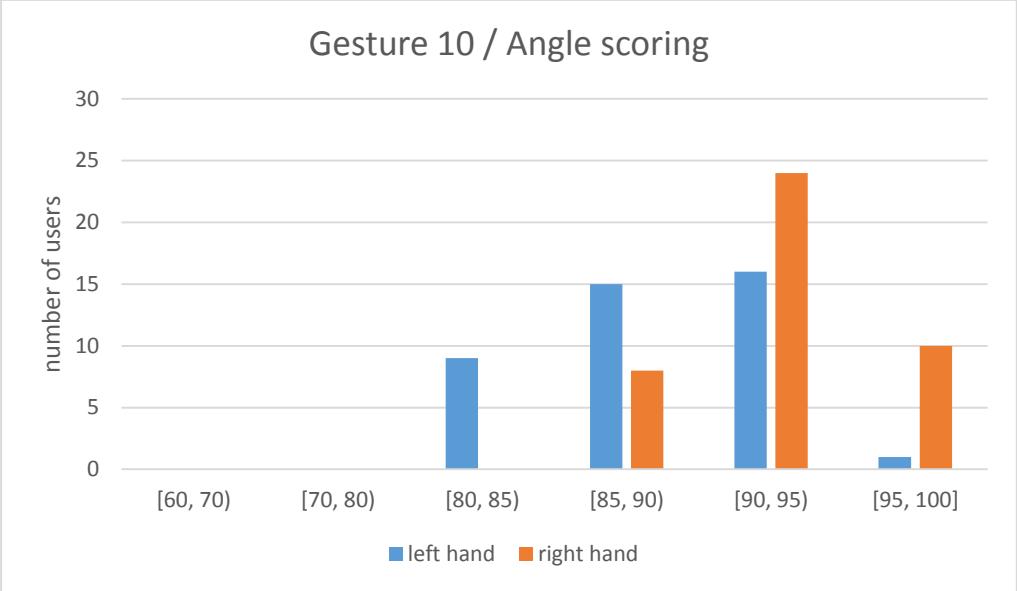

Gesture 5 / Angle scoring
number of users
35
30
25
20
15
10
5
0
[60, 70)
[70, 80)
[80, 85)
[85, 90)
[90, 95)
[95, 100]
left hand
right hand


Gesture 6 / Angle scoring
number of users
25
20
15
10
5
0
[60, 70)
[70, 80)
[80, 85)
[85, 90)
[90, 95)
[95, 100]
left hand
right hand


Gesture 7 / Angle scoring
number of users
18
16
14
12
10
8
6
4
2
0
[60, 70)
[70, 80)
[80, 85)
[85, 90)
[90, 95)
[95, 100]
left hand
right hand


Gesture 8 / Angle scoring
number of users
35
30
25
20
15
10
5
0
[60, 70)
[70, 80)
[80, 85)
[85, 90)
[90, 95)
[95, 100]
left hand
right hand

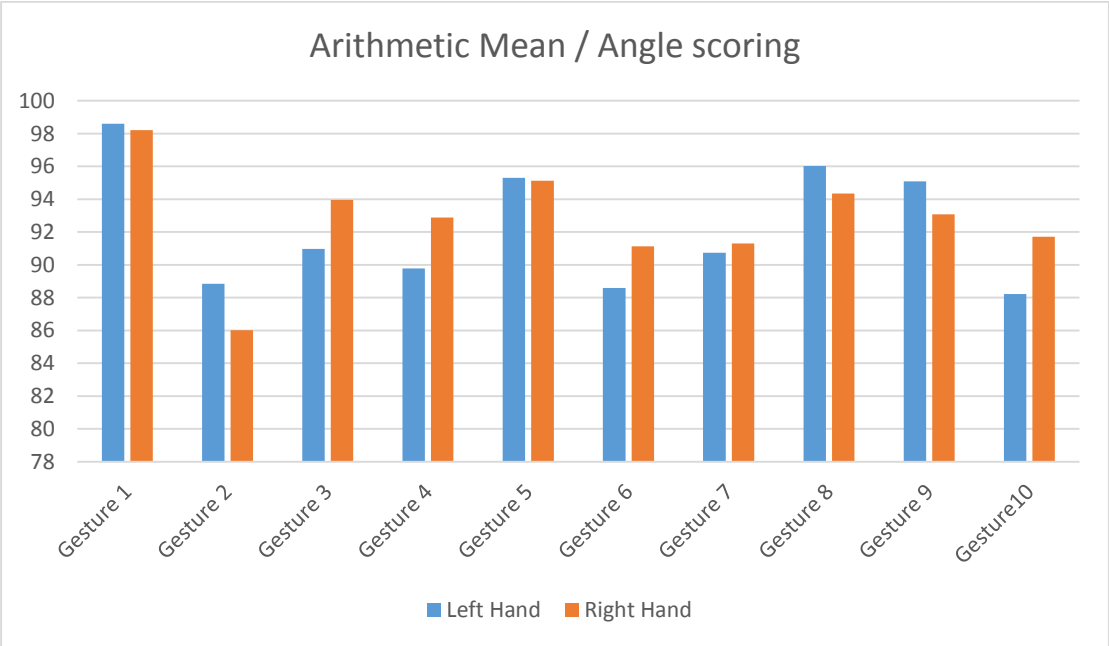

Gesture 9 / Angle scoring
number of users
30
25
20
15
10
5
0
[60, 70)
[70, 80)
[80, 85)
[85, 90)
[90, 95)
[95, 100]
left hand
right hand



8.2 Analysis

8.2.1 Angle Scoring Function

The arithmetic mean for each gesture for the *angle scoring function* are presented below:

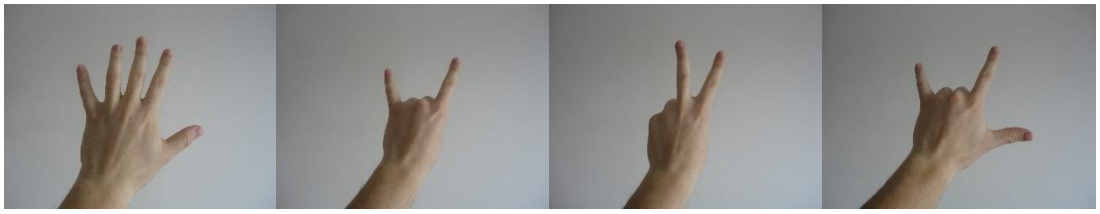


By taking a closer look to the scores we can see that the overall performance of the left hand of the users is approximately the same as the right hand one, so we will examine each gesture as one, not once for the left and once for the right hand.

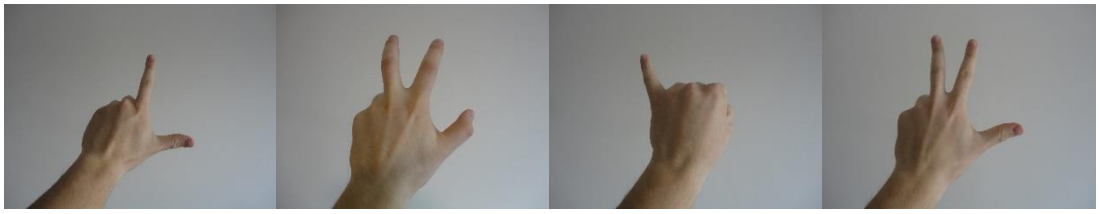
The average scores are presented in the table below in a decreasing order.

Gesture 1	98.40
Gesture 5	95.21
Gesture 8	95.14
Gesture 9	94.04
Gesture 3	92.52
Gesture 4	91.42
Gesture 7	91.02
Gesture 10	89.98
Gesture 6	89.94
Gesture 2	87.36

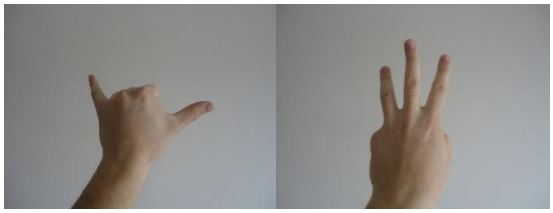
Table 8.1: Average scores for Angle Scoring.



(Gesture 1) (Gesture 2) (Gesture 3) (Gesture 4)



(Gesture 5) (Gesture 6) (Gesture 7) (Gesture 8)



(Gesture 9) (Gesture 10)

Gesture 1 is naturally the one with the highest score since it is the plainest one and the Leap doesn't have any difficulty recognising the fingers.
It is not a surprise that the next gestures are *5* and *8*. They both don't have interactions with different fingers and they are fairly easy to perform (for a healthy control).
Gesture 9 also does not have such problems but it carries a difficulty that the previous ones do not. It requires the little finger to be up while the ring finger is down. This requirement can be very challenging for some people.

This can be explained by the complex anatomy of the fingers of the hand. The movement of the fingers is performed through the tendons-muscles and it is controlled by the nerves. While the movement of the stretching of the fingers is controlled by the

same nerve (radial nerve), the movement of the bending is different for the ring finger and the little finger (ulnar nerve) in comparison with the other fingers (median nerve). In addition, it is of high importance that there are connections between neighboring tendons. Thus if the tendon of a single finger moves, it affects the neighboring finger to do the same movement. That is why when the ring finger is bent, the pinky finger tends to bend as well.

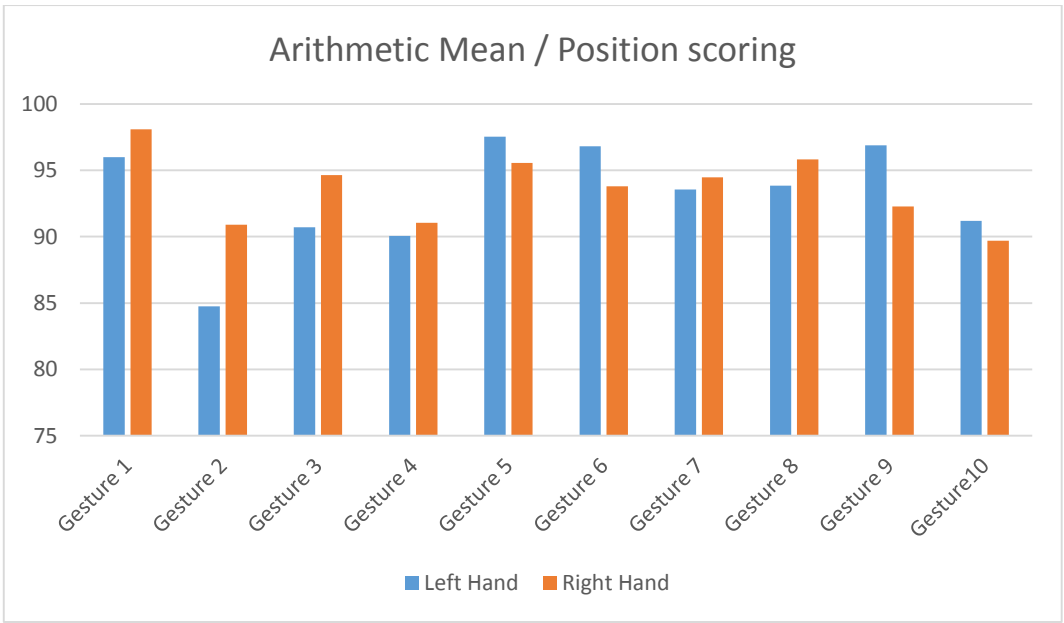
Gesture 3, on the other hand, requires the thumb to be below the ring finger, that is the reason why its score is so much lower than the previous ones.
Gesture 4 is a harder version of *Gesture 9*, gesture *7* as well.
Gestures 10 and *6* don't have any problems with the Leap, but it is really difficult to recreate the exact same angles as the ones recorded. In addition, *Gesture 10* is one of the hardest ones from its nature. Most users had felt pain when they tried this gesture for a long time continuously.
The fact that *Gesture 2* has the worst score is not a surprise either. It requires the ring finger to be separated from the pinky finger, and additionally it creates problems with the Leap. This gesture is the hardest one in terms of both human and machine capabilities.

So we notice that the data collected reflect the reality and our results can be explained with precision.

At this point it would be safe to say that 92.52% is the approximate total score of a single healthy control, which is a good score, considering the difficulties mentioned.
We should also keep in mind that the users tried the program once and none of them had worked with Leap Motion before, so they weren't familiar with it. Thus there could not be any vast improvement in their scores. If they had had time to work on the game longer, they would have performed better.

8.2.2 Position Scoring Function

The arithmetic means for each gesture for the *position scoring function* are presented below:



Once more we will examine each gesture as a single entity and not as two. The table 8.2 below contains the average scores for each gesture concerning the *position scoring function*.

Gesture 1	97.34
Gesture 5	96.65
Gesture 6	95.54
Gesture 9	95.11
Gesture 8	94.80
Gesture 7	94.06
Gesture 3	93.03
Gesture 4	90.51
Gesture 10	90.47
Gesture 2	87.45

Table 8.2: Average scores for Position Scoring.

The first thing that we notice is that the difficulty described before exists for this scoring function as well. Only the score of *Gesture 6* soared. Apart from that we can see that *Gesture 2* is once more at the bottom of the table and *Gesture 1* is at the top.

It is also notable that the average position score of most gestures is higher than the respective angle score. This should not surprise us since it is far easier to achieve 100% in this scoring function than the previous one. There were users that found the *position scoring way* more addictive and they tried more to achieve a perfect score, on the other hand the users that preferred the *angle scoring function*, achieved a really high score but not perfect, in most gestures.

The approximate total position score of a healthy control is 93.96%, which is almost 1.5% higher than the previous average score.

8.3 User Feedback

The people that used the program were asked to give some feedback. They responded with comments on what they liked and what they believed would make the game better. They explained which scoring function they preferred and why. Here are some of them:

"I liked the fact that I saw my hand as a collection of spheres. It was a challenge to perform some gestures. The whole procedure was a bit more time consuming than I expected." – Panagiotis.

"The truth is that it was kind of addictive to try to eliminate all the red spheres! I had a hard time understanding what I had to improve. The program felt more like a game, it was really interesting and at the same time very demanding on several gestures. Because I tried all the gestures my hand got tired and I think that if I tried again I would get better scores." – Maria.

“The experience was pleasant and challenging. The environment was simple and understandable. The only disadvantage for me was that the picture of the gesture that I had to reproduce didn’t allow me to see the exact position of the fingers on the bottom side of the palm. I preferred the position scoring way because it was easier to achieve your goal despite the fact that the photograph wasn’t really helpful.” – Nikos.

“I liked Leap Motion. I also liked the fact that the user had the ability to switch between cameras. Some gestures were massively difficult. But the program was cool and amusing and if the Leap didn’t have those problems it would be much better.” – Georgia.

“I preferred the position scoring function, mainly because I could get 100% with it.” – Tomas.

“It was interesting seeing a simulation of my hand on the screen. It was a challenge and I wanted to achieve the highest score I could, like playing a game. Really amusing and not much tiring. Some gestures were hard but that is not a drawback. The fact that you didn’t understand a detail that you had to achieve was annoying. I preferred the angle scoring function. I would like to try it again, it was new-found.” – Zafeiris.

“The angle scoring way was a bit boring. I didn’t know what I had to improve. I think position one was better, it was nice to see what exactly was wrong.” – Katerina.

“For me it was easier to work with distances. The spheres were truly helpful. I had a hard time to understand what I had to do with the fingers that were located below the palm. I really enjoyed the experience because it was like a game, not a boring procedure and it made me particularly happy that this project can be used to help people that have a motor difficulty to improve and smooth their gestures.” – Hara.

“I didn’t really like that the sensor wasn’t much sensitive on some gestures, I needed to try from the beginning my gesture several times because the sensor didn’t understand exactly what I was doing. The program was really pleasant and friendly. I believe the angle way was the hardest one although I believe that this way is more useful for a patient because the position of his hand is not essential. Because the user treated the program as a game, he got more addicted to achieve a good score, which led to a pain in the hand sometimes. The position way was more fun, it was easier to get a high score or even a perfect score, something that was a confidence boost. The only disadvantage was that the procedure needed a lot of time to be completed.” – Dimitris.

“I found the exercise creative, smart and practical, because everybody can test his palm flexibility. By trying to improve my score, my palm hurt, but this may be a good thing because it could reveal a hidden disorder that I wasn’t aware of. I really enjoyed it because I saw it as an additional tool in the field of Medicine.” – Aggeliki.

“Every time that you achieve a high score in a gesture, you get motivated to achieve the same or a better score in the next gesture, even if that is a harder one. It becomes more addictive when you watch the red and the yellow spheres diminishing and the green ones increasing. You know that you are closer to your target and it is harder to give up.” – Markella.

Figure 8.1: Gathering data informally from friends. In a patient surgery the setup would require the sensor to be placed in a set location, and each patient to sit a set distance away from it.

9. Conclusions

It its final form, the game that we created was a success. We managed to achieve our initial goals and requirements. Both scoring functions we designed appear to be suitable for the task. Further randomized control trials will be used to decide which scoring method to use.

Our application can be used from apraxia patients during their rehabilitation period. It can adjust its functionality according to each user's needs and it can provide them with feedback. The program that we created is designed to be used from the same patient over a long period of time. The user can keep track of his progress, and as the time passes and his condition improves he can try a harder version of the challenges.

But apart from the rehabilitation process, this application can be used for diagnosis as well. Up until now, doctors were scoring the patient's gestures and made decisions about their condition. Our program can be used to diagnose the patient's condition. The patient could be asked to perform a number of gestures above the Leap and according to his final score and the time that he needed to achieve that score, a decision would be made. Depending on that decision, the appropriate level of difficulty could be chosen and the patient would use the program to track his recovery.

Apart from all the formal requirements that were met, the greatest achievement of our game is probably that it is pleasant and challenging at the same time. The users were both excited and interested in testing it, which makes it a success.

9.1 Future Work

There are a number of different and useful improvements that could be made to the program that was created. Here are some of them:

9.1.1 Cosmetics

Our program could refer not only on apraxia patients but on people that recover from an upper-limb surgery. That would mean that young children could use it. Which means that they would be more interested in an even more game-like environment. Our application already has the functionality of a game, so it can be extended to have more game features, such as points, sounds or different background images. The younger users would be more excited if they could enjoy the positive features of a game while they perform their daily exercises.

An improvement that could be made is if the user's progress wasn't saved in txt files in some folder but was part of the game. Which means that the user could load his previous scores by pressing a button within the program. His scores would be loaded inside the game, so their manipulation could be part of the game as well. The line graph could be an option through the game. The user would have the option to see his previous scores and progress or continue with his exercises, and as it is happening now, that when the user shuts down the program the folders containing the results are created, the results would be saved inside the game in a non-readable form.

Some may argue that knowing only the high score of a gesture is not enough in the long run. That the user should know how well he did on a single effort overall and not at a single moment. The way that this issue could be tackled is by counting the time that the user's score was above a certain threshold. For example the results could say that his high score was 98.3% and he managed to stay above 90% for 36 seconds out of 60 seconds. That would require that the time for each task was fixed.

An initial idea for this project was to add some "intermediate" gestures for the patients. These gestures would start with an open palm and continue with the fingers slightly bending until they reach the original goal gesture. The purpose of this would be that the user would try an easier gesture than the final one so he would be able to familiarise himself with the gestures more easily.

9.1.2 Functionality

Additional scoring parameters could be added. In the *Angle Scoring Function* for example, the angles between the user's fingers could also be taken into account, or the rotation of the wrist. For the creation of a better scoring function, a deeper understanding of the anatomy of human hand is required. The tendons and the nerves that exist in the human hand should be taken under consideration and depending on their importance and the hardness of manipulating in a specific way each finger, a final score would be produced which would be much more objective and scientifically documented.

Based on the previous statement, there could be some distinction between the gestures. The results show that some gestures are genuinely more difficult than others. So, for those gestures, the scoring could be less strict, or some additional parameters could be introduced so that the final score would be higher.

9.1.3 Applications

Finally, a major improvement for the game would be if it was created to work with the use of Microsoft HoloLens [8]. Windows Holographic is a mixed reality computing platform, enabling applications in which the live presentation of physical real-world elements is incorporated with that of virtual elements such that they are perceived to exist together in a shared environment.

During the Build 2015 keynote, it was revealed that Unity 3D is partnering with Microsoft on HoloLens development, and the development team have recently confirmed on their blog that HoloLens will enjoy full support from the Unity engine. Which means that the game created could be massively upgraded using this exciting technology. The user could perform a gesture wherever he wanted without the use of a screen, he could perform a task whenever he went to the kitchen, in front of the kitchen door. The possibilities of the use of a platform such as Unity and the technology of Microsoft HoloLens would be countless.

9.2 Personal Note

For the creation of this program, I had to put into practice various concepts from different courses that I attended this year. *Computer Animation* was the most useful one. The concept of vector was extensively used to solve different problems in the program. For example, when we had to adjust the goal gesture to the user's hand. In the practicals, I became familiar with the way *Maya* works. The knowledge gained from there was put in practice because many aspects of *Unity* are similar to the ones in *Maya*, for example the creation and manipulation of objects.

The concept of *semaphores* that was introduced to the course *Concurrent Programming* helped me a lot to control the flow of my program, especially since the function Update( ) was called once per frame and the function OnGUI( ) was called multiple times per frame.

Finally, the precious hours that were devoted on the practical assignments of all my courses gave me the experience and knowledge that I needed to complete my program.

During these last few months I gained valuable experience and learned numerous lessons. Firstly, I learned how to organise my code. I had to improve my program several times for example, originally I thought it would be easier to store the user spheres in different arrays, one for each finger, but later I realised that there were many redundant commands and a lot of repetition in my code. So I had to improve my code and store the spheres into a single array.

I also learned the importance of testing the program in all the possible ways. There were countless bugs in my game, for example when I implemented the way to count the time that the user needed to complete his task, or the time he needed to achieve his best score when he tried the same task more than once before he shuts down the game. Or to deal with the case when the user has chosen a left hand gesture and he had the right hand above the leap etc.

One of the greatest lessons was that I familiarised myself with new technologies, as well as a new programming language, and I learned how to read and use their documentation.

I was excited about this project from the beginning, and now that it has come to an end, I am really pleased with the outcome. All the goals have been achieved, the environment that was created is great, and the people that used the program were pleased as well. Having the opportunity to create a game which could be used from patients and help them improve their condition was a pleasant and inspirational experience.

Acknowledgements

I would like to use this opportunity to thank the people that helped me complete this project and those who assisted me throughout the year.

Firstly, I would like to thank my amazing supervisor Irina Voiculescu. Your comments and suggestions were always helpful. Thank you for all the time that you spent with me and for believing in my skills when I doubted them.

Then, I would like to thank Prof Glyn Humphreys and Dr Mihaela Duta for their feedback and of course all those who tested my program and happily offered their time and comments. The game would not have been the same without you.

I would also like to thank all of my co-students who assisted me this year. Thank you all for helping me adapt to the field of Computer Science. Moreover, all the wonderful and interesting people that I met here in Oxford. You all made my life here more beautiful.

Next, I would like to thank all my friends back in Greece for being there for me whenever I needed them. I would like to thank specifically Fotis and Aleka for their help and especially Vasilis Diakomanolis for all the long hours that he spent explaining to me Computer Science concepts and assisting me this year. Thank you for making my life a lot easier my friends.

Additionally, I would like to thank my family for their support. Thank you for having faith in me from the first moment. Lastly, I would like to thank my partner Markella. You were always my light in this year's darkest hours. Ευχαριστώ.

References

[1] Michael Gorman, Leap Motion controller review, July 2013.
http://www.engadget.com/2013/07/22/leap-motion-controller-review/

[2] Unity Technologies.
https://unity3d.com/unity

[3] Unity Technologies, Official webpage.
https://unity3d.com/

[4] Leap Motion – Unity3D.
https://developer.leapmotion.com/gallery/category/unity3d

[5] Leap Motion – C# Skeletal Tracking Model.
https://developer.leapmotion.com/documentation/csharp/devguide/Intro_Skeleton_API.html

[6] Leap Motion SDK and Plugin Documentation.
https://developer.leapmotion.com/documentation/index.html?proglang=current

[7] Oracle Documentation – Using JavaFX Charts, January 2014.
http://docs.oracle.com/javafx/2/charts/line-chart.htm

[8] Microsoft – Microsoft HoloLens.
https://www.microsoft.com/microsoft-hololens/en-us

[9] World heart federation – Stroke.
http://www.world-heart-federation.org/cardiovascular-health/stroke/

[10] NHS Stroke – NHS Choices, September 2014.
http://www.nhs.uk/conditions/Stroke/Pages/Introduction.aspx

[11] Microsoft – Kinect for Windows.
https://www.microsoft.com/en-us/kinectforwindows/

[12] Nintendo – Wii.
https://www.nintendo.co.uk/Wii/Wii-94559.html

[13] Playstation, Official webpage.
https://www.playstation.com/en-gb/

[14] Piotr Kozlowski, Measuring gesture mimicking accuracy, University of Oxford, 2013.

[15] Kamil Chmurzynski, Assessment of hand gestures, University of Oxford, 2013.

[16] Emil Culic, Objective hand gesture scoring for apraxia patients using Leap Motion, University of Oxford, 2014.

[17] Unreal Engine.
https://www.unrealengine.com/