E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Comparing tangible and mid-air interfaces for interacting with and 3D audio controller.

Grado en Ingeniería Informática

Trabajo Fin de Grado

Fernando Merino Pinedo

Supervisores: Asier Marzo Pérez, Rubén Eguinoa

Pamplona, 7 de enero de 2022

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Content

ACKNOWLEDGEMENTS

This report sums up a 2-year work not only full of efforts and dedication but also full of collaboration with many different people that have made possible to me to carry with the development of this project.

I would like to thank all of them, starting with the director of this project, the professor Asier Marzo Pérez, who has been there to help me whenever I needed it, provide me with all the stuff I've needed, share all his knowledge and propose me his crazy ideas that in the end we have carried out.

I would like to express my gratitude too to all the three engineers from the Acoustics Lab of Upna: Ricardo, Asier and Ruben, for helping and collaborating with me in such a wonderful way that has allowed to develop an incredible product which combines acoustics and informatics.

Finally, I would like to thank of course my family and friends who helped me both making the journey more enjoyable and resolving with me some problems I encountered in the development.

ABSTRACT

3D Audio is becoming a new standard in society for music listening and general audio. New applications emerge every day trying to build each time more immersive experiences: videogames, environments simulations, virtual reality… where audio is a crucial part to reach this immersion.

The department of Acoustics developed a 24-speakers metal sphere called JAULAB where users can go in and experiment real 3D audio. But we wanted more. Nowadays applications highlight by its user-machine interaction and freedom of action. That's why, in a collaboration between the Acoustics Lab and UpnaLab, we would like to give users the opportunity to play and interact with the sphere to offer a much more immersive experience that could be used in the future for a great variety of applications (environment simulations, virtual reality, music production, music exhibition…)

For this purpose, an implementation of two different interactive interfaces has been developed to be integrated in the sphere. The project has been divided in four different parts: the tangible interface implementation, the mid-air interface implementation, the sphere integration and finally the user study. We will be seeing all this parts during this report.

KEY WORDS

- Spatial sound/3D sound
- Interactivity
- Aruco markers
- Camera world coordinates
- Leap Motion / Ultraleap
- Focal amplitude
- Docking task

INTRODUCTION AND MOTIVATION

After vision, hearing is probably the most important human sense, and modern software applications know this and make use of it. Till the date, the most immersive applications interact through the eyes, ears and they are starting now with touch, but is clear that hearing is a crucial part. That's why that, when trying to imagine future virtual reality, we imagine something much better and futuristic than just current normal headphones.

JAULAB was conceived with this idea, to develop a new way of hearing sounds, of hearing 3D audio, regardless of its application. And I say regardless of its application cause JAULAB can be used for multiple applications.

JAULAB was initially designed for creativity and learning purposes. The idea was for it to be an environment where producers and artists can reproduce their 3D music compositions, in a spatial sound prepared environment, enjoying a true 3D experience and getting rid of the 2D headphones which offer a poor 3D spatial sound simulation. At the same time, it would serve as a learning resource for new acoustic engineers, that would use the sphere as a tool to practice with spatial sound.

Image 1: Image of the JAULAB sphere already assembled with the 24 speakers.

This is how the Acoustic Lab presented me the project. Although it was in an early version, thanks to my passion for music, for which I even produce my own music as a hobby, I really loved the project and decided to take it.

The main problem was the poor interactivity of the sphere. For the initial purposes it was created, it was enough with that, you just create your compositions in some software and then try them inside the sphere and enjoy the results. But the lack of interactivity closes the door to any other kind of application. This is where the need to implement an interactive interface arises.

Interactivity allows the development of a lot of new different applications. By introducing this concept in the sphere, we can develop several new functionalities like in-runtime music producing, dynamic nature environments, videogames… So, it seems crucial to introduce it to the sphere.

The main idea was to implement an interface as an initial test for the interactivity we can reach with the sphere. It would be an interface that would power up the basic functionalities of the sphere and an interface that would made the sphere much more attractive and family-friendly to use for non-specialized users nor with acoustics nor with the sphere.

We thought of a single and simple-to-use interface, where people would play with some tangible physical tokens representing the sound sources played inside the sphere. But then Covid-19 came, and non-touch interfaces became a new standard in society.

Therefore, we started thinking of developing a new complementary interface totally Covid free where users could be able to have the total control over the sphere without touching anything.

We will be seeing these two interfaces and its development along this report.

OBJECTIVES

The main objective of this project is to implement two different interfaces capable of interacting with the sphere and giving us total control over the spatial sound sources reproduced inside the sphere.

Once the two interfaces are finished and fully operative, a user study will be carried in order to determine which one of them two is easier to use and likes most to users.

For these purposes, the following objectives must be achieved:

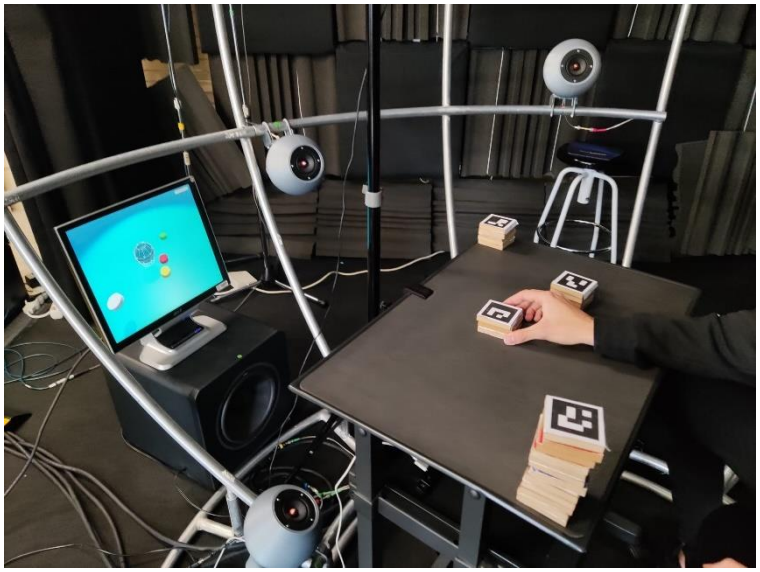
1. Design the two different interfaces and its functioning.
2. Analyze and understand the necessary technologies for the interface's implementation.
3. Implement the first interface, the tangible one.
4. Implement the second interface, the mid-air one.
5. Integrate the two interfaces with the sphere software
6. Carry out the user study with volunteers
7. Analyze user study results and make conclusions

DEVELOPMENT

TWO INTERFACES

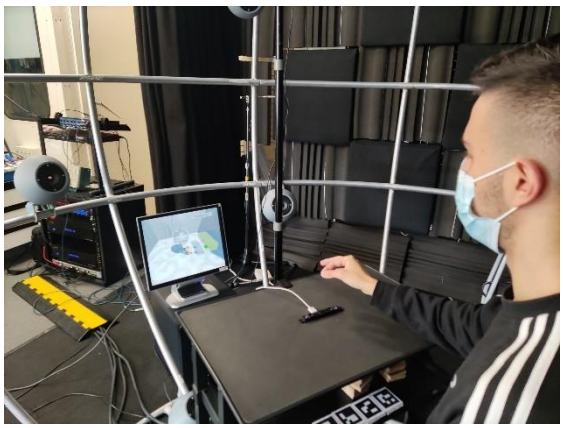
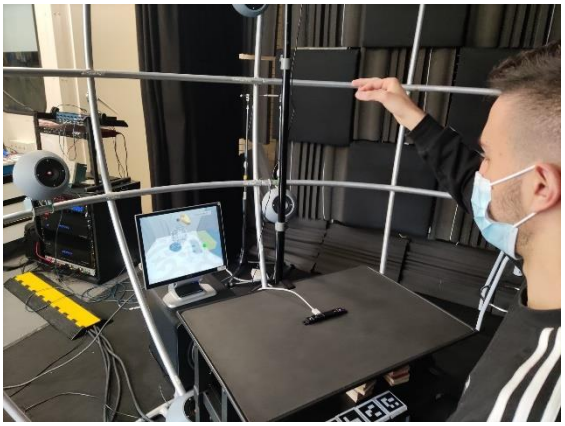
As mentioned before, there will be two interfaces different from each other.

The first one will be a tangible interface where there will be some physical tokens representing the sound sources that will be played inside the sphere and with which the user will be allowed to play and move them, having total control over the sphere.



Images 2 & 3: Image showing volunteers using the tangible interface.

The second interface will be the mid-air one. The user will use only his hands and without touching and just by doing some gestures and movements in the air, he will be able to control the sound sources and move them all over the sphere.

Images 4 & 5: Image showing volunteers using the mid-air interface.

Both interfaces will have visual guidance for the user to better orientate. The tangible version will have a limited workspace where the user can move the tokens and inside this workspace there will be a mock-up representing the sphere so that the user is better oriented. The mid-air version will count with a virtual simulation made in Unity so that the user can see in a screen the scene, his hands and where the sound sources are.

These two interfaces will be further explained within its corresponding sections, but before we have to understand how the sphere works.

JAULAB

Before starting with the interfaces' implementation, it's good to understand well what JAULAB is.

JAULAB is a metal sphere built with 24-speakers covering uniformly the whole spherical area. These 24-speakers work coordinately thanks to the Spherical Wavelet Format (SWF) a newly developed spatial audio format which replaces the Ambisonics' spherical harmonics by an alternative set of functions with compact support (spherical wavelets). It also covers all the steps of audio production, with a complete audio chain from encoding to decoding based on the discrete spherical wavelets built on a multiresolution mesh.

Image 6: One of the 24 speakers that are part of the

For no acoustics experts, what makes this technology is to discretize the space in such a way that the coordinated speakers can reproduce sounds in any point. The sound can be reproduced in points where there is a speaker, but also virtual sounds can be generated in points in the middle of the empty air where no speaker is placed. This is thanks to this discretization of the space where the 24 speakers emit sound in a coordinate way such that virtual sounds can be generated anywhere in the workspace. The bigger the discretization is, virtual sound can be generated in a bigger number of points and with more precision.

An important fact to note before continuing is that JAULAB is designed to simulate sounds that seem to be played outside the sphere. This means the following:

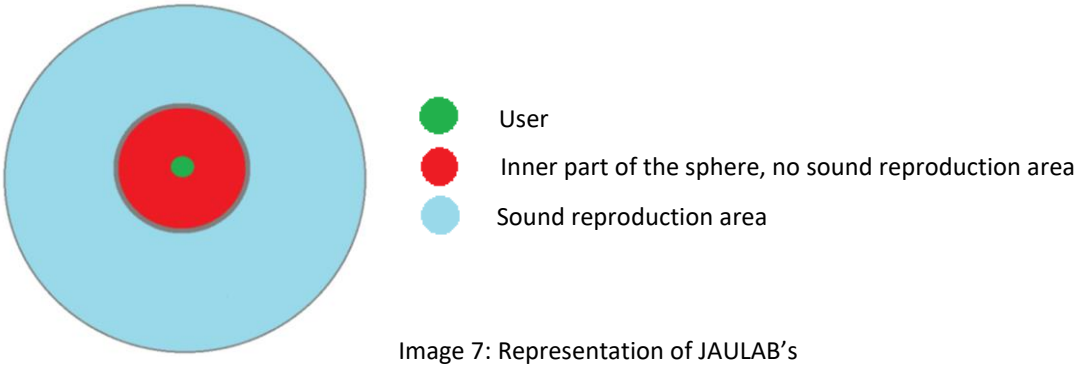


Image 7: Representation of JAULAB's working

The inner red circle represents the sphere, the green circle is the user, and the blue circle is the area where virtual sounds can be simulated. The user is inside the sphere and the speakers are oriented towards the center of the sphere. However, only sounds that simulate to be playing outside the sphere, in the blue zone, can be generated.

This is because the sphere is supposed to generate sounds that surround us, giving a spatial sound experience and generating sounds inside the sphere would lead to some bad functioning due to technical reasons that would ruin the experience.

This is important to take into account as it will affect later our interfaces.

Tangible interface

The tangible interface will consist of a sphere mock-up representing the real sphere and a set of physical tokens that will represent the sound sources. With this and limiting a workspace to work, the user will be able to place the tokens around the sphere mockup in such a way that each of these tokens will represent a real sound source that will be played on the sphere on real time.

For this interface to work, multiple technologies are involved. The process is complex as explained briefly below.

A camera focusing on the workspace will be used to detect the tokens and capture its position in real world. This camera must be calibrated before starting with the detection. The tokens will have some special markers called Arucos on the top of them to facilitate its detection and position estimation. Once the position is captured, several matrixial transformations must be carried out in order to get the appropriate coordinates. Finally, with the coordinates correctly processed, they can be sent to the sphere using some specific way of communication and once there the sphere software will be able to reproduce the sounds in the correct position inside the sphere.

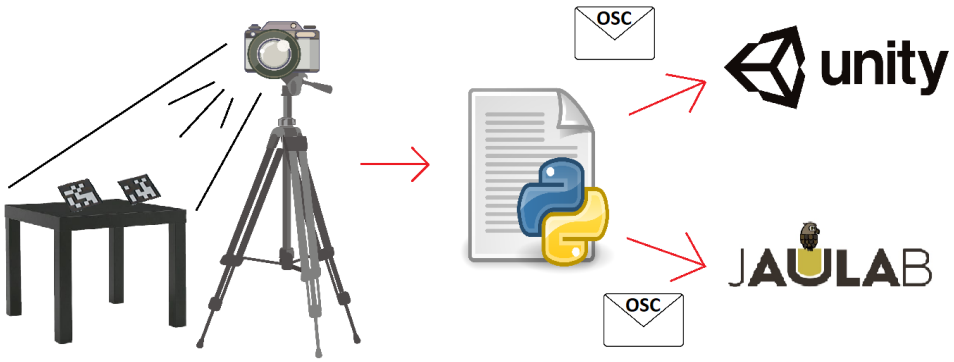


Image 8: Diagram showing the process flow for the tangible interface

*NOTE: The Unity process showed in the diagram above refers to a 3D Unity simulation whose need is explained later in the report.

All this process must be carried out on real time, so that we offer the user a fluid experience where as soon as he moves a token on the workspace that same move is represented in the corresponding sound source moving around the sphere.

Now we will get on detail on all these points the process has.

Camera calibration

The cameras used in the tokens' detection introduce distortion to the images captured, especially radial and tangential distortion.

Radial distortion curves the image as we move away from the center, making straight lines no longer be so straight. Tangential distortion occurs because the camera lenses are not aligned perfectly parallel to the plane of the object to be captured.

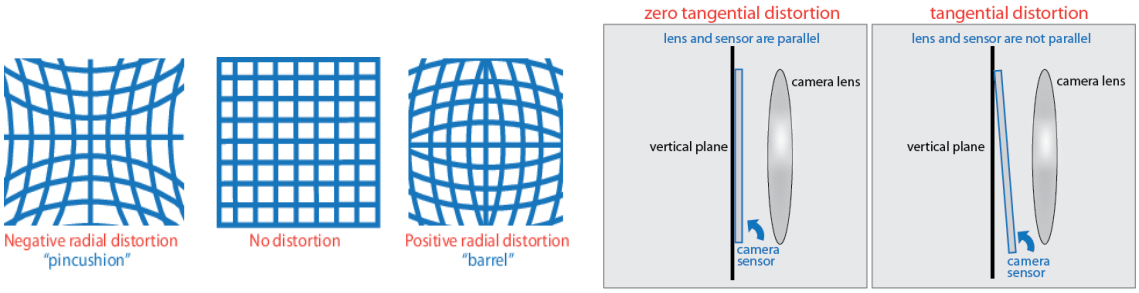


Image 9 & 10: Images showing the radian and tangential distortion problem

These two problems together produce distortions that, in the subsequent processing of the images and obtaining of the estimated positions of the tokens, these will differ from the real position where the user has placed the token.

Therefore, camera calibration is necessary and almost mandatory to try to counteract and reduce as much as possible the distortions.

To calibrate the camera, the OpenCV library has been used, which offers us a series of functions and methods capable of easily constructing an algorithm with which to obtain the parameters of the camera used and perform the necessary operations to calibrate it.

This algorithm consists of taking with a well-fixed camera a series of images of a chessboard and observe how distortion affects the chessboard in different positions and orientations as we can see in the images below.

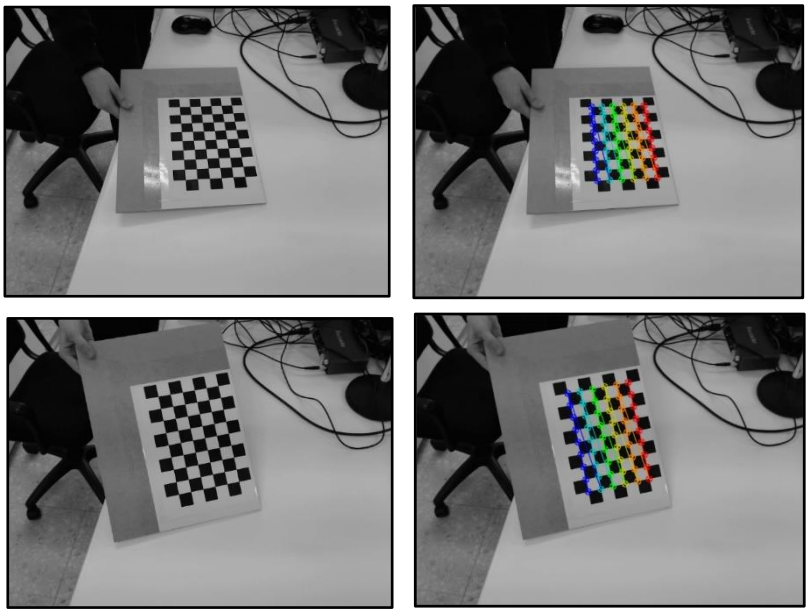


Image 11: Chessboard calibration algorithm images

I don't want to explain further how this algorithm works, as it is all done by the functions of the OpenCV library, but in summary, by drawing distortion lines on the chessboard and seeing how they change between the different images we get the necessary parameters with which later will be performed a series of operations used to calibrate the camera. For more detailed information on how this process works, see at OpenCV "Camera Calibration Tutorials"[1].

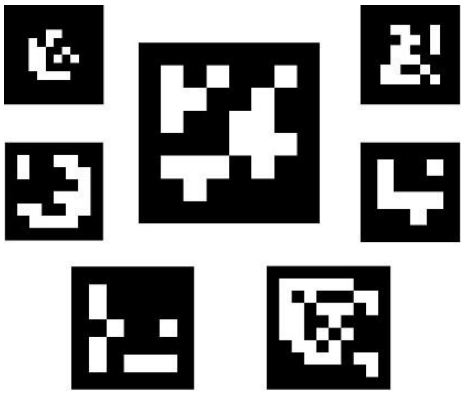
Tokens detection

With the camera well calibrated, next step is to detect tokens in the real world for in further steps estimate its virtual position that we will pass to the sphere.

Tokens detection process is also carried out using the OpenCV library plus the addition of an auxiliary module called Aruco.

Aruco is a module developed for OpenCV which consist of a series of operations capable of detecting its Arucos markers in a very easy and efficient way.

The Aruco markers are a series of binary square markers (white and black) with the main benefit that its detection is very simple and fast. Therefore, they are a perfect option to estimate the position of the tokens in real time.



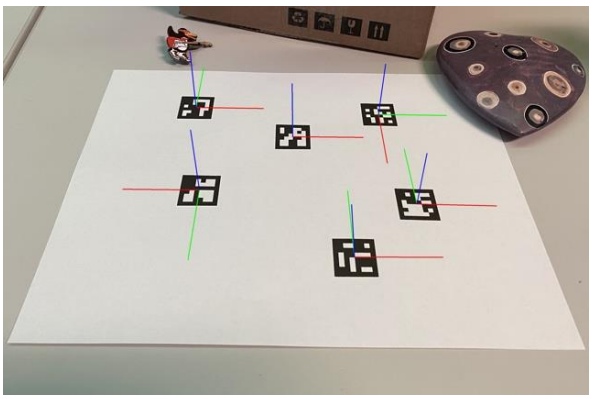


Image 12 & 13: Some examples of Aruco markers.

The functions the library offers are very powerful and just by using a couple of them we can already not only detect the markers but also get its camera world coordinates. However, before doing this, an initial setup has to be made.

First, we must indicate which Arucos dictionary we are using. There are a lot of Arucos dictionaries that can be used, all of them differing in two things: the number of bits of each marker and the number of markers the dictionary contains. The maximum number of sounds source the sphere can reproduce at the same time is 24, so for this project, as we will need as much 24 tokens, the best idea is to select a dictionary close this number. The smallest dictionary is DICT_4x4_50 (4x4 bits and 50 markers). With 50 markers is enough to cover all our sound sources, so it's logic to take the simplest dictionary to gain in efficiency. This is the dictionary I'm using.

After that, the marker size has to be setup too. The marker size is a really important parameter. We are using only one camera to detect the markers that will be placed above the workspace. With only one camera, we are only able to measure two dimensions, height and width, but we are not able to measure the depth. For measuring the depth, another camera should be placed tracking this dimension. But this would make the process really complex and slow. By indicating the markers size (length of one side) we are now able to detect how much the size change when moving it in the third dimension, the depth, so with this variation of the size, we are capable of estimating its variation in the depth. This depth variation will be later calculated in the coordinates processing.

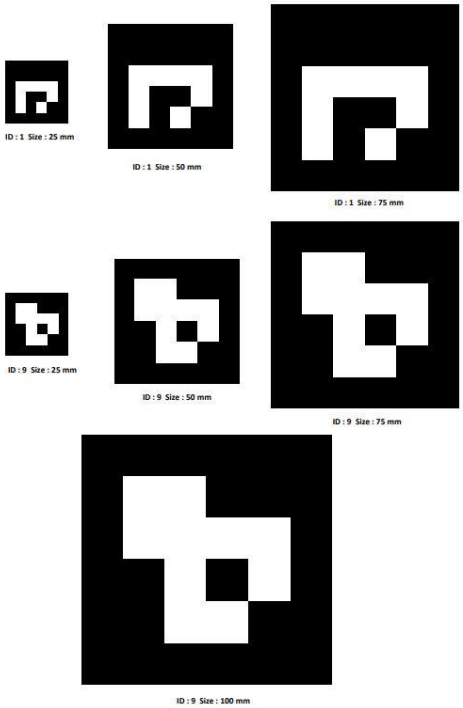


Image 14: Set of Arucos used during the development with different sizes.

At the end, the sized chosen for our Aruco markers was 50mm long, a square 5x5cm.

Coming back to the Aruco's functions, they return us the camera world coordinates of the markers through the main functions of detectMarkers() and estimatePoseSingleMarkers(), functions that I don't want to explain here cause it would take too much but that you can see at "Aruco Marker Detection"[2] and "Detection of Aruco Markers"[3] tutorials for further information. Apart from the coordinates, the functions also return a unique identifier which identifies each marker.

The OpenCV libraries return us a set of coordinates for each marker, but these coordinates are nothing more than just numbers without the correct post-processing.

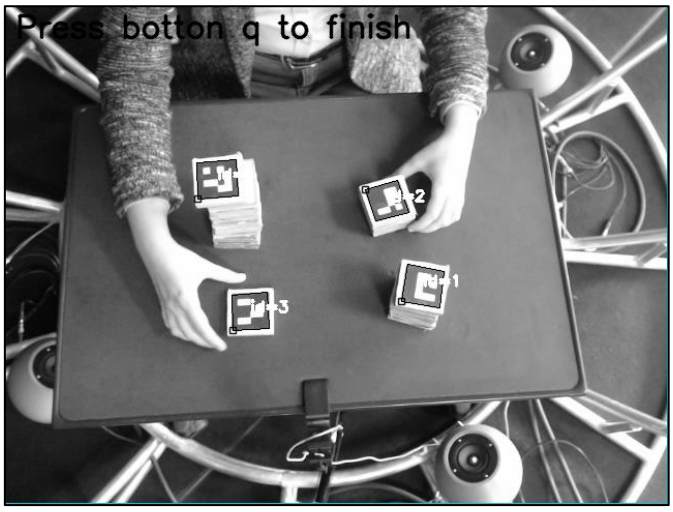


Image 15: Tokens detection during a test with real users.

Coordinates processing

Once the camera detects the Arucos, the detection functions return a series of coordinates that without the correct processing have no value at all. They must pass through different transformations processes.

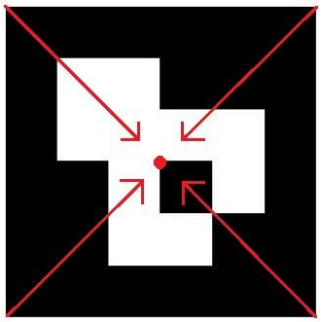
The first problem with the coordinates has already been commented. The coordinates returned are relative to the camera world in a 2-axis coordinates system (x, y), they are not world absolute coordinates with the three cartesian axis (x, y, z). This must be changed as we need also the third axis, the depth. We already know the solution. The idea is simple, if the marker approaches to the camera, it will become bigger. If the marker moves away from the camera, it will become each time smaller and smaller. Consequently, by controlling how much the size of each marker variates, through some complex mathematical operations, we are able to calculate the coordinates in the third axis z.

```
mhalf = markersize / 2.0

mrv, jacobian = cv2.Rodrigues(rvec)

#in markerworld the corners are all in the xy-plane so z is zero at first
X = mhalf * mrv[:,0] #rotate the x = mhalf
Y = mhalf * mrv[:,1] #rotate the y = mhalf
minusX = X * (-1)
minusY = Y * (-1)

# calculate 4 corners of the marker in camworld. corners are enumerated clockwise
markercorners = []
markercorners.append(np.add(minusX, Y)) #was upper left in markerworld
markercorners.append(np.add(X, Y)) #was upper right in markerworld
markercorners.append(np.add( X, minusY)) #was lower right in markerworld
markercorners.append(np.add(minusX, minusY)) #was lower left in markerworld
# if tvec given, move all by tvec
if tvec is not None:
    C = tvec #center of marker in camworld
    for i, mc in enumerate(markercorners):
        markercorners[i] = np.add(C,mc) #add tvec to each corner
# print('Vec X, Y, C, dot(X,Y)', X,Y,C, np.dot(X,Y)) # just for debug
markercorners = np.array(markercorners,dtype=np.float32) # type needed when used as input to cv2
return markercorners, mrv
```

Image 16: Code showing the process to transform 2D coordinates to 3D.

After resolving the camera world coordinates, we get a three-axis (x, y, z) coordinates for each corner of detected markers. This means that for each marker we get a 4x3 matrix with the cartesian coordinates for each corner of the marker. As we are working with the idea of moving one unit of sound source it is not logic to work with 4 coordinates for each marker. The sound source must be played at a specific point with just one measure, so it does make sense to transform these four coordinates to one single one. Thus, the four coordinates are transformed using a mean to one single coordinate. As we are using a mean, the coordinates we get are the corresponding one

to the center of the Aruco Marker. This is really good for us because these coordinates of the center of the marker will represent the exact point where the sound source will be reproduced inside the sphere. Such that we get a single point that will facilitate the work in great way rather than having four different measures for each corner.



```
final_corners = np.mean(corners_3D[i], axis=0)[0]
final_corners = np.matrix([final_corners[0],
                           final_corners[1],
                           final_corners[2]])
```

Image 17 & 18: Representation of the reduction to only one point by the mean and its code.

After applying the first two transformations, we obtain a series of coordinates that are correctly correlated with the positions of our tokens in the real world, but that are not adjusted to the coordinate range of our workspace. Therefore, one more transformation is necessary to map the coordinates to the workspace coordinates we want to work with. For this, it is necessary to calculate a matrix M that will serve as a change of basis matrix. This matrix is calculated with the function from the OpenCV library estimateAffine3D(), which requires the initial workspace, the target workspace and some other inputs, and returns different parameters among them the change of basis matrix M. So, this matrix is obtained by mapping the points of the initial workspace to the points of the new workspace we have defined. The matrix must be saved as it will be needed later to map the coordinates of all markers detected on run time.

```
MyAffine=np.array([[0, 0, 0, 0],
                   [0, 0, 0, 0],
                   [0, 0, 0, 0]])
MyInLi=np.array([7,3,7])

#Corner points of workspace in Unity simulation
target = np.array([[-0.8,0.8,-0.2],[0.8,0.8,-0.2],[0.8,-0.8,-0.2],[-0.8,-0.8,-0.2],
                   [-0.8,0.8,0.2],[0.8,0.8,0.2],[0.8,-0.8,0.2],[-0.8,-0.8,0.2]])
a,M,c = cv2.estimateAffine3D(src=calibrations,dst=target,
                             out=MyAffine,inliers=MyInLi)
```

```
calibrated_corners = calibrated_corners*k + (1-k)*np.array((M * np.vstack((final_corners.reshape(3, 1), 1)))[0:3, :].reshape(1, 3))[0]
```

Image 19 & 20: Code to transform coordinates to the range of our workspace.

Finally, one last transformation is necessary. JAULAB sphere software works with polar coordinates (ρ, θ, ϕ) and the coordinates we have at this point are cartesian (x, y, z), thus, it is necessary to perform this last transformation of cartesian coordinates to polar coordinates so that our control program and the sphere program understand between each other. Through a series of mathematical operations this transformation takes place.

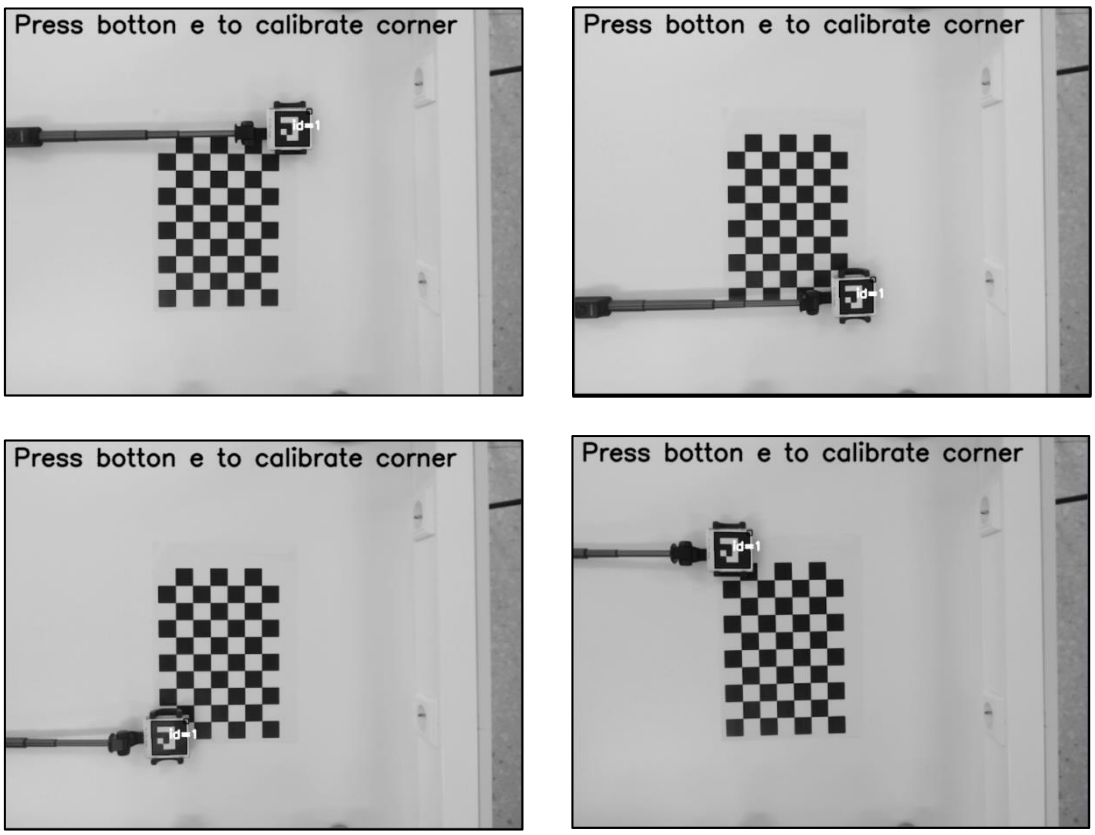
With all these transformations performed, the coordinates are ready to be sent to the sphere. But before that a process necessary for the coordinates processing must be pointed out, the workspace setup.

Workspace setup

As mentioned before, one of the necessary transformations we have to apply to the coordinates is the mapping to the desired workspace using a change of basis matrix M.

To obtain this matrix we need to have the target workspace, that could be whatever we want, and the initial workspace, that can change depending on where we are physically working: a bigger or smaller table, on the ground, in a stand… Is because of this that we need a system to set up and configure the workspace we are working in. And a very simple system was created for this.

The system consists of setting up only the ground of the workspace. Using an Aruco marker, the use will define the four corners of its workspace. The user has to place the marker on each of the four corners of the workspace and capture the image by pressing "e" as shown on the image below.



Images 21: Images showing the process
to calibrate the workspace.

*NOTE: The chessboard paper is not necessary for workspace setup, it was there just as guide to limit my workspace in that moment.

By doing this the user is setting up the ground of its workspace, the x and y axis. The z axis is automatically set by replicating these four points but adding to them a specific height previously determined in the code. This predetermine height can be changed in the code, but the one by default is supposed to work fine for a standard workspace.

With these 8 points calculated (the four set by the user and the four set automatically by adding a height) a 3D cube is defined that will be the target workspace. This new workspace is the one that will be used later as input to estimate the change of basis matrix.

Communication with the sphere

To conclude the interface, it remains to communicate to the sphere the coordinates where it has to emit each sound source.

For this purpose, the communications with the sphere will be carried out through OSC packages using the PythonOSC library.

The functioning of OSC packages is very simple and effective. They are packages containing string-only data from a client to a specific IP and port where the server will be hearing.

Our interface acts as the client sending for each frame captured by the camera the OSC packages to a previously specified IP address and port. For each marker detected, it will send a package containing a string with the coordinates and identifier information.

Then the sphere acts as the server. Hearing in the corresponding IP and port it will receive the packages sent by the interface. From them, the sphere will extract the coordinates of each Aruco and its ID. The ID will be associated with a sound source and the coordinates will be used to broadcast the sound in the right place inside the sphere.

Mid-air interface

This second mid-air interface is much simple than the first one.

The mid-air interface consists of a Leap Motion camera who will be tracking all our movements and gestures we do with our hands and will perform some operations over the sphere depending on these gestures.

This interface involves much less technologies than the tangible one, only three: a Leap Motion camera, a Unity simulation and the sending of OSC packages. These three technologies are combined to work as following:

The Leap Motion camera will be focusing on an empty air space. We must consider that the Leap Motion camera it's not a normal camera, as the ones we used in the tangible interface, but it is an infrared camera. This camera will be working and tracking all our hand movements inside this empty space which will be our workspace.

Parallel, a Unity simulation is developed containing an environment representing the sphere and the sound sources, the equivalent to the mock-up of the first interface but this time it is a virtual mock-up. This Unity will contain some components and visualizer which allows to draw the hands tracked by the Leap Motion camera inside the Unity simulation, as shown below.

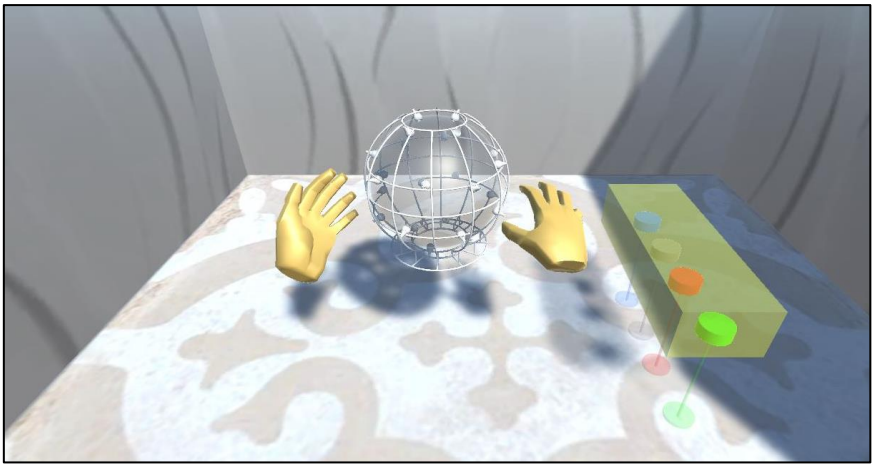


Image 22: Image showing the hand tracked inside the Unity simulation

This Unity simulation will serve as a visual guide, where the user will see how his movements are tracked and will also see the sound sources we can take and move. With

some code and scripts, the user will be able to grab the virtual sound sources doing some specific gestures and then move them all around the environment. The sphere will be on the center of the simulation and we will be moving the sound sources around it. At the same time, OSC packages are being constantly sent with the updated positions of each virtual sound source to the sphere.

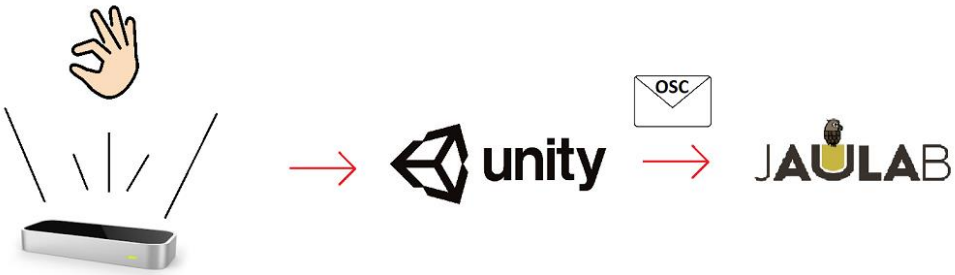


Image 23: Diagram showing the process flow for the mid-air interface

Some other later explained functionalities will be added to expand this interface functioning.

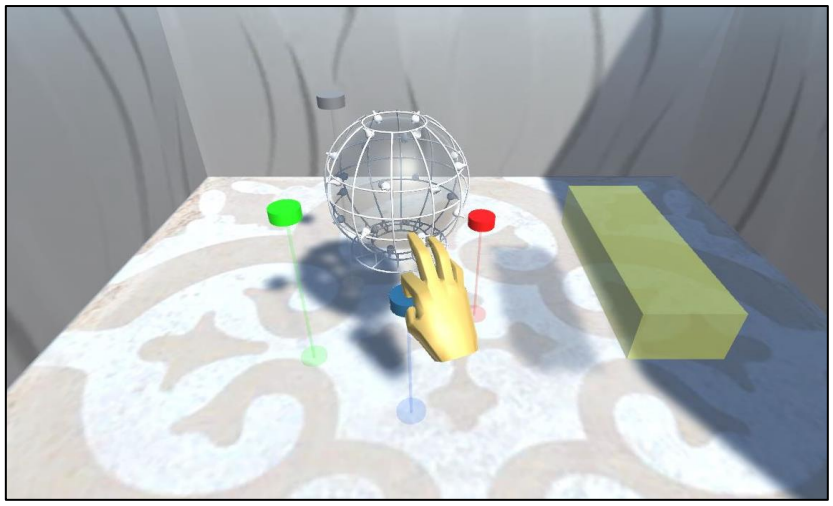


Image 24: Image showing a user grabbing sound sources and moving

Hand tracking

Hand tracking functionality inside our environment is divide in two parts: movement tracking and pinching.

The hands movement is just the tracking of the user's hands and arms along the whole workspace. It is automatically done by the Leap Motion camera, we just have to install the necessary Leap Motion packages, connect it to the computer and the camera will automatically start tracking our hands.

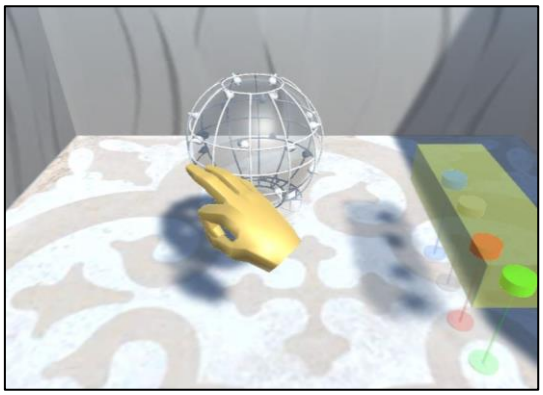
With the hands' movement tracked, we now need gestures to interact with the environment. Our interface consists of grabbing the virtual sound sources represented inside the Unity scene and move them all around the sphere. Thus, we need a gesture to grab the sound sources, move them and release them. The most logical thing is to configure a gesture to grab the objects as similar as possible to the gestures we do when grabbing things in normal life. Hence, I decided to configure a pinch as the gesture to grab the sound sources.

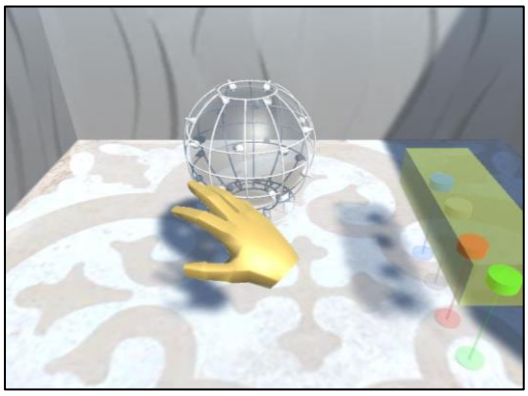
Image 25: Code to grab and move the sounds sources by pinching.

```
// Update is called once per frame
Mensaje de Unity | 0 referencias
void Update()
{
    tokens.Clear();
    minimumDistance = 5f;
    frame = controller.Frame();
    hands = frame.Hands;
    pinch = 0;
    if (frame.Hands.Count > 0)
    {
        fristHand = hands[0];
        pinch = fristHand.PinchStrength;

        foreach (GameObject token in GameObject.FindGameObjectsWithTag("Token"))
        {
            distance = Vector3.Distance(pivote.transform.position, token.transform.position);
            if (distance < distanceThreshold && distance <= minimumDistance)
            {
                minimumDistance = distance;
                tokenSelected = token;
            }

        }

        if(tokenSelected != null)
        {
            if (pinch > 0.9f)
            {
                tokenSelected.transform.position = pivote.transform.position;
            }
            else if (pinch < 0.1)
            {
                tokenSelected.transform.parent = null;
                tokenSelected.transform.position = tokenSelected.transform.position;
                tokenSelected = null;
            }
        }
    }
}
```

Pinching is quite simple to control with Leap Motion. As shown in the code above, we just initialize a new controller that, frame by frame, will check how near the index finger and the thumb are. The closer they are, the stronger the pinch. The pinch gets a value between 0 and 1 depending on this, 0 if the hand is fully open and 1 if the two fingers

are touching. This way, stablishing a threshold, in my case of 0.9, we can determine whether the user is pinching or not.





Images 26 & 27: Images showing a player pinching (> 0.9) and not pinching (< 0.9).

The user will get visual guidance of all this process of pinching and sounds source movement through the unity environment as we will see now.

*Important note: The pinching only works for the right hand.

Unity environment

In this interface, the user plays mid-air in a totally empty space area without anything to touch or to use to get some guidance. This can lead to great difficulty to use the interface due to a lack of orientation producing a sense of frustration on the user. Thus, a visual guidance is necessary. Additionally, we need some 3D environment which receives the Leap Motion tracking inputs and process them.

As we have said before, Unity fits perfectly for this. Unity already has its own components and packages to implement all the Leap Motion functionalities and, also very important, visualization.

Representing the sphere environment inside Unity gives the user a great visual guidance with which he can orientate perfectly. Unity allows us to draw the sphere, the sound sources and the hands tracked movement in real time, so we can build a complete virtual environment to be used as our workspace. The hand movements the user makes are represented in the scene and this includes pinching too. So, when the user pinches, he sees how the two fingers move exactly as he has pinched, and he knows that what he is doing is working.

Only one thing left. The user must see also how the sound sources get attached to his hands when he is pinching, and they move with him. So, when we detect the pinch, the closest sound source to the hand gets automatically attached to the point of union between the thumb and index finger and doesn't detach until the pinch ends. Like this, the user sees how the sound sources move with him when he pinches, getting all the visual guidance he needs.

Coordinates processing

At this point, we have the hand tracking and the unity environment, but the most important part lefts, the coordinates processing.

Somehow, we need to know how the movements we are doing inside Unity must be reproduced inside the sphere. For this, a simple solution was found.

The Unity virtual sphere would work exactly as the JAULAB sphere. The center of the sphere will represent the point where the user seats inside the sphere. This center will be exactly set at the point 0,0,0.

Then we will have the sound sources, that will be floating around the sphere. The user will be able to move them around the scene and its position relative to the center of the sphere, the 0,0,0 point, will represent the coordinates that we will be sending later to the sphere.

Better explained, if inside the Unity environment we move a sound source to a position 2 units away from the center of the sphere, in the JAULAB sphere we will reproduce this same movement by playing the sound 2 units away from the user multiplied by a factor of adjustment. This factor of adjustment it's just because the range in which the JAULAB works it's different from the range in which the Unity environment is designed.

So, the coordinates processing is carried out by constantly sending to the sphere the virtual positions of sound sources. A last transformation is needed because the coordinates we get from Unity are in cartesian coordinates and the JAULAB needs polar coordinates. For this transformation, the exact same process as in the tangible interface is done.

OSC sending

The communication between the Unity environment and the JAULAB is also the same as the tangible interface one. The only difference is the library used. Previously we used the PythonsOSC library and here now we use a OSC package developed for Unity.

For the rest, it works exactly the same as the communication in the tangible interface.

Additional features

This mid-air interface has to limitations that the tangible one doesn't what requires to implement two additional features.

The first limitations is the possibility to place sound sources inside the sphere. As explained before, JAULAB only plays sounds that simulate to be outside the sphere, no sounds can be reproduced inside it. With the first interface we had a sphere mock-up that couldn't be physically trespassed by the Aruco tokens. Here in the mid-air interface we have a virtual environment where everything can happens.

We have to possible solutions to this: placing a collider to the sphere that doesn't allow the virtual sound sources to enter the sphere when the user moves them or letting sound sources going in inside the sphere, but not sending the updated position to the JAULAB.

I opted for the second solution, as placing a collider would lead to some collision's problems with objects crazy bouncing that would frustrate the user. By choosing not to update the sound sources position when they are inside the sphere we resolve the problem, but we a little inconvenient. If we introduce one sound source by one side of the sphere and it exits by the opposite side, in JAULAB the sound will directly jump from one point to the other, not doing a continuous movement. But this is a little issue without major importance.

The second limitation is due to the Leap Motion. A Leap Motion is an infrared camera with a great focal amplitude, but it is not infinite, so if we move away the hand too far, the camera will lose it and will stop moving the sound source.

This directly affects the sounds mute. This is one aspect that I haven't commented yet cause is a natural event in the mid-air interface.

If you move away the sound sources far enough from the sphere, the sound will become each time lower and lower until one point that will mute. The problem in the tangible interface is that the focal amplitude of the Leap Motion is not enough to lower the

volume enough to mute it. Thus, we must find another way to mute the sound sources when we want.

This problem doesn't appear in the first interface, as the camera here can be much further than the Leap Motion camera, covering a wider workspace and having enough space to lower the volume until mute it.

The solution I've proposed for the mid-air interface is to include a transparent box inside the Unity environment that will serve as mute box. All the tokens introduced inside this box will be automatically muted. And once they exit the box, they will start again playing.

The box is close enough to the sphere to be reachable without getting out of the focal radio and at the same time is far enough from the sphere to let a wide range where the volume can be lowered progressively without muting it abruptly.

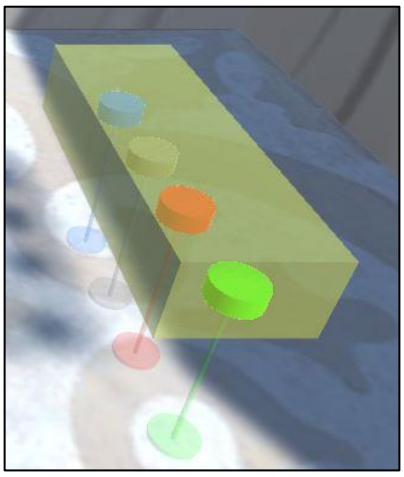


Image 28: Image of the mute box.

To finish with the additional features, once the interface was finished I realized that sometimes the user doesn't orientates really well on the third axis-z, the depth.

This is mainly because the Unity environment is saw by the user from one unique perspective, having a good perception of the movement in two axis but harming the perception in the third cause with only one view, only 2 axis can be covered.

There are two possible solutions to this. The first one is to add one more view of the Unity environment giving the user two different points of view of the scene, like having two screens. This is a good approach because we give the user more information that he can use on his behalf to improve the visual guidance.

The problem here is that after some test with volunteers they all agreed that the two perspectives shown at the same time was more harmful than beneficial. To look at the two perspectives at the same time confused a lot. This way this solution was discarded.

Then I came up with a new solution. Instead of giving a second perspective, the Unity environment would have more visual guidance by having some shadows that indicate where each sound source is.

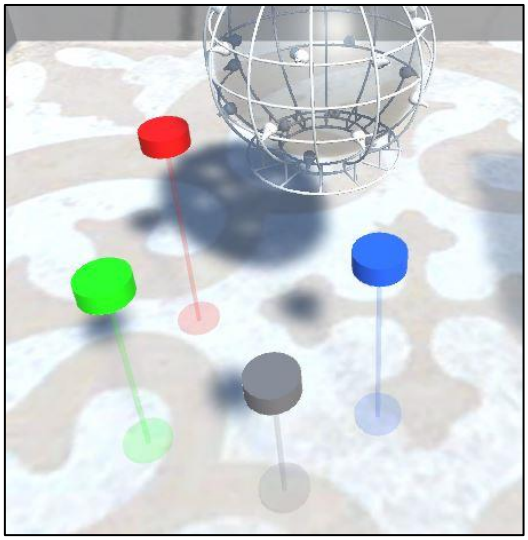


Image 29: Image showing the new shadows projected on the ground.

As we see above, now each virtual sound source projects his shadow to the ground. Like these, looking to all the shadows from all the sources that are in the scene, we can orientate ourselves much better where each source is.

Volunteers agreed that this was a better solution, so this was the one finally chosen.

USER STUDY

At this point, two interfaces have been developed for JAULAB. Two different versions with their advantages and disadvantages.

During the development of the interfaces, some people passing around the lab got curiosity about the project and tested it. Some of them, advised me about some aspects to improve that they disliked, like the two cameras perspective mentioned before. Other ones told me about some aspects of the interfaces that they really loved. But they really didn't know which of the two interfaces they prefer.

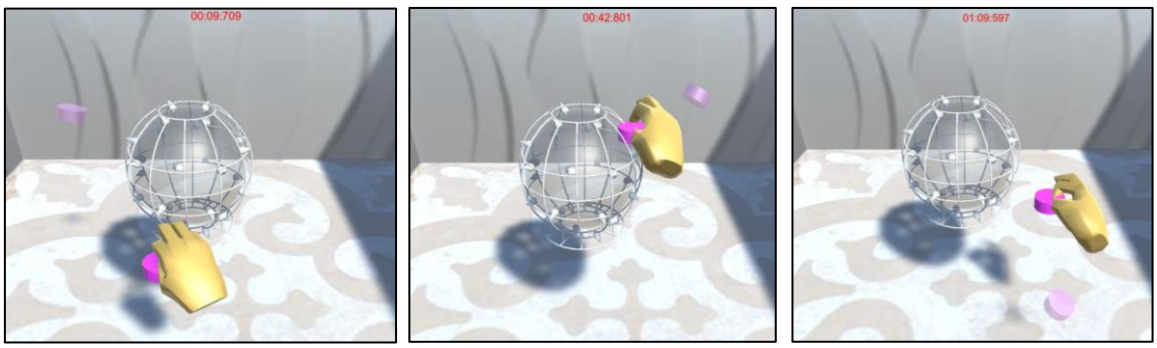
Having developed two interfaces for the same software, it makes sense to try to know which of them two is better, both in terms of ease of use and in terms of user appeal.

For this purpose, the most logic process is to carry out a user study where real users make us of the interfaces and different data is collected. Then this data can be analyzed, and a lot of conclusions can be extracted from it concerning which interface is better in certain aspects.

In this case, the user study is divided in two parts.

One first part where the user will realize an exercise to measure the user's performance with that interface. The exercise consists of a docking test.

A docking test is a task where the user must follow a determined set of steps until the end. For example, in the case of a robot, a docking test would be the robot to follow a sequence of points until the finish. In our case, the docking test consists of the user moving one token to a determined sequence of points until the end. The time he takes to complete the task will be measured and used to get to conclusions.



Images 30: Images showing a user performing a docking test.

The docking task needs a 3D environment to be performed. Unity perfectly fits this requirement as its designed for videogames developing and a docking task is essentially a very simple game. That's why here raise a new necessity.

The tangible interface needs a Unity simulation just as the mid-air one in order to dispose of a 3D environment to perform the docking task. However, this simulation must not be exactly the same one as the one from the Leap Motion version.

In the Arucos version, all the logic is developed in Python with the camera detection, coordinates processing… So, the Unity simulation must not to have no more code than the necessary to perform the docking test. Therefore, this simulation is much simpler than the Unity simulation from the Leap Motion version, where almost all the logic is located at the simulation.

Thus, a Unity environment was developed for the Arucos version.

We realized that it could not only be used for the docking test, but also it serves as a great visual guidance. So now on, when using the tangible interface, there is always the option to open the Unity simulation just as an extra resource to visually see how your movements are being processed.

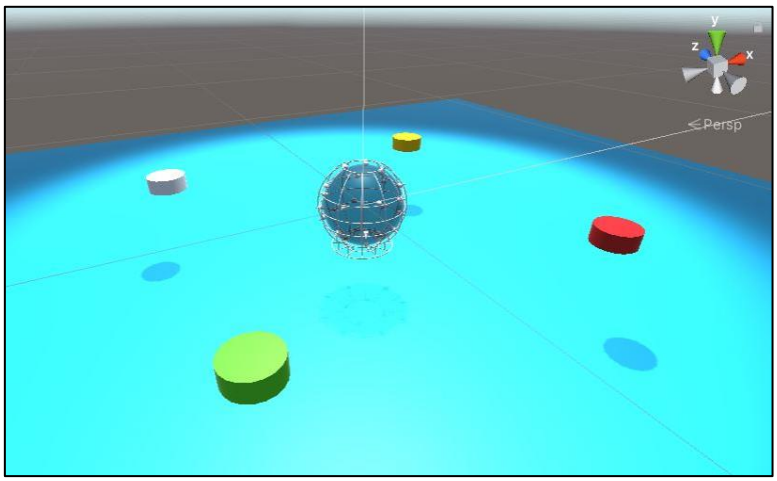


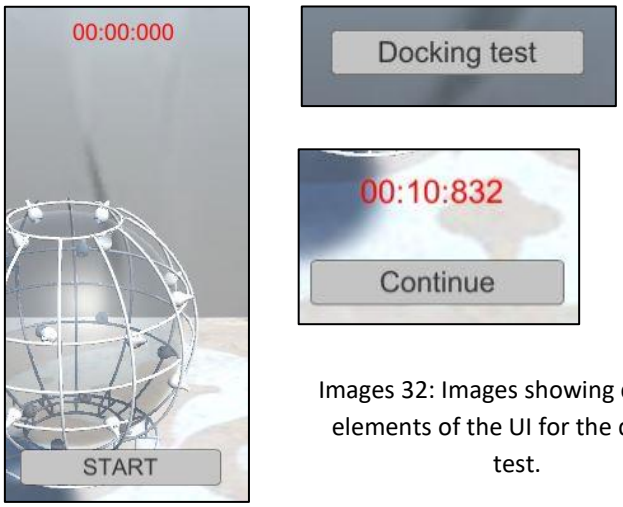
Image 31: Unity simulation of Arucos' version.

With the two Unity environments developed, the next step is to implement the docking test in both.

Obviously, the test must be the same in both versions for the results to be comparable.

As said before, the test consists of a set of points that the user has to reach by moving one token to those points. Depending on the version, he will have to do the test with tangible tokens from the first interface or with the mid-air interaction of the second interface.

Apart from the test itself, a little user interface has been developed too to guide the user. This UI consist of a set of buttons to start the docking test and a chronometer showing at the top of the screen to see the user's performance.



Images 32: Images showing different elements of the UI for the docking test.

The second part of the user study will consist of a questionnaire with a set of subjective questions which users will rate from 1 to 7 depending on how much they agree with the question, 1 totally disagreeing and 7 fully agreeing.

Below in the next page we can see an example of the questionnaire:

Nombre: **ID:**

Edad: **Genero:**

Puntúe las siguientes afirmaciones:

1 - totalmente en desacuerdo, 2 - bastante en desacuerdo, 3 - algo en desacuerdo,

4 - opinión neutral,

5 - algo de acuerdo, 6 - bastante de acuerdo, 7 - totalmente de acuerdo

Pregunta	Mano en el aire	Piezas madera
El esfuerzo mental necesario para usar este método es muy alto		
El esfuerzo físico necesario para usar este método es muy alto		
La dinámica de las actividades con este método ha sido muy rápida		
He tenido éxito realizando las actividades con este método		
He tenido que invertir mucho esfuerzo para llevar a cabo las actividades con este método		
He sentido mucha frustración realizando las actividades con este método		

Pregunta	Mano en el aire	Piezas madera
Me gustaría usar este método con frecuencia		
El método es innecesariamente complejo		
El método era sencillo de usar		
Necesitaría la ayuda de un especialista para usar este método		
Las diversas funciones del método están muy bien entrelazadas		
Había demasiadas inconsistencias en este método		
La mayoría de personas podría aprender a usar este método con rapidez		
He encontrado el método muy aparatoso		
Me he sentido muy seguro usando el sistema		
Necesito aprender muchas cosas para empezar a usar este sistema		

Puntúa del **0** (suspenso) al **10** (sobresaliente) las técnicas

Mano en el aire	Piezas madera

Tiempo tardado en la primera prueba (a rellenar por el evaluador)

Mano en el aire	Piezas madera

CONSENTIMIENTO INFORMADO PARA PARTICIPACIÓN EN EL PROYECTO

1. ¿Qué es y qué persigue el estudio?

El estudio analiza la mejora el control de fuente de sonido 3D dentro de una esfera para renderizado de audio espacial.

2. ¿Quién me va a realizar la prueba?

Investigadores del Departamento de Estadística, Informática y Matemáticas de la Universidad Pública de Navarra (UPNA) así como del Departamento de ciencias.

3. Metodología empleada.

El estudio se realizará en el laboratorio de acústica del Sario, UPNA, y consiste en una breve entrevista personal para recoger información sobre edad, género, y dos actividades en las que usted controlará las fuentes de sonido.

4. Contacto

Para solicitar más información sobre el proyecto y/o tratamiento de sus datos puede contactar con el equipo de investigación a través de **merino.127911@e.unavarra.es**

DECLARACIÓN DEL PARTICIPANTE

☐ Se me ha entregado una copia de este consentimiento informado, fechado y firmado. Se me han explicado las características y los objetivos del estudio y los posibles beneficios y riesgos que puedo esperar. Se me ha dado tiempo y oportunidad para realizar preguntas. Todas las preguntas fueron respondidas a mi entera satisfacción. Puedo ponerme en contacto con el equipo investigador en cualquier momento para realizarle preguntas sobre el mismo.

☐ Sé que se mantendrá en secreto mi identidad.

☐ Soy libre de retirarme del estudio en cualquier momento.

Por tanto, YO_____, con DNI _____, tras comprender toda la información que se me ha proporcionado, consiento participar voluntariamente en este estudio.

Fecha _____ Firma del participante _____

Firma del investigador _____

Combining the docking task and the answers of the users to this questionnaire we have sufficient information to analyze and compare both interfaces.

The process to carry out the user study was always the same one for every user:

- The volunteer enters the sphere and I give him a little explanation of 2-3 minutes about JAULAB and the two interfaces developed.
- After the explanation, we start with the first interface. I give the user around 1 minute to familiarize with the interface, giving him little advises to get used to.
- After 1 minute playing a bit, I interrupt him and we start the docking test. The user does the docking test for this first interface (around 2-3 minutes) and I write down the time he lasts and some other things I may see, like problems, recurrent gestures…
- After the docking test, the user has 3 minutes to freely use the sphere with this first interface.
- Finished the 3 minutes, we change to the second interface and we repeat the same process.
- Once the user finishes with the second interface, he goes out of the sphere and I give him the questionnaire to fill all the questions and give a score to each interface. (5 – 10 mins)
- At the end, I ask them if they have any observation or additional comment about the interfaces they would like to share with me-

Depending on the user performance on the docking tasks taking more or less time, the whole process lasts around 20-25 minutes for each user.

To ensure the equality of conditions between the two interfaces, each time a new user performs the study, the order of the interfaces within the study is changed. This means, one time the user starts with the LeapMotion version and then the Aruco markers version, and next user will start the other way around, first the Aruco markers and then the LeapMotion.

With this, we avoid the users to perform much better in one version. Both interfaces are similar and making the study always in the same order would lead to the users getting used to in the first interface and then better performing in the second one as they have some experience. By changing the order we avoid this.

RESULTS AND DISCUSSION

The user study was done by a total of 14 volunteers, male and females from different ages. Specifically, a total of 11 males and 3 females of an average age of 30.9 ± 11.2 years old.

The mean docking task completion time was 151.6 seconds (2 minutes and 31 seconds) for the LeapMotion interface and 111.5 seconds (1 minute and 51 seconds) for the Aruco markers interface.

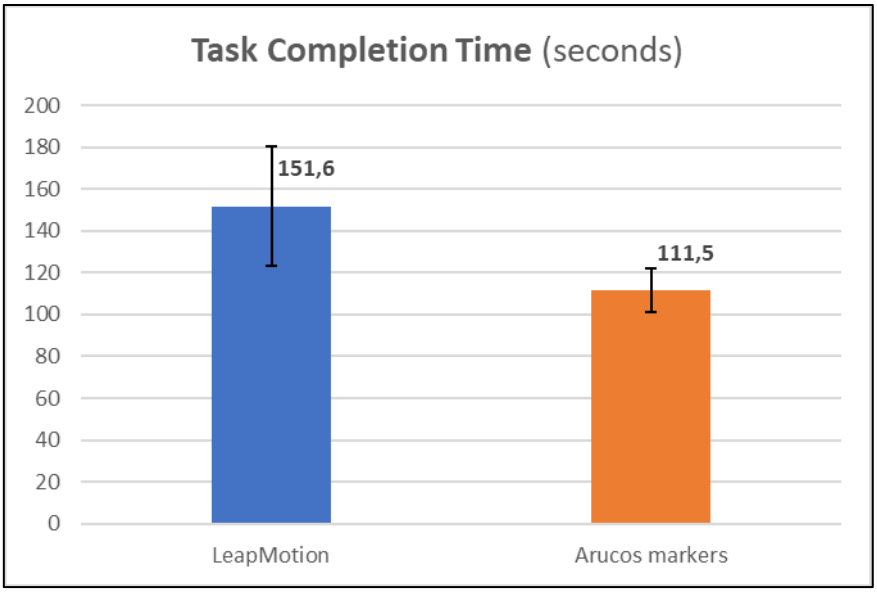


Chart 1: Chart showing the average task completion time for both interfaces.

The mean task completion time from the Aruco markers interface is considerably less than the LeapMotion interface. Considering the equality of conditions during the user studies, this big difference between the two interfaces leads us to the conclusion that the Aruco markers interfaces is much easier to use.

However, this conclusion is a bit biased from the reality. During the realization of the studies there were 3 users who had a lot of problems using the LeapMotion interface. The LeapMotion camera wasn't tracking well the users' hands and gestures, resulting on really bad performance at the docking task.

These 3 users made 3:59 mins, 5:33 mins and 7:00 mins at the LeapMotion docking test, times much greater than the average time of 2:31 mins, so these 3 users are rising considerably the average time leading us to a biased conclusion.

In fact, if we remove these 3 samples from the volunteers' pool, the average times change in a very interesting way.

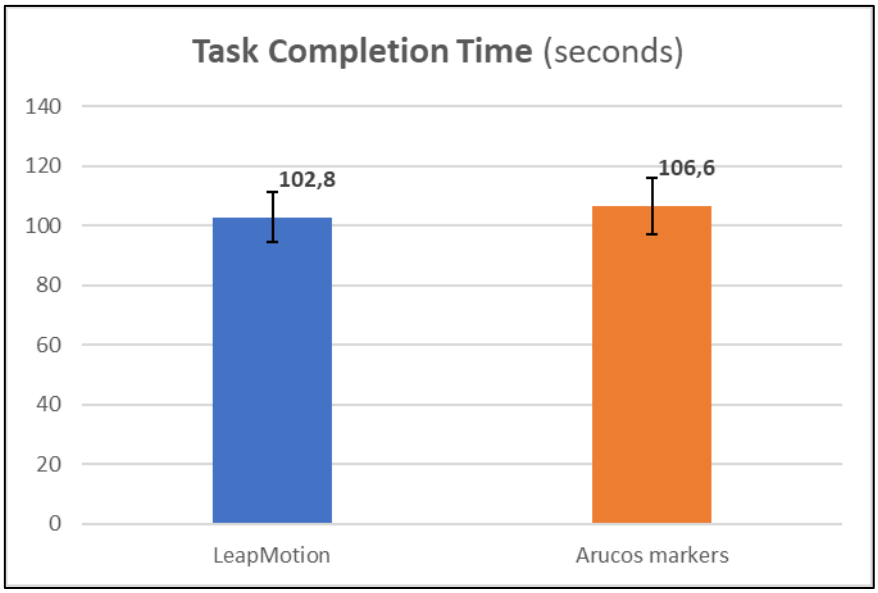


Chart 2: Chart showing the average task completion time for both interfaces after removing the 3 problematic samples.

Now we can see how the average task completion time is very similar between the two interfaces, only 4 seconds of difference. The average time for the Aruco markers lowers only 5 seconds, to the 106.6 seconds, while the average time for the LeapMotion version is greatly reduced to the 102.8 seconds, almost 50 seconds less. The LeapMotion average time is less than the one from the Aruco markers version.

Thus, we can get a different conclusion from the previous one which is that, if the hand tracking goes well, in average, the LeapMotion interfaces is a little easier and faster than the Arucos one.

This was something I wrote down in my notes during the studies, for users that the LeapMotion tracking goes well it's easier to use this version than the other one. But if the hand tracking goes bad, the difficulties increases and thus the task completion time increases as well.

However, the hand tracking is intrinsic to the LeapMotion interface, so we can not remove the samples where the tracking has gone wrong, letting us the first conclusion: in general, the Aruco markers version is easier to use.

Looking at the scores the users gave to each interface we come to a similar conclusion.

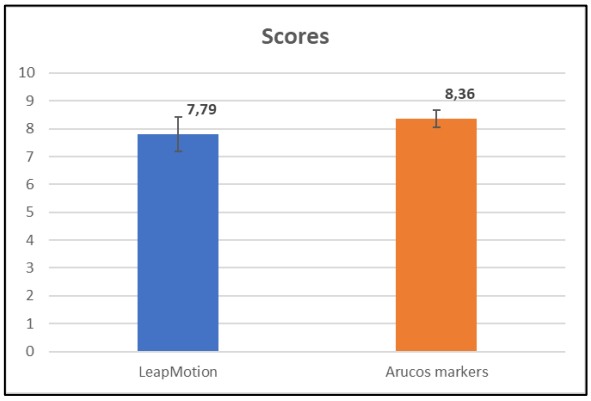


Chart 3: Chart showing the average scores for both interfaces.

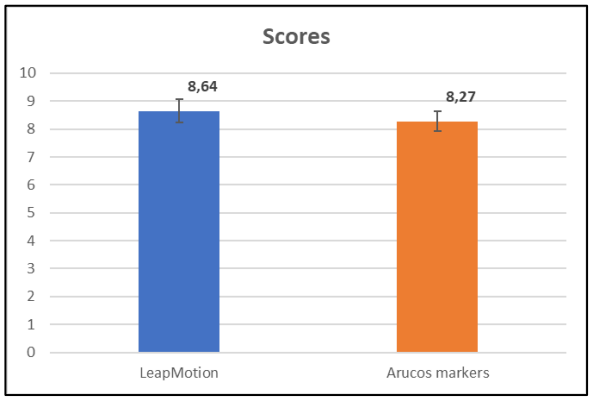


Chart 4: Chart showing the average scores for both interfaces after removing the 3 problematic samples.

Without excluding the 3 problematic samples, users give in average half a point more to the Aruco markers version, 8.36 points against the 7.79 points from the LeapMotion version. But excluding the 3 problematic samples we see two interesting details.

The first one is that the LeapMotion version increases its score almost one point, telling us that when the hand tracking goes bad it leads to a great frustration of the user who obviously gives worse scores.

The second interesting thing is that the average score from the Aruco markers version lowers when we exclude the 3 problematic samples. This means that the problematic users are giving to the Aruco markers versions in average higher scores to this version compared to the rest of users. This is somehow telling us that they score better the Aruco version not really because they think is better but because their bad experience with the LeapMotion one is leading them to score and appreciate higher the Aruco's one

Thus, we come to the same conclusion as before. If the hand tracking goes well users take less time to complete the task and rate better the LeapMotion interface. However, the hand tracking is intrinsic to this version and is something that must be revised and fixed.

At the following chart we can see a similar conclusion.

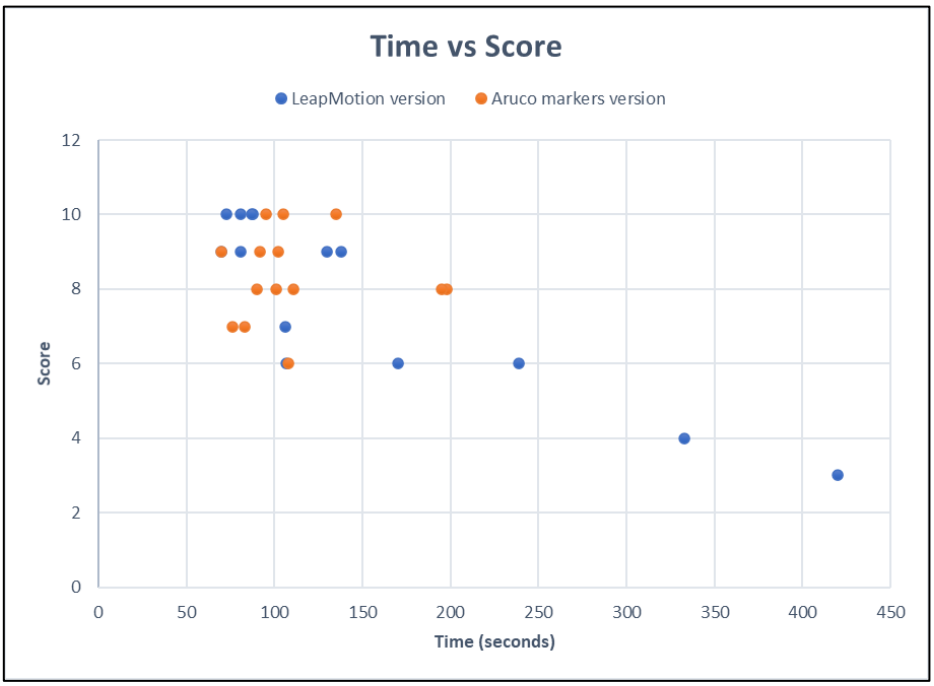


Chart 5: Chart showing the average score against the task completion time for each user.

This chart shows the task completion time (X-axis) against the score the users gave (Y-axis).

In the chart, we can see a clear tendence where the less time it takes the user to complete the docking test, the higher score he gives to the interface and vice versa.

Users prefer the interface easiest to use, being in this case the Aruco markers one and the harder the interface is, the less they will like it.

Not only the completion time and the scores have valuable info, but also the subjective questions from the questionnaire have info interesting to analyze.

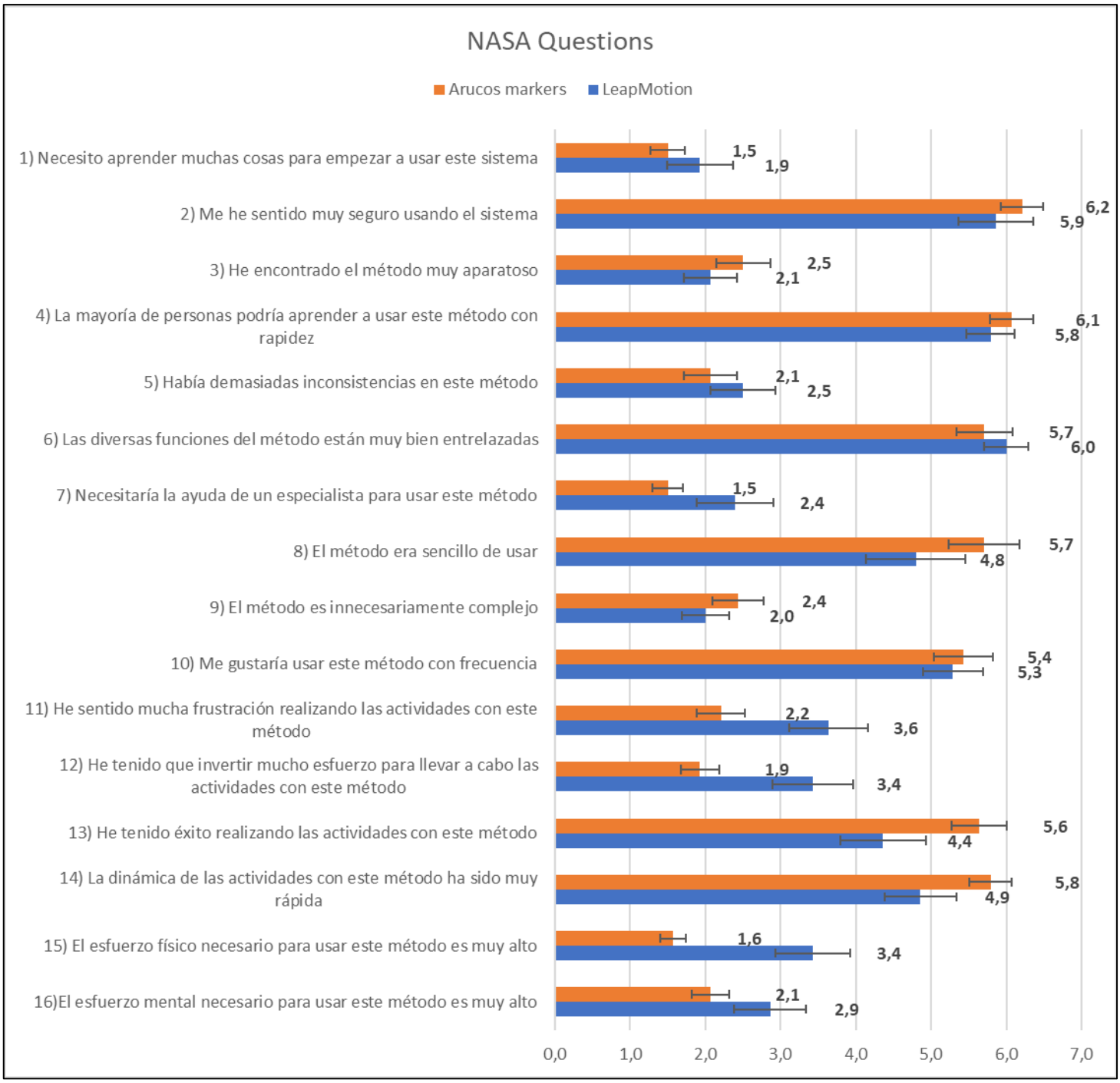


Chart 6: Chart showing the average score of each question from the questionnaire for each interface.

The chart above shows the average score users gave to each of questions from the questionnaire.

Looking at the results we can extract some powerful conclusions.

*NOTE: It's important to remember users rated from 1 to 7 depending on how much they agree with the question, 1 totally disagreeing and 7 fully agreeing

Looking at question 1 we see how users consider they don't need too much to start knowing how the interfaces works, this means that the functioning of them is easy to understand same the may find later some complications or not. Not a significative difference between the interfaces in this question (1.5 vs 1.9). Question 4 is similar to 1. Users almost completely agree for both interfaces that most people could learn how to use the interfaces quickly. Here again there is not a significative difference between interfaces.

The first great difference between interfaces comes with the question 7, asking if the user would need the help of an expert to use the interface. The scores users gave to this question are relatively low (1.5 and 2.4), but there is a great difference of almost 1 point between the Aruco and the LeapMotion version, leading to the conclusion that users generally think that the LeapMotion version is more prompt to generate the need of help from an expert, meaning that they consider this version harder to use.

Similar to this one we have the questions 12, 15 and 16, all of them asking about the mental and physical effort needed to use the interface. And in all of them users agreed that the LeapMotion version needs a greater effort, both mental and physical to use it. This leads to the question 11, where users were asked whether they felt any frustration using the interface, agreeing again that the LeapMotion version frustrated them the most (2.2 points Arucos vs 3.6 points LeapMotion).

One outstanding fact is that, despite the answers to these questions, when asked at question 10 about with which frequency they would like to use one interface or the other one, the Aruco version only wins by 0.1 points.

However, it's logical to suppose that this analysis is also biased by the problematic hand tracking samples so would be nice to do the same analysis excluding these sample as before.

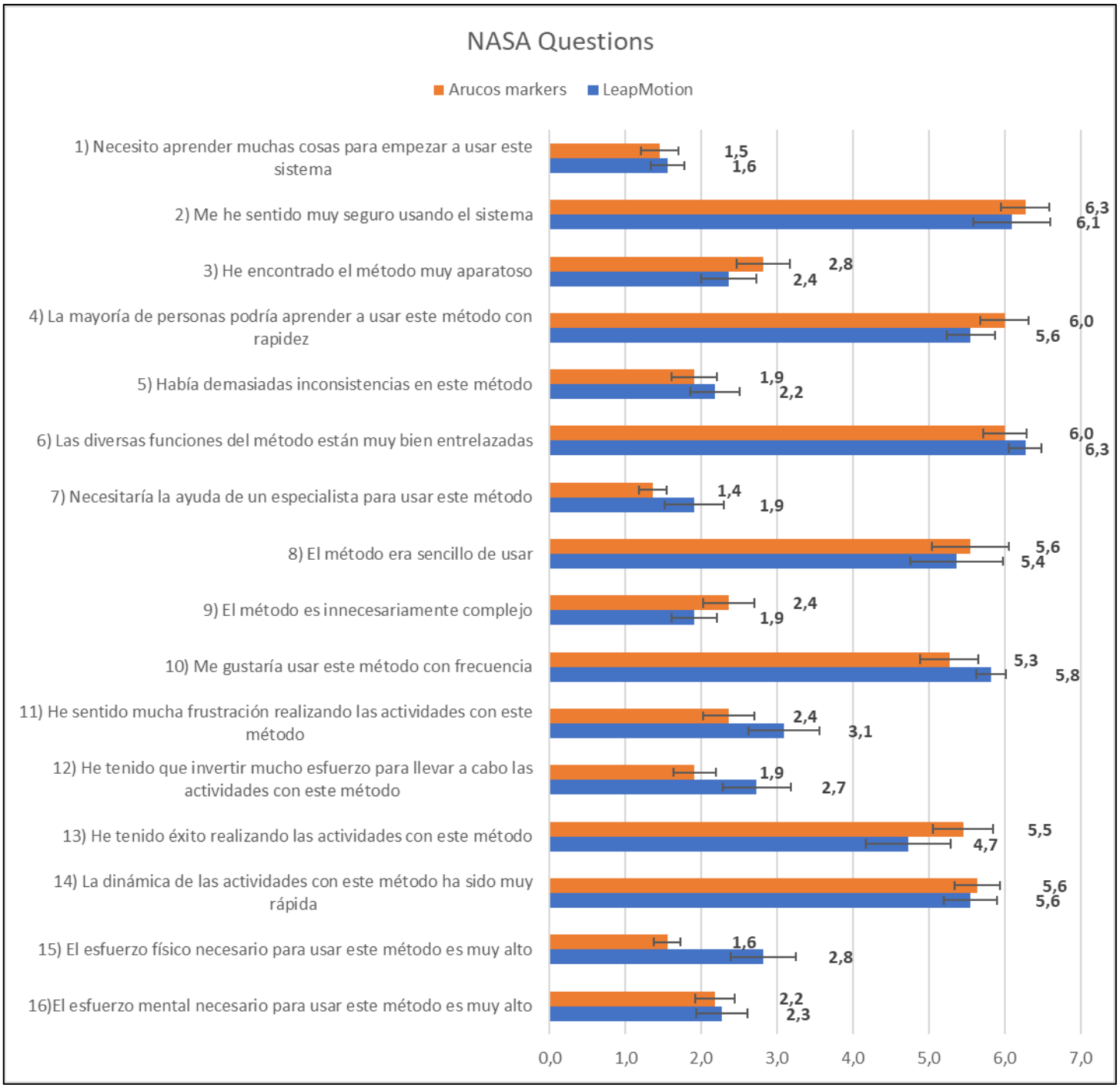


Chart 7: Chart showing the average score of each question from the questionnaire for each interface after removing the 3 problematic samples.

Looking to the chart above we observe the same conclusions as before. By removing the 3 problematic hand tracking samples the general opinion about the LeapMotion version improves considerably.

All the questions about the effort needed to use the interface (12, 15 and 16) lower its score more than half a point (0.7, 0.6, 0.6 points less respectively).

At question 10 users are asked about with which frequency they would like to use one interface over the other one. Before the Aruco interface won by 0.1 points and now the LeapMotion version wins by half a point (5.3 vs 5.8). Even the Aruco version has lost 0.1 points compared to his previous score of 5.4 points.

Also, at question 11 the frustration users feel using the LeapMotion lowers half a point and another questions concerning how easy is to use the interface, like question 7 or 8, improve compared to its previous score.

Therefore, all this leads us to the same conclusion we have been seeing this whole time. In general terms, the Aruco markers interface is easier to use due to the possibility that in the LeapMotion interface some hand tracking problems can arise for specific users, deteriorating the experience. However, if the hand tracking goes well, as it goes for most people, users handle them shelves better and prefer the LeapMotion interface.

ADDITIONAL NOTES AND FUTURE WORK

Before finishing with the conclusion, I want to portray here some notes I wrote down about the users during the studies. Some of them are interesting points that could be expanded as future work:

- LeapMotion interface:
    - Some users tend to try to grab the tokens by clenching their fists or grasping the tokens with all their fingers rather than pinching with the index and the thumb as explained to them. Seems these gestures are more intuitive for them rather than the pinching established. Could be interesting in the future to analyze this and maybe implement additional gestures to grab the tokens and move them.
    - It seemed to me that people who had the problem with the hand tracking were people with not thin and short hands. This is a personal observation but could be interesting in the future to see if the shape of the hands is affecting to the LeapMotion, making it to perform worse at the hand tracking.

  - When moving the token in the LeapMotion interface, moving them to the right works better than moving them to the left. An interesting point to fix also in the future.

- Aruco interface:
  - A lot of users tended to join the four tokens and move them at the same time. With only two hands it's difficult to move the four, the only way is pushing some tokens with the other ones you are moving. Additional functionalities could be implemented to facilitate these movements.
  - There was a little delay of around half a second between the movement of the token and the reproduction of the movement at the Unity simulation and at the sphere. This could be due to the camera image processing. Would be interesting to work on this in order to reduce that little delay.
  - When moving too fast the tokens, the image the camera is capturing gets blurred and the camera don't track the tokens for few frames. This problem is harder to solve than the previous one cause is something intrinsic to the camera quality. Maybe a better camera which is able to track the tokens when moving too fast could be used.

CONCLUSION

Along this report we have seen the development and functioning of two interfaces, at the same time similar but different at some aspects.

The development of each interface has had its various drawbacks throughout the project. Each interface has its advantages and disadvantages. The user studies have given us valuable info about them, showing us that, in general terms, the Aruco version is easier to use due to the well-functioning with every user, but if the hand tracking from the LeapMotion version goes well, as for most people, users handle them shelves better and prefer the LeapMotion interface.

As seen in the previous section, there is some issues and functionalities that could be improved as future work concerning the actual interfaces.

However, these two interfaces are just an introduction of interactivity, a little taste of the potential of interactivity inside the JAULAB. A new door has been opened where, from now on, a lot of new applications and interfaces can be designed and developed, especially concerning the area of virtual reality and immersion, as explained in the introduction of this report. This is the real future work.

BIBLIOGRAPHY AND REFERENCES

[1] “Java documentation / OpenCV-Python Tutorials / Camera Calibration and 3D reconstruction / Camera Calibration”, OpenCV Open-Source Computer Vision. [Online] Available: https://docs.opencv.org/3.4/dc/dbb/tutorial_py_calibration.html

[2] “Java documentation / Aruco Marker Detection”, OpenCV Open-Source Computer Vision. [Online]. Available: https://docs.opencv.org/4.x/d9/d6a/group__aruco.html

[3] “Java documentation / Tutorials for contrib modules / Aruco Marker detection (aruco module) / Detection of Aruco Markers”, OpenCV Open-Source Computer Vision. [Online]. Available: https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html

[4] Bouguet, J. Y. “Camera Calibration Toolbox for MATLAB.” Computational Vision at the California Institute of Technology. [Online]. Available: http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/example.html

[5] “Documentation / What is Camera Calibration?” MathWorks. [Online]. Available: https://es.mathworks.com/help/vision/ug/camera-calibration.html

[6] Goosebumps, “3D Camera coordinates to world coordinates (change of basis?)”, June 20, 2013. [Entry at Stack Overflow]. Available: https://stackoverflow.com/questions/17210424/3d-camera-coordinates-to-world-coordinates-change-of-basis

[7] “Calculate X, Y, Z Real World Coordinates from Image Coordinates using OpenCV”, April 11, 2019, FDXLabs. [Online]. Available: https://www.fdxlabs.com/author/fdxlabs/

[8] Anti, “Aruco markers with OpenCV, get the 3D corner coordinates?”, September 22, 2017. [Entry at Stack Overflow]. Available: https://stackoverflow.com/questions/46363618/aruco-markers-with-opencv-get-the-3d-corner-coordinates

[9] A. Herrera Escudero, “Sistemas de Coordenadas en 3D”, Universidad Veracruzana, 2014. [Online]. Available: https://www2.unavarra.es/gesadj/servicioBiblioteca/tutoriales/Citar_referenciar_(IEEE).pdf

[10] Biblioteca de la Universidad Pública de Navarra. Oficina de Referencia. “Guía para citar y referenciar. IEEE Style”, 2016. [En línea]. Disponible en: https://goo.gl/LaUj46