Personal Sound Zone Simulator

- Multi-Modal Interactions for Personal Sound Zone Systems -

Project Report

Atanas Atanasov Nikolov

Aalborg University
Electronics and IT

AALBORG UNIVERSITY
STUDENT REPORT

Electronics and IT
Aalborg University
http://www.aau.dk

Title:
Personal Sound Zone Simulator

Theme:
Multi-Modal Interactions for Personal Sound Zone System

Project Period:
Fall Semester 2020

Project Group:

Participant(s):
Atanas Atanasov Nikolov - 20145673

Supervisor(s):
Lars Bo Larsen

Copies: 1

Page Numbers: 43

Date of Completion:
December 22, 2020

Abstract:

The concept of personal sound zones has been around nearly half a century, in that time many method for how to create them has been proposed. While leaving interaction techniques with sound zones unexplored. One of the reasons for that is because sound zones are not yet ready. To circumvent that issue, this project builds a system capable of simulating dynamic and static sound zones for multiple users. This creates an environment in which future researchers can run experiments with spatial sound. As a prove of concept this project reports the findings of an experimental study aiming to explore use of gestures and research on how sound zones should be move. User Experience Questionnaire and exit interview were used to measure the user experience. From our findings we can conclude that zones should move slowly instead of instantaneous as the later is perceived as aggressive by the users.

Contents

Preface **vii**

1 Introduction **1**
1.1 Problem Formulation 2

2 Problem Analysis **3**
2.1 Localisation of Sounds 3
2.2 Location Tracking 4
2.3 Design considerations 5
2.3.1 Support for multiple input systems 5
2.3.2 Support for gesture recognition 5
2.3.3 Support for multiple Sound Zones and multiple types of Sound Zones 5
2.3.4 Support for multiple users in a simulation 5
2.3.5 Support for predefined scenarios 5
2.3.6 Support for data logging 6
2.4 Problem Formulation 6

3 Design **7**
3.1 Setup 7
3.2 Networking 7
3.3 Simulator 9
3.4 Tracking and Detection 10

4 Implementation **13**
4.1 Environment preparation 13
4.2 Networking 15
4.3 Simulator 16
4.3.1 Sound Zone 16
4.4 Tracking and Detection 17

5 Evaluation **19**
5.1 Technical Evaluation 19
5.1.1 Latency Test 19
5.1.2 Synthetic Accuracy Test 20
5.1.3 Real-world Accuracy Test 20
5.2 User Study 21
5.2.1 Evaluation Plan 22
5.2.2 Evaluation Procedure and Setup 23
5.2.3 Participants 25
5.2.4 UEQ Results 25
5.2.5 User Responses 25

6 Discussion **29**
6.1 User Responses 29
6.2 Future Work 29

A Appendix A Server Code **35**

B Appendix B Simulator Commands **37**
B.1 Intro 37
B.2 Local commands 37
B.3 Global commands 38

C Appendix C Simulation **41**

Preface

Aalborg University, December 22, 2020

Atanas Nikolov - 20145673
<aniko14@student.aau.dk>

Chapter 1

Introduction

In recent years, technology has found a way to integrate itself in almost every aspect of our lives, mundane tasks as turning on the lights in a room can be done from anywhere using connected devices. Most of the devices in a household are interacting not only with the people using them but also with one another. For example, a smart device a user is carrying can detect where they are and use that information to control other devices owned by the same user, like turning on lights when they arrive at their house and turning on the TV to their favourite show. With the variety of connected smart devices that have become available over the past few years, smart sound reproduction devices are soon to begin appearing.

In an environment with multiple users each desiring to listening to their own sound source, personal listening devices, like headsets and earphones, are the current solution [3]. If in the same situation users also desire to verbally communicate with each other, there is no market ready solution. Further on, it has been found that prolonged use of personal listening devices is related to an increase in tinnitus and high-frequency hearing loss in adolescent girls[1].

A technology that is expected to make an appearance in the near future will allow for the separation of physical space in to invisible areas of sound, or Personal Sound Zones. Personal sound zones will allow listeners in the same physical space to enjoy audio playing without headphones and with minimal interference to one another.

The concept of conjuring personal sound zones can be traced as far back as 1967 at Illinois Institute of Technology, in a demonstration, loudspeakers were placed on a surface in a way to create an environment in which sound can be controlled while listeners remain able to move freely[4]. At Microsoft Research TechFest 2007, a group of researchers demonstrated their "Personal Audio Space" project[19, 2]. They used an array of 16 linear loudspeakers and were able to enhance the audio waves in one area while cancel them in another, effectively creating two sound zones. Users present at the time reported that they were not able to hear the reproduced music once they left the targeted area. Development in the field of sound zone technology and spatial audio has extended to number of contexts such as vehicle cabins[8], personal mobile and stationary devices[7, 9], office spaces[2], and more recently in homes[13], which is also the primary focus of this study.

1.1 Problem Formulation

"How can a Personal Sound Zone system capable of simulating realistic spacial audio for multiple listeners at once be designed."

Chapter 2

Problem Analysis

Chapter 1 provides and overview of the concept of sound zones and touches upon research done in the field. While most of the effort is on making the technology real there is already some research being done related to interaction with Sound Zones. One problem such researchers are facing is the lack of proper tools. Hence we propose to develop a system capable of simulating Sound Zones using technology that is currently available.

2.1 Localisation of Sounds

First things first, it is important to understand some of the limitations we are facing when designing such simulator. One such limitation is the way humans perceive audio. Minimum Audible Angle (MAA) is the smallest change in the position of an audio source relative to the head, that is detectable by the average person[15]. For tones that are below 1000 Hz and located infront of the head it is calculated that the MAA is around 2°[12, 15, 18]. As the sound source moves to the side of the head the MAA increases and the exact position becomes almost undetectable, if the the azimuth of the sound source relative to the front of the head is greater than 60°, for all frequencies. Figure 2.1 illustrates that. For this study we have chosen to work with MAA of 6°.

Researchers study MAA by having a sound source orbit a stationary person. In reality that would be the same as a person orbiting a stationary sound source, as long as the person is facing the same direction at all times. And Formula 2.1 can be used to calculate needed accuracy for the simulator [17]. Where α is the MAA, a is the distance from the person to the source, and d is the accuracy.

$$d = 2 \times a^2 \times (1 - cos(\alpha)) \tag{2.1}$$

By using the MAA and an average human walking speed of 1.2 meters per

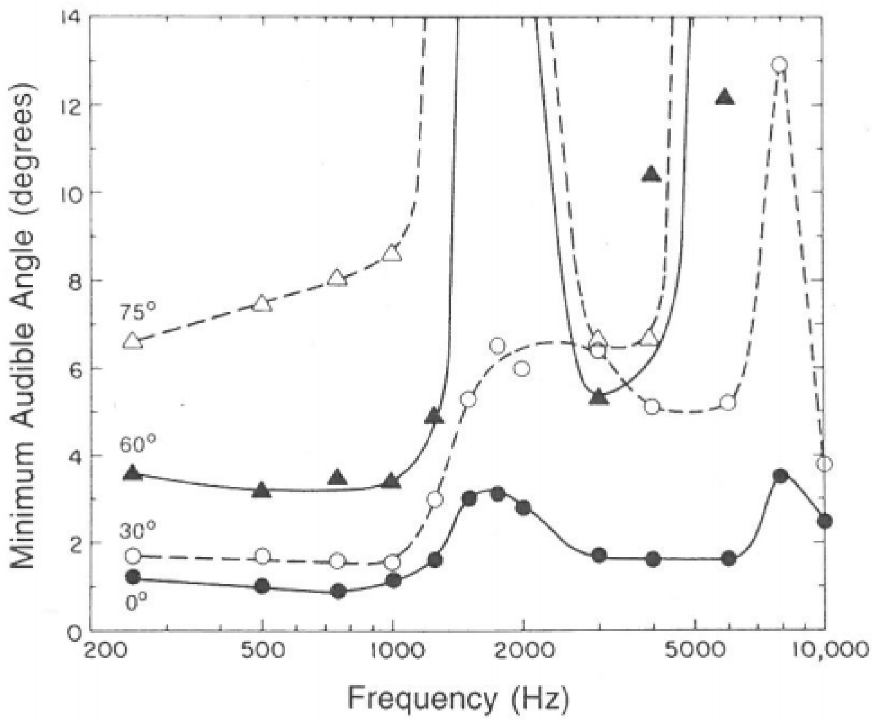

Figure 2.1: Minimum Audible Angle as a function of frequency for various angles [15]

second [14], we can calculate that to simulate realistic spacial sound we need to design a simulator capable of tracking the position of a person within 0.12 meters.

2.2 Location Tracking

In a previous report on a similar topic [17] we compared three different location tracking solutions, namely HTC Vive, Bluetooth Low Energy (BLE) and Visual Tracking solutions. A group of researchers[16], investigated if HTC ViVe could be used as a scientific tool. During their tests they measured average positioning error was 1.7cm. However because the quality of the tracking was severely reduced if the trackers were in any way obstructed from the light towers. BLE is another form of tracking, that uses an array of beacons emitting low energy Bluetooth. A smartphone can then triangulate its own position based on the strength of the signals from that array. However the accuracy of such systems falls in the range of 1.5 meters to up to 3.3 meters in some studies[20]. Lastly is the visual tracking, state of the art tracking systems utilising multiple cameras can achieve accuracy of 0.072 meters [10]. Unlike all other options available visual tracking does not require a tracker to be carried by a person, and allows for orientation to be tracked as well, something only the HTC VIVE is capable of from the other systems.

2.3 Design considerations

2.3.1 Support for multiple input systems

Currently, there are two more solutions installed in the lab that offer location tracking within the room. One is using Ultra Wideband to track tags around the room. And the other is using radio waves to do the same. Both solutions offer high positional accuracy and do not suffer from reduced performance if there is partial occlusion. Both require a device to be carried by a person in order track their position. The simulator needs to incorporate option to be used with both systems if necessary.

2.3.2 Support for gesture recognition

One of the input modalities of interest when it comes to Sound Zone interactions is the use of hand gestures. That is why it is important of the system to either incorporate an interface that will allow for third party system to be connected or include a gesture recognition module on its own.

2.3.3 Support for multiple Sound Zones and multiple types of Sound Zones

Currently Sound Zones could be separated in two sub-groups based on their general behaviour, static and dynamic. However, both of those could be further separated based on functionality. For example, it might be desired for a zone to be static unless a user is about to accidentally leave it. The simulator should make it easy for other researchers to develop and implement Sound Zones that are needed for their research. In other words, the simulator should support multiple Sound Zones at the same time, allow for creation of additional Sound Zones from a template at runtime through an interface, and allow for the easy creation of those templates.

2.3.4 Support for multiple users in a simulation

Another feature of the simulator that is needed to conduct proper research on interaction with Sound Zones is multiple user support. That is required as we are interested to explore what effect Sound Zones can have on human interactions. This sets the minimum number of simulated users during the same session at two.

2.3.5 Support for predefined scenarios

The main focus of the simulator is to aid research into Sound Zones. Such research will often be done through specifically designed scenarios. Those scenarios might require for specific commands to be executed at the same time or objects to move in the exact same way every time. The simulator must include an interface for such sequences to be programmed.

2.3.6 Support for data logging

Last but not least, having the data logged in a way that allows for easy analysis is vital as it can severely cut down on processing time after experimentation. For that reason, data should be internally logged by the system in an accessible way.

2.4 Problem Formulation

Chapter 3

Design

Based on research publications in the area of Sound Zone interactions and the direct feedback of researchers the following considerations and conclusions can be drawn. Those can be seen as system requirements as any prototype that does not include an implementation of those features will not fully satisfy the needs for research in the area of Sound Zone interactions.

3.1 Setup

This section describes the envisioned setup as a whole as well as the modules that make it. The system will be designed using a modular approach, where each module will be responsible for processing certain tasks and communicating with other modules. This approach allows for modules to be replaced and modified without the need to make changes to the rest of the system. On top of that working with modules makes the implementation process simpler as there should not be any big chunks of code to manage. The modules that make up the system are as follows: 1. Networking 2. Simulator 3. Tracking and Detection 4. Gesture Recognition The modules and their tasks will be described later in this chapter.

3.2 Networking

While Unity 3D is a powerful video game engine and comes packed with many tools and features related to the industry, the same make it invaluable asset when prototyping. One such feature is UNet, Unity's own Multiplayer and Networking API, which offers low latency and takes care of most data synchronisation that occurs both on client and server side. Unfortunately, it was deprecated in late 2018 and support is limited. Even if that was not the case, UNet was notorious for causing problems when clients not developed in Unity requested to connect.

Because of that, we opted out to develop our own networking solution. The main benefit of designing a custom solution is that we can implement only the functionality we need and avoid aspects that are not needed. By going through the design considerations, our solution needs to support multiple clients at the same time and be platform independent. The later being necessary since the simulator will be running on at least two operating systems, Windows, and Android. The Networking solution is based on sockets, the main reason being that sockets are one of the most popular and reliable methods for cross-platform communication. As such sockets can be accessed through both Python (our preferred coding language outside unity) and C# (used in Unity) code. In previous iterations of the simulator the server was part of either the tracking module or the simulation module. While that approach makes transfer of data slightly faster it is an unnecessary overcomplication. In the case where the server was part of the tracking system, it was impossible to use another form of tracking. By moving the server to a separate module that can run on its own process resolves that issue. As a result, both the tracking module and the simulator module are threated as clients by the system. One major benefit is an improvement to the stability, as well as the recovery after crashes. Since both modules are clients if one of them fails due to user error or unaccounted for circumstances, the module can simply reconnect and resume from where it was. Something that was not possible when the server was part of any of the two modules. The server is responsible for receiving messages from all clients and sending those messages out to only those clients that need them. There are two types of messages, targeted messages, and broadcast messages. Targeted messages originate from the tracking module and are meant to be received by only one client. Broadcast messages on the other hand usually are send from the simulator clients as a direct response to user input, those messages serve as synchronisation between devices and are received by all, except the tracking system. It is important to separate the two types of messages and to not overload the network as weaker devices, like smartphones, will have difficulties keeping up with the incoming stream of data and will slow down, and eventually crash. Targeted messages are to be send from 15 to 20 times per second, depending on the system, that number is the framerate of the tracking module. With three people in the simulation, weaker devices struggle to read data from the socket and process it fast enough, however processing the data with one person seems to not be a problem. Broadcast messages are not automated and are only send when a user performs and action, which realistically will not happen more than once every few seconds, the effect of those is negligible and can be ignored. Also due to the nature of the transmitted data there is no easy way to reduce it without desynchronizing the clients.

Examples of targeted messages:

1. The position of a specific user.

2. A gesture performed by a specific user.

Examples of Broadcast messages:

1. User moved/resized a Sound Zone.
2. User pressed play/pause/volume button.
3. User changed the state/type of a zone.

3.3 Simulator

The simulator app is the part of the system where we leverage the power of Unity 3D, that includes rendering the virtual room and simulating accurate enough audio. The app serves as the main interface between the users and the Sound Zones, providing both options to interact with them and visual feedback. Additionally, the app is responsible for simulating the audio and playing it through headphones. The app should show a view of the room with icons showing the location of the Sound Zones as well as marking the location of furniture that is in the room. The room size and shape of the virtual room should be proportional to the real room to avoid confusing the users. The app should also show the position of the local user, clearly indicating which direction they are facing. Most of the interactions will be placed on top of the Sound Zone icons to avoid confusion as it is possible that with a large number of Sound Zones or in cases where they are overlapping. Figure 3.1 shows how a sound zone will look like when off. Interactions with the Sound Zones though the touch interface should happen in one of two ways, through the use of interface elements and gestures. Interface elements like buttons and sliders, will be used to interact with the audio player functions like play, pause, stop, volume, previous, next. Gestures like dragging, pinching holding, will be used to change the appearance or position of the Sound Zone. This decision is based on our view of Sound Zones as digital speakers. On a real-world speaker buttons will be used to control the audio, and the speaker will have to be physically "dragged" to another place if relocation is desired. The one exception is the gesture for resizing as real-world speakers cannot change size. To accurately recreate the behaviour of Sound Zones in our simulation we will be using two virtual sound sources per Sound Zone. One sound source, called "master" will be representing the area of space in which audio is desired, this zone's audio volume will be attenuated starting at a certain distance from its centre. The attenuation will be linear over a short distance and will stop when max distance is reached. The second zone will be located on the edge of the room and will act as the "leaking" sound from the "master" source, the volume of this zone will not be attenuated over distance, instead it will always be 20dB lower than the volume of the master zone. The 20dB difference between the two zones is based on our research which showed that the

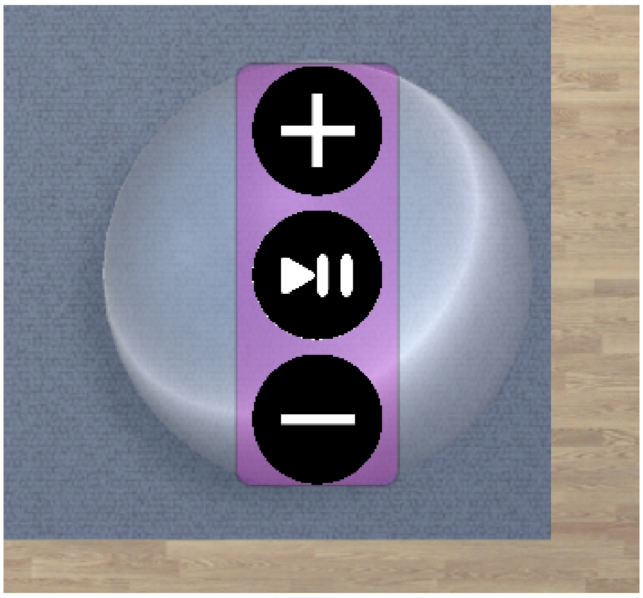

Figure 3.1: A sound Zone, the colour around the buttons shows the colour and type of the zone. When the zone is playing it will turn that colour. Play/pause and volume buttons are located on top of the zone. If the zone dragged on the left it will change in size. If it is dragged on the right it will be moved.

this should be a realistic contrast value between a “bright zone” and a “dark zone” in a real Sound Zone system. The sound source is located on the edge of the room to create the illusion that the sound is coming from a sound bar, similar to how real Sound Zones will be generated. This is not done for the “master” source as the goal is for the audio in this zone to be uniform.

3.4 Tracking and Detection

For our own tracking system, we decided to use OpenPose[6, 5, 21] with four cameras, while two other tracking solutions are already installed in the room, both of them offer only location tracking. The benefit of OpenPose is that it is possible to extract both location orientation and gestures from the same images. In addition, there is no need for specialised hardware since OpenPose works on standard RGB video files. While higher resolution yields better result it also comes at a higher computational cost, but for our needs processing images at 480x640 is sufficient. OpenPose does a great job at extracting keypoints from images, but one place it fails is at matching keypoints from different camera views to the same person. Fortunately, there are plenty of methods that will do just that. We experimented with two histogram comparison and k-means clustering. For both methods, a section of the body of the person that is being matched is used to build a colour-based model.

For histogram comparison the section locked between the two shoulders and the waist was used to build a histogram in HSV colour space. HSV was used as it is less sensitive to light compared to RGB. The histogram comparison performed well matching people correct in 80-90% of the time. However, one drawback is that the thresholds had to be adjusted manually and often that was a problem when new colour data was presented. For k-means clustering the same section of the body was used to extract the most dominant colour in RGB colour space. Both HSV and RGB showed similar results, but RGB was chosen in the end as the video was captured in the same format. Unlike with histogram comparison, k-means clustering does all the work on its own, all the data that needs to be given is the data points that need to be clustered and few additional sample colours which determine the number of clusters that are to be formed. The problem of matching people in different camera views based on colour variable is in essence a clustering one, so there is no surprise that k-means clustering solves it nearly perfect with little human input. With all the keypoints clustered the data is ready to be send to the appropriate clients, the data will not be normalised as in its current form it also encodes information about which camera it is coming from. The last thing done in this module will be to check if any of the users is performing any gestures that are recognised by the system. For that we are going to use the best camera view we have for each person, we assume that the best view is the one that is facing a camera. Then this view is compared to a model of a known gesture, if there is a match the corresponding client is notified that a gesture is performed. This message is sent only once when the gesture is first detected.

Chapter 4

Implementation

In Chapter 2 the needs and requirements of a personal sound zone system were described and analysed, and in Chapter 3 a simulator that will satisfy those needs and requirements was designed

This section will go in detail on how the design described in the previous section was implemented. Unity 2019.4.1f1 was chosen as the environment for development for two reasons. First, as discussed earlier, Unity is a powerful game engine which satisfies all our needs and design considerations. It comes with a build in Vector calculations and Sound Sources which will be very useful during the implementation phase. Second, this is LTS version, which means Unity will continue supporting it for at least a few more years, which means the project is less likely to be corrupted in the future due to updates on their end.

4.1 Environment preparation

The lab in which the simulator is setup has a size of about 8x7 meters. Four cameras are placed in each of the corners of the room as close to the ceiling as possible to get as much area covered as possible. The cameras chosen for the project were Logitech C920 PRO, as those are inexpensive and readily available, additionally the quality of most webcams is similar. The C920 has a sensor capable of capturing images with max resolution of 1080p at 30 frames per second.The hardware available for building the simulator is capable of processing the captured images at around 15 frames per second at resolution of 480x640, the purchase of high refresh rate cameras could not be justified. The cameras are be mounted at a 45°relative to the wall and 30°relative to the ground, facing towards the centre of the room. 3D printed plastic brackets were used to mount the cameras to the wall. The stands allow for small corrections in the orientation of the camera to be made later on if needed. The cameras are directly connected to a computer via USB, active cable extenders had to be used for two of the cameras as the recommended maximum length of passive USB 2.0 cables is 5 meters.

The first step is to calibrate the cameras before they can be used to extract any usable data from the acquired images. Calibration is needed since webcams often use plastic lenses which have many imperfections and may introduce distort the captured images.A standart calibration method utilises a black-white chessboard, the pattern is used as the corners of the squares on the board are easy to detect and are structured in a square grid. The algorithm is implemented in Python 3 using OpenCV, an image processing library . The implementation of the calibration algorithm used in this case is simple and could be subject to further optimisations it provides the desired result. The steps the system is performing to calculate the intrinsic camera parameters are as follows:

1. The settings are stored in a file and are loaded by the system when started. Those include which camera is being calibrated, the number of images that should be used, the type of calibration pattern is being used, the size and dimensions of the calibration pattern, the expected size of the input image, and where to store the calibration data.

2. The system captures a series of images using the selected camera every 200ms. Each captured image is checked if it contains a chessboard pattern of the selected size. If it does the image is saved, otherwise it is discarded. The process is repeated until the desired number of calibration images is obtained. To determine if a chessboard is observed a standard OpenCV function cv2.findChessboardCorners() is used with CALIB_CB_FAST_CHECK flag. With this flag a TRUE value is returned if a chessboard is observed, in cases where no chessboard is observed it returns FALSE and shortcut the call.

3. When the required number of images has been captured the system will call the calibration function. During that phase the camera matrix and distortion coefficients will be calculated and saved at the desired location. For the calibration the OpenCV function cv2.calibrateCamera() is used. For simplicity it is assumed that the chessboard was kept stationary at a set distance from the camera, and the camera was moved accordingly. Thus, the object points are in the form (0, 0), (0, 1), (0,2),... The image points used are obtained through the chessboard corners found by the cv2.findChessboardCorners() and further optimised by using another OpenCV function, cv2.cornerSubPix().

4. Lastly, after the calibration is done, a sample image is taken, undistorted and displayed alongside a re-projection error. After review if the results are satisfying, they are saved as a .xml file.

When performing the calibration for each camera 25 calibration images were taken. On each image the chessboard is on a different position in the image and slightly rotated around its Y axis. For simplicity a resolution of 480p was used, however the calculated camera matrix parameters can be scaled up with. No scaling is necessary for the distortion coefficients.

4.2 Networking

As discussed in Chapter 3 the first module is the Server, this section will explain the code and how the data is being transferred. The full code can be found in Appendix A. All data is send as UTF-8 encoded string.

The server is coded in Python and uses the socket protocol to establish a direct connection to a port on a given IP address. In the code lines 5-6 show the declared variables used by the server to start listening on a specific port. The declaration on line 4 is a header, the header will act as an identifier when we are sending data over the network. On lines 8-19 a new socket object is created and set to listen at a the port declared in line 6. The two variables at lines 15 and 17 hold all the data about who is connected to the server. Once the server is ready, a message is displayed in the console, notifying us that the server is up and running.

The function on line 22 is the one that is handling all the incoming messages. It checks if the new message received on the port contains a valid header, if it does the message and the client's name is returned. Otherwise an exception occurs but it is handled immediately by the *try* statement that starts on line 24.

On line 36 the main part of the server starts working. An infinite while loop is checking as often as possible if a new message is received on the port. Whenever a new connection happens the server checks if it is a new one or if it is one of the existing clients.If it is a new valid connection lines 46-50 are executed and the new connection becomes a client. If it is not a valid client it is ignored.

For the cases where it is an already existing client, lines 54-58 check if it is a disconnect message, if it is the connection is deleted from the list of existing clients. In any other case the newly received message needs to be broadcast to the rest of the clients on only to a specific one.

On Lines 59-64 the message is decoded and parsed, and extract the target if one is specified. If the message starts with /user then the message is targeted,line 64 obtains the target client. All commands are available in Appendix B.

On Lines 66-71 a *for* loop, it iterates over all connected clients and sends the message to them if the message is of broadcast type. If it is targeted it skips every client that is not the recipient.

4.3 Simulator

The simulator is the second module that was described in Chapter 3. The simulator is build in Unity and C#. The client side of the simulator works in a similar way as the Server, except when a message is revived it has to be checked for global commands. A list of all commands can be found in Appendix B. If the message contains a global command that command is executed.

There are two types of objects that can be manipulated with global commands. Sound Zones and the local user.

4.3.1 Sound Zone

A sound zone is an object capable of playing audio files in a specific range. Each Sound Zone is made of two sound sources, one called *Master Source* on the location where the zone needs to be and one called *Secondary Source* outside of the room. A script is used to keep the volume of the *Secondary Source* 20 Db lower, Listing 4.1. Figure 4.1 and Figure 4.2 show the roll off of both sound sources. Each button can be pressed virtually with a command, those commands can be used both manually by entering them into a console or by pressing the corresponding button on a Sound Zone of the same colour on any other connected app.Some commands can also be used with gestures, however the list of those is limmited.

```csharp

public class Audiodb
{
    public static float LinearToDecibel(float linear)
    {
        float dB;

        if (linear != 0)
            dB = 20.0f * Mathf.Log10(linear);
        else
            dB = -144.0f;

        return dB;
    }

    public static float DecibelToLinear(float dB)
    {
        float linear = Mathf.Pow(10.0f, dB / 20.0f);

        return linear;
    }
}
```

Listing 4.1: DecibelToLinear

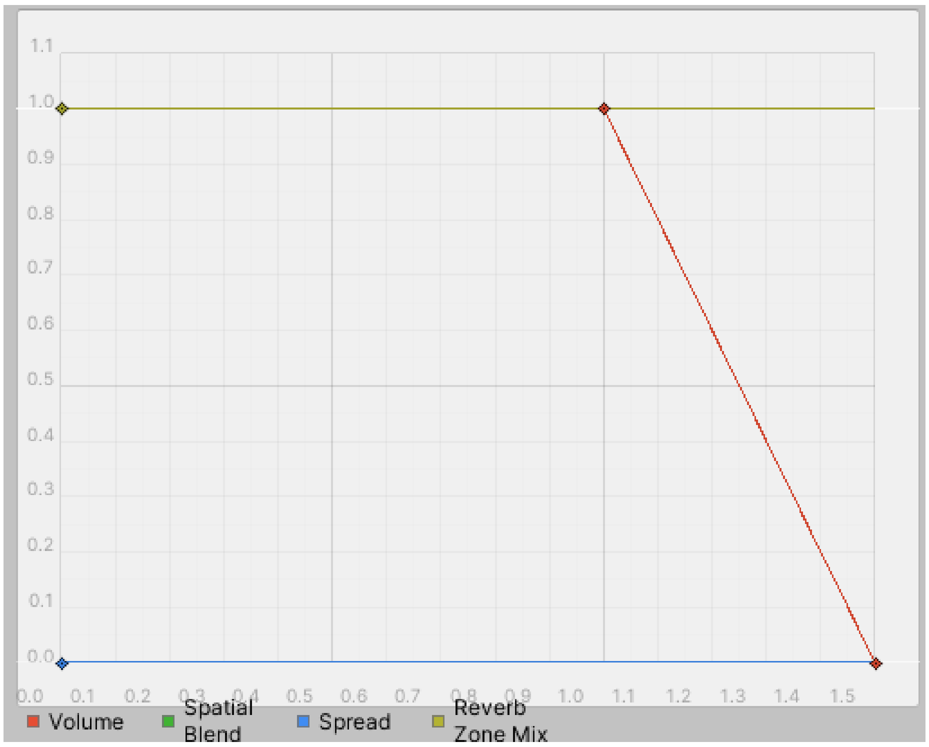

Figure 4.1: Volume attenuation of the Master Source over distance, a linear roll-off is a good enough approximation over a distance of 50cm.

4.4 Tracking and Detection

Commands are executed on objects, like Sound Zones and the user as *Coroutines*. *Coroutines* are special functions that can pause their execution and resume at a later point, this is a feature that is very useful when a lot of commands could be given the client at any time. Additionally, *Coroutines* can be called through the use of *strings*, which makes them perfect for execution over network.

Lastly the most important part of the simulator is the triangulation module itself. Shown in Appendix C, this module uses the location of an object in camera coordinates and casts a series of rays in the scene. All that is made easy because of Unity's build in raycast capabilities. The *Coroutine* on Line 38 takes coordinates from at least two images and turns them in to rays using the proper camera matrix. The proper matrix is determined by the function on Line 96, it returns the Id of the camera that took the picture based on coordinate data. The new coordinates need to first be trimmed by the function on Line 107, which translates from Python and OpenCV coordinates to Unity coordinates.

If at least two rays can be cast, then the function on Line 86 is called. Because it is next to impossible for two rays to cross paths in 3D space, we opted out for the

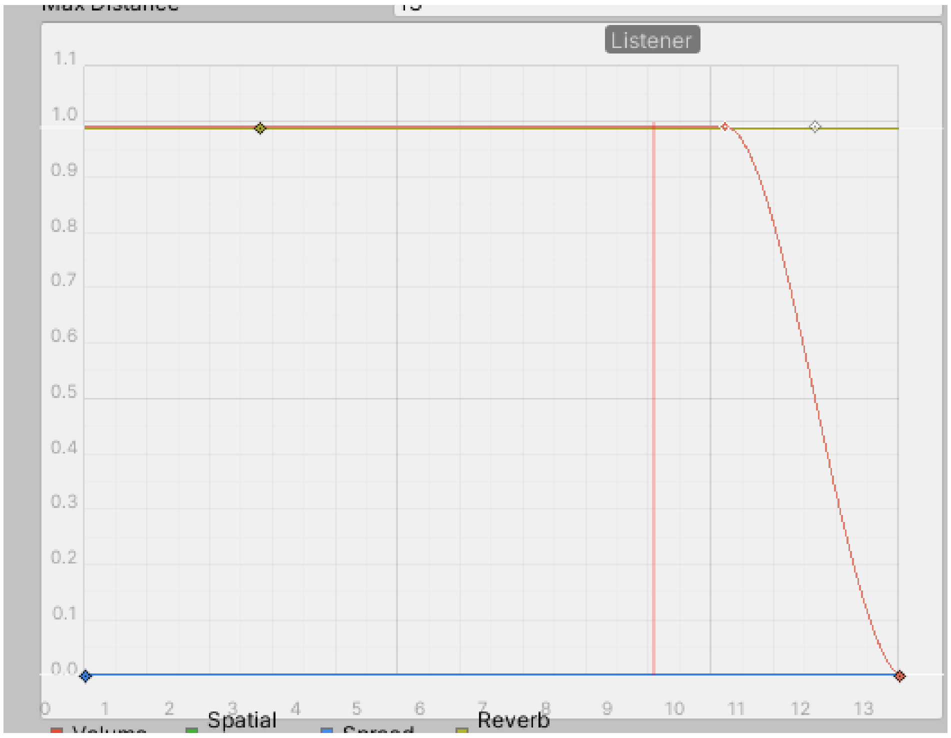

Figure 4.2: Volume attenuation of the Secondary Source over distance, the drop in volume is controlled by a script, the shown range is big enough to cover the entire room.

next best thing.The closest point to all rays in.

This process is repeated for every keypoint that is send from the server. The two keypoints of interest for the simulator are the Neck and the Right Shoulder. The Neck is used as the position of the user in the room, and the Right Shoulder forms a line with the Neck, which is then rotated by 90°counterclockwise to get the orientation of the user.

Chapter 5

Evaluation

The goal of this project is to develop a concept of a simulator that can serve as a tool in future research in the field of Personal Sound Zones. The simulator was evaluated on two levels, technical and user oriented. The technical evaluation tests the position and orientation accuracy of the simulator by performing two studies, a synthetic test run in Unity engine with perfect cameras, and real-world test in the lab with Logitech C920 cameras. The user oriented experiment demonstrates the capability of the simulator to be used in a user study with multiple participants.

5.1 Technical Evaluation

One of the most important aspects of the system is to be responsive to the user's movement and actions.The responsiveness determined by three factors, the time it takes a frame from the camera to reach the computer, the time it takes the computer to process the frame and output the positional data, and the time it takes the data to reach the targeted device.

5.1.1 Latency Test

Accoding to the technical specifications of a Logitech C920 camera, the average latency is at around 200-300ms, however since the cables connecting the cameras to the computer are at the limit of the USB standard for length, a higher than normal latency should be expected. To more accurately measure the delay an Arduino board with an LED was used as shown on Figure5.1. A camera was placed in front of the LED recording trough a Python script. The time was recorded when the Arduino provides power to the LED and once again when the camera registers the light coming form the LED. The python script uses a fast background subtraction and threshold, assuming that the LED is on when the image gets bright. It was also assumed that the cameras using active USB extension cables will be suffering

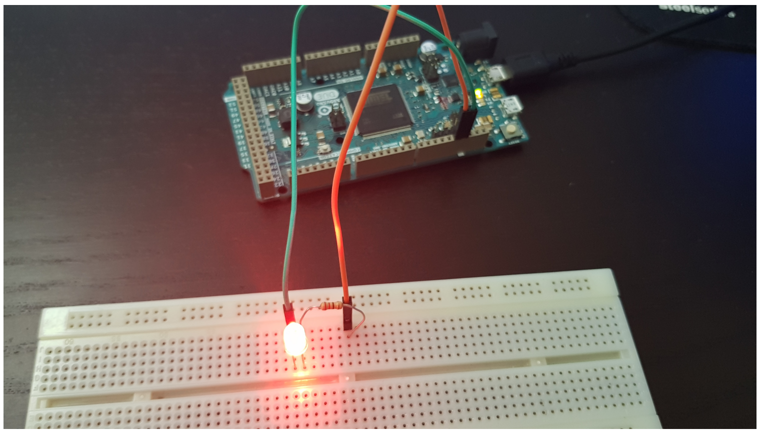

Figure 5.1: Arduino DUE with red LED used to measure camera latency.

the most so only this worse case scenario was tested. The camera resolution was set at 640x480, same resolution was used in the simulator. The calculated latency of the camera is in the range of 200-252 ms.

5.1.2 Synthetic Accuracy Test

The first accuracy test performed uses synthetic data captured in Unity, Figures 5.2. A synthetic test like the one performed allows for the accuracy of the triangulation software to be tested while isolating any errors that might be caused by bad camera calibration or the real world. A 3D human model in generic T-pose and was placed in a specifically designed scene. The model was moved to random positions around the scene while still in view of four virtual cameras. At each new position a picture was taken by all cameras and the real position of keypoints was recorded. In total 250 points were tested using this method. OpenPose was used to process the pictures and the produced keypoints were user to reconstruct the position of the body in the scene. Figure 5.2 shows the two steps of the process. The left image shows that 3D model is accurately processed by OpenPose. The right images shows the perfectly reconstructed wrist point, marked with a red cube. The predicted position matches the world position of the wrist which was recorded earlier. The results are identical for all body points predicted in that way. The orientation of the body is calculated through the position of two keypoints relative to each other. That makes it is save to assume that there will not be any discrepancies between the orientation of the reconstructed body and the original.

5.1.3 Real-world Accuracy Test

To get accurate real world positions 25 points were marked on the lab floor in a 5x5 grid pattern 1 meter apart. The virtual room used in the simulator is made to have the same dimensions as the real world room. The marked points could easily be placed there. The measurements were taken as a person was standing on each marked position in the lab, the simulator is then used to calculate and record the position of the person. Data from each position in the lab was recorded twice with the direction the person was facing randomised. The height of the person is 175 cm, the expected height of the reconstructed neck point is around 145cm. The calculated mean height of the reconstructed point is 86 cm, and the variance is 12cm across the room. Figure 5.3 shows a top down view of the virtual room, with the 25 target points marked in red, and the corresponding triangulated points marked in green. The calculated mean error is 13 cm and the variance is 0.01 cm.

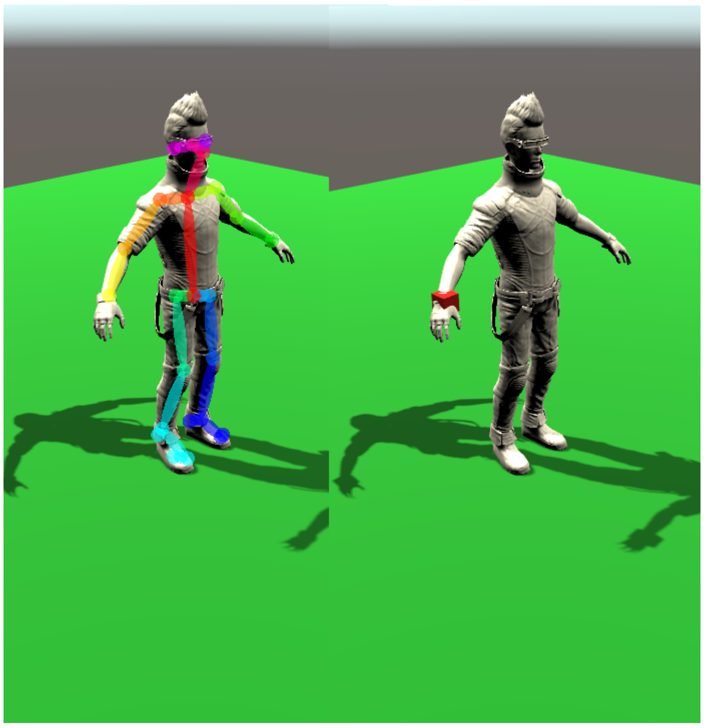

Figure 5.2: Images from the synthetic accuracy test. Virtual character with keypoints detected by OpenPose on the left, and Right wrist detected on the right.

5.2 User Study

The evaluation will follow two scenarios. The first will be aimed at introducing participants to the concept of Sound Zones and how the simulator functions. While the second will present the participants with two different methods for interacting with Sound Zones.

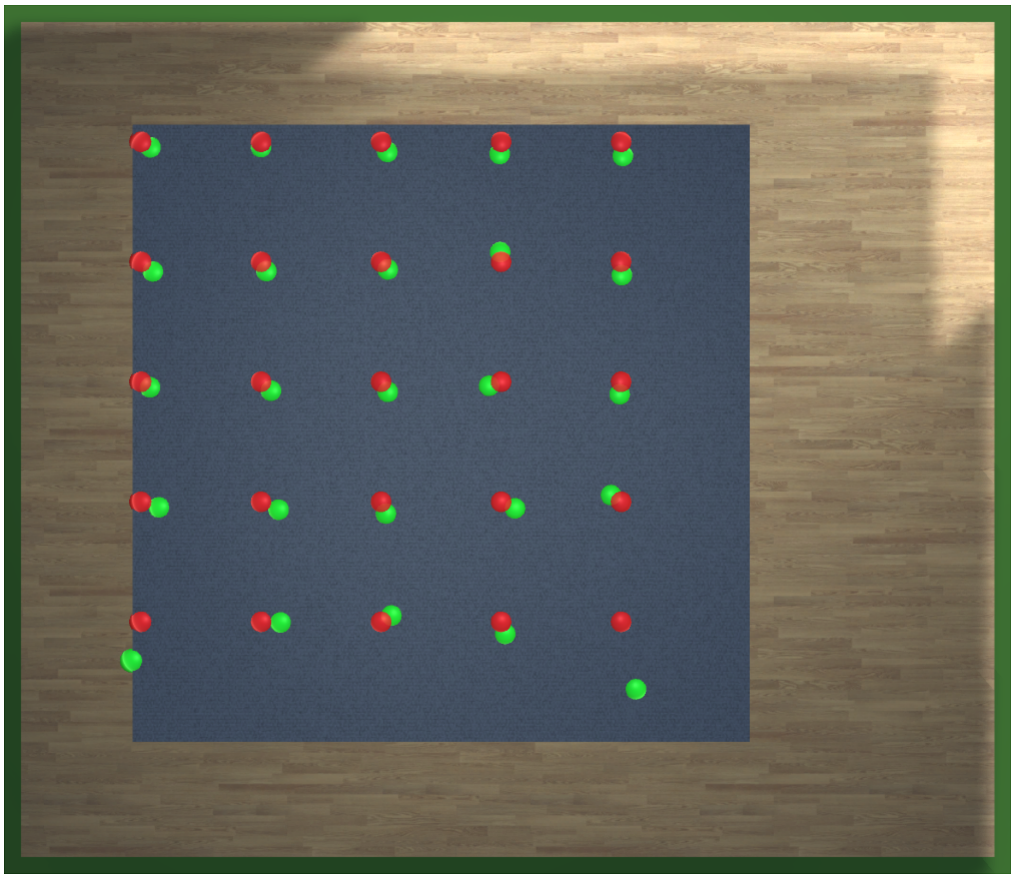

Figure 5.3: Top down view of the virtual room with markings. Red circles are the expected position, and green circles are the triangulated position.

5.2.1 Evaluation Plan

Scenario 1:
'Two listeners watching a movie with bi-lingual soundtrack. Listener A is in a zone with English audio, and listener B in a zone with Danish audio.' This scenario will serve as an introduction to Sound Zones and to get the participants familiar with the interface, it is based on a scenario described by T. Lee, J. K. Nielsen, and M. Græsbøll Christense [11]. Four Sound Zones will be placed on the edges of the room beforehand. Two zones will be connected to a TV and will be preloaded with audio for a movie, a red zone will play Danish audio while a blue zone will play English audio. The two other zones will be Green and Purple, each preloaded with music. The participants will be asked to move the red and blue zone to specific positions in the room, where chairs will be placed. The users will be asked to resize the zones so that they can sit on the chairs and comfortably watch the short movie. Once over, the movie will automatically play again from the start,

participant will be asked to use gesture commands to pause the movie and change seats, then use gesture commands again to resume playing.

Scenario 2:
Scenario two will not require or evaluate users interacting with the system through an interface, instead participants will be exposed to specific situations of interest and their reactions and feedback will be monitored. Two situations will explore the way a Sound Zone should move from one location to another. In the first situation a Sound Zone will move over a predetermined path at a constant speed from one end of the room to the location of a participant and remain on that location for a fixed amount of time. The time the Sound Zone stayed at the location was chosen to be 5 sec, the speed at which the Sound Zone will be moving was chosen to be 1.5 meter per second approximating the average human walking speed. After, the Sound Zone will move to the opposite end of the room, relative to where it started. For the duration of this simulation the Sound Zone will be playing music. In the second situation, a Sound Zone will start in the same corner of the room as during the first one. At the start of the simulation the Sound Zone will instantly move to the location of a participant and remain on that location for a fixed amount of time. The time the Sound Zone stayed at the location was chosen to be 5 sec. After the Sound Zone will instantly move to the opposite end of the room, relative to where it started. For the duration of this simulation the Sound Zone will be playing music. And two situations will explore different degrees of Sound Zones overlapping.

In the first situation, two Sound Zones both playing music will be placed next to each other in such way that their attenuation areas are overlapping. Participants will then be asked to walk between the two zones and describe the experience. In the second scenario, two Sound Zones both playing music will be placed next to each other in such way that they are overlapping. Participants will then be asked to walk between the two zones and describe the experience.

5.2.2 Evaluation Procedure and Setup

All experiments were conducted in the same lab, with approximate size of 7 by 8 meters. An area of 6 by 6 meters marked by a carpet, was the area in which participants could move freely while being tracked by the system. Stepping out of bounds severely limits the tracking as some cameras might lose line of sight. A TV was placed next to one of the walls and two chairs were placed in front of it two meters apart. Chairs were used instead of a couch as they take less space and are easier to move around, which we did a lot between the two scenarios. Additional locations were marked with tape on the ground, those were needed for the second scenario indicating where participants should stand. A table with a computer running the simulation, and a table for conducting the post scenario interviews were placed outside the carpeted area. A rigorous experimental procedure was designed and followed during all tests. Including precautions regarding health concerns. During the experiments both participants and the experiment conductors were wearing face masks in accordance with local regulations, all equipment was thoroughly cleaned with alcohol between tests. Because of the nature of the system, and the way it utilises colour-based tracking, participants were asked to wear different coloured shirts while avoiding the colour black. Black colour was reserved for the experiment conductors and was generally ignored by the system, however additional shirts were available to be worn by the conductors if the situation required it. Participants were first introduced to the project, the concept of sound zones, and what was included in the experiment they were taking part in from a prepared statement See Appendix. Then they were asked to sign a consent form See Appendix. Participants were then asked to stand in the centre of the room to make sure they are detected by the system and there are no issues with the tracking, time was allocated during that part to perform a system calibration if needed. During this step participants were asked to take a walk around the room, following the line of the carpet. If no issues were found participants were introduced to the JBL Soundgear and the app running on Pixel 2 phones. Participants were played a test audio file and were asked to work together and adjust the master volume of the device they were given to a comfortable level, such that they were hearing as little sound as possible from the device used by their partner. Participants were asked to not change the master volume on the device during the tests unless it was necessary. Instead, they were asked to use the interface in the app to control the volume of the Sound Zones.

During the experiment data was collected using four different methods. This includes data logging on the server side, audio recording, User Experience Questionnaire (UEQ), and exit interview. Data logging will be performed on the server side, where each action performed by the participants will be written to a file with a timestamp. The position and size of the Sound Zones will be recorded in standard Unity measurement units. Participant position will be recorded in pixel coordinates as detected by OpenPose, this can later be converted to standard Unity measurement units. The state of the zone will be recorder only when change occurs, e.g. [timestamp] Play ZoneRed [start time]. This will allow for each test to be replayed in the future if necessary. Participants were asked to fill a UEQ and to take part in a short interview after the first scenario. The interview questions were related to their overall experience with the system, both the interface and the hardware. Users were asked for their experience with the JBL Soundgear in particular as there were concerns that audio interference from the device itself could affect the user experience. After each of the four situations explored in the second scenario, participants took part in a shorter interview focusing only on their experience during that specific situation.

5.2.3 Participants

In total 4 participants took part in the experiment. Of the participants 3 where male and 1 female, all between the age of 25 and 32. The participants signed up as dyads and knew each other before.

5.2.4 UEQ Results

UEQ contains 6 scales described by 26 items. Attractiveness shows what was the overall impression the product leave with the participants. Perspicuity measures how easy it is for participants to gain an understanding of the product. Efficiency is a measure of the effort participants have to make to perform a given task. Dependability is how in control the participants feels when interacting with the product. Stimulation shows the motivation of the participant to use and interact with the product. Novelty shows how creative a product is and how good it catches the attention of the participants. The data collected during the first scenario was analysed using the Excel Tool Version 8, and the data from the second scenario was analysed using the Excel Tool Version 4. Both are provided with the questionnaire. The main difference between the two is that Version 4 is directly comparing two products over the 6 scales. While Version 8 provides more details without making a direct comparison. Mean and variance as well as graphs from the analysis can be seen below.

Table 5.2 shows results from the first scenario, the UEQ scales with their means and variance. Figure 5.4 shows a graph of the mean and variance of the UEQ scales shown in Table 5.2.

Figure 5.5 shows a graph of the mean values per item from the UEQ from the first scenario.

Lastly, Figure 5.6 shows a comparison between the Sound Zones moving slowly and the Sound Zone moving instantly to its new location when moved.

The p-values when performing a T-Test on the data from the two situations shows that there is no significant difference in 5 of the 6 scales. The two comparison shows significant difference only on the Novelty scale (p=0.0138).

5.2.5 User Responses

This section will present an analysis of the transcribed interviews and the experience users had with the simulator. In general using the JBL Sound Gear instead of the more compact AirPods Pro was well received by the participants, only one participant stated that the leaking audio from the Sound Gear of their partner was too distracting. It should also be mentioned that according to them the distraction was because there was too much delay between the leak and the sound coming from their Sound Gear. Both groups stated that the Sound Gear was not causing

UEQ Scale	Mean	Variance
Attractiveness	1.250	0.16
Perspicuity	0.375	1.35
Efficiency	0.813	0.10
Dependability	0.188	0.31
Stimulation	1.563	0.06
Novelty	2.000	0.13

Table 5.1: Scenario 1 UQE Scales Mean and Variance

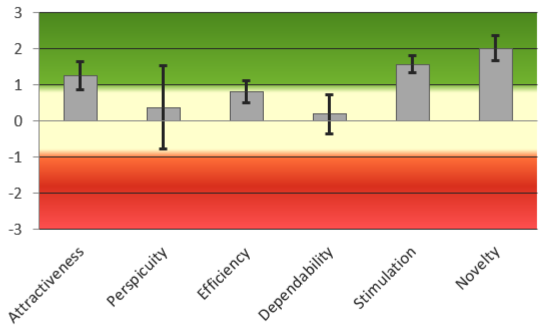

Figure 5.4: Scenario 1 UEQ Scales Mean and Variance

any problems when communicating with each other if there was music playing. However, they also agree that when there was dialog in the movie communication was a little difficult at times. Participants used words like "exciting" and "unique" to describe their experience when walking into a Sound Zone for the first time. One of the groups said that Sound Zones would be perfect solution for them since their desks are next to each other and they often wear headphones when working, but that makes them feel isolated. They would rather be able to talk to each other. Both groups found the app interface easy to learn when it was explained once, but thought that they wouldn't be able to figure most of it on their own. One participant also said "I was worried the zones will get too small for my fingers." Two participants said they prefer using an app to interact with the system. One participant said that for them it depends on the situation and would prefer a hybrid interface. And one participant expressed desire to only use gestures, if they provide similar options as an app. One of the groups had no problems with sharing the available Sound Zones and asked for permission before moving a Sound Zone to a new location. One group also experienced problems with sharing the available Sound Zones. In the words of one participant "You(the other participant) stole my Zone." they later added "It felt like intrusion of my personal space.". That led the participants to express desire to lock Sound Zones so that only they can change

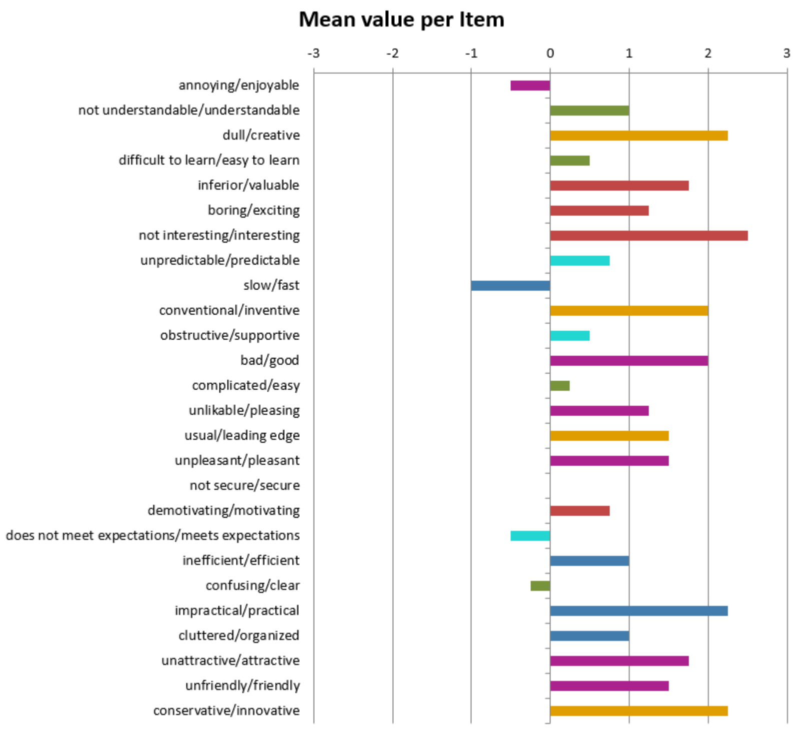

Figure 5.5: Scenario 1 UEQ Scales Mean per Item

them. At least a way to prevent other users from stealing their Sound Zone.

After the participants completed the second scenario this is what they had to say: On participant said that he couldn't tell the difference between the slow and instant moving zones. While the rest agree that a slow moving zone is more natural. "I'm playing a lot of video games and if something appears out of nowhere, I assume it is not working properly." said one participant. "It felt aggressive, I did not like it." said another. All participants agreed that overlapping Sound Zones is a terrible idea.

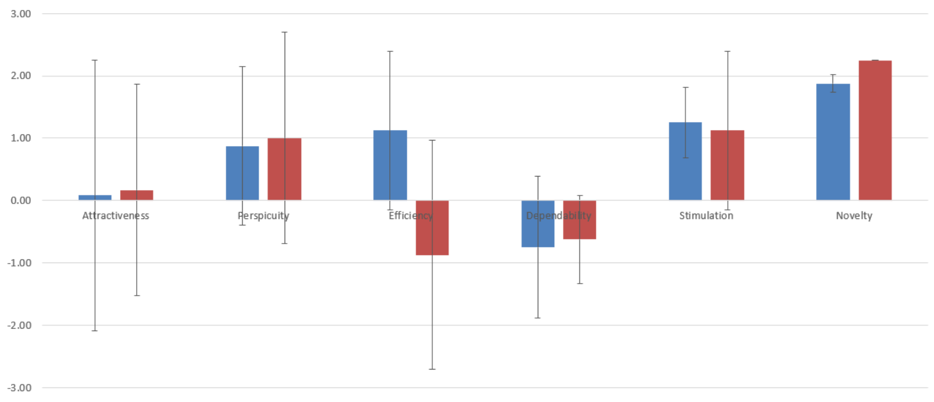

Figure 5.6: Scenario 2 UEQ Scales Comparison between Zones moving slow and Zones moving instantly. Blue is slow movement, Red is instantaneous.

UEQ Scale	value	lambda2
Attractiveness	0.9547	0.30
Perspicuity	0.9121	0.86
Efficiency	0.1363	-0.11
Dependability	0.8615	-0.28
Stimulation	0.8687	0.25
Novelty	0.0138	0.79

Table 5.2: Scenario 2 UQE Scales T-Test and Guttmans Lambda2 coefficient

Chapter 6

Discussion

In this section we discusses the validity and reliability of the collected data and some of the reasons why the sample sizes were so small. We also talk about what we have achieved with this simulator and what are the next steppes.

6.1 User Responses

On of he main takeaways from the evaluation is that while the system managed to pass our own technical tests and full fills all of the requirements set in the design and problem analysis section, we have failed in proving that the system is ready to be used in a real study. We also failed to provide any new data from our study.

The problem with our study is the small sample sizes, while many of our participants agree that Sound Zones should move slowly from point to point, the sample size of 4 is too small and unreliable. Unfortunately because of unforeseen circumstances and regulations regarding certain world events many of participants that we had scheduled sessions with decided to cancel days before. While this can not be used as an excuse, we believe that the two tests we performed show that the Sound Zone Simulator we have designed is more than capable of simulating personal spacial sound for multiple listeners while also allowing allowing them to interact through not only standard User Interface in the shape of the mobile app but also gestures. While currently the system supports a single gesture command that should serve as prove of concept that gestures are a viable interaction technique for Personal Sound.

6.2 Future Work

In our work we tried to develop a system that is modular and allows for other modules to be connected with minimal changes to the main code. Future researchers could use the simulator and expand on the gesture module which proved to be preferred method of interaction but a few of our participants.

Bibliography

[1] Abbey L. Berg and Yula C. Serpanos. "High Frequency Hearing Sensitivity in Adolescent Females of a Lower Socioeconomic Status Over a Period of 24 Years (1985?2008)". English. In: 48 (2011), pp. 203–208. ISSN: 1054-139X. DOI: 10.1016/j.jadohealth.2010.06.014.

[2] Terence Betlehem et al. "Personal Sound Zones: Delivering interface-free audio to multiple listeners". English. In: 32 (2015), pp. 81–91. ISSN: 1053-5888. DOI: 10.1109/MSP.2014.2360707.

[3] "Investigating the Culture of Mobile Listening: From Walkman to iPod". English. In: *Consuming Music Together*. Ed. by Michael Bull, Barry O'Hara Kenton; Brown, and Barry Brown. Vol. 35. 2006, pp. 131–149. ISBN: 978-1-4020-4031-3. DOI: 10.1007/1-4020-4097-0_7.

[4] Marvin Camras. "Approach to Recreating a Sound Field". English. In: 43 (1968), pp. 1425–1431. ISSN: 0001-4966. DOI: 10.1121/1.1911002.

[5] Zhe Cao et al. "OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields". In: *arXiv preprint arXiv:1812.08008*. 2018.

[6] Zhe Cao et al. "Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields". In: *CVPR*. 2017.

[7] Ji Ho Chang et al. "A realization of sound focused personal audio system using acoustic contrast control". English. In: 125 (2009), pp. 2091–2097. ISSN: 0001-4966. DOI: 10.1121/1.3082114.

[8] Jordan Cheer and Stephen J. Elliott. "Design and implementation of a personal audio system in a car cabin". English. In: 133 (2013), pp. 3251–3251. ISSN: 0001-4966. DOI: 10.1121/1.4805230.

[9] Stephen J. Elliott et al. "Minimally radiating sources for personal audio". English. In: 128 (2010), pp. 1721–1728. ISSN: 0001-4966. DOI: 10.1121/1.3479758.

[10] Tao Hu, Sinan Mutlu, and Oswald Lanz. "Multicamera People Tracking Using a Locus-based Probabilistic Occupancy Map". In: *Image Analysis and Processing – ICIAP 2013*. Ed. by Alfredo Petrosino. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 693–702. ISBN: 978-3-642-41184-7.

[11] T. Lee, J. K. Nielsen, and M. Græsbøll Christensen. "Towards Perceptually Optimized Sound Zones: A Proof-of-concept Study". In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019, pp. 136–140. DOI: `10.1109/ICASSP.2019.8682902`.

[12] Ruth Y. Litovsky. "Developmental changes in the precedence effect: Estimates of minimum audible angle". English. In: 102 (1997), pp. 1739–1745. ISSN: 0001-4966. DOI: `10.1121/1.420106`.

[13] Stine S. Lundgaard and Peter Axel Nielsen. "Personalised Soundscapes in Homes". In: *Proceedings of the 2019 on Designing Interactive Systems Conference*. DIS '19. San Diego, CA, USA: Association for Computing Machinery, 2019, 813–822. ISBN: 9781450358507. DOI: `10.1145/3322276.3322364`. URL: `https://doi.org/10.1145/3322276.3322364`.

[14] "Manual on Uniform Traffic Control Devices for Streets and Highways." In: (2003). URL: `https://sfx.aub.aau.dk/sfxaub?sid=sage&iuid=255428&title=ManualonUniformTrafficControlDevicesforStreetsandHighways.&date=2003mutcd.fhwa.dot.gov/`.

[15] Brian C. J Moore. *An Introduction to the Psychology of Hearing*. 5th ed. Amsterdam: Academic Press, 2003.

[16] Diederick C. Niehorster, Li Li, and Markus Lappe. "The Accuracy and Precision of Position and Orientation Tracking in the HTC Vive Virtual Reality System for Scientific Research". In: *i-Perception* 8.3 (2017), p. 2041669517708205. DOI: `10.1177/2041669517708205`.

[17] Atanas Nikolov. *Multi-Modal Interactions for Personal Sound Zone System*. English. Student Project. Aalbourg Universitet, 2019, p. 47. URL: `https://projekter.aau.dk/projekter/da/studentthesis/personal-sound-zone-simulator(4af09694-ac81-4d52-934b-b9bfc2086d4e).html`.

[18] David R. Perrott and Kourosh Saberi. "Minimum audible angle thresholds for sources varying in both elevation and azimuth". English. In: 87 (1990), pp. 1728–1731. ISSN: 0001-4966. DOI: `10.1121/1.399421`.

[19] *Personal Audio Space: The Headphones Experience sans Headphones*. `https://www.microsoft.com/en-us/research/blog/personal-audio-space-headphones-experience-sans-headphones/`. Accessed: 2020-01-01.

[20] Jens Trogh et al. "Bluetooth low energy based location tracking for livestock monitoring". eng. In: *Proceedings of the 8th European Conference on Precision Livestock Farming*. Nantes, France, 2017, pp. 469–475. ISBN: 979-10-699-0991-5.

[21] Shih-En Wei et al. "Convolutional pose machines". In: *CVPR*. 2016.

Appendix A

Appendix A Server Code

```
import socket
import select

HEADER_LENGTH = 10
IP = "127.0.0.1"
PORT = 1234

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

server_socket.bind(('', PORT))

server_socket.listen()

sockets_list = [server_socket]

clients = {}

print(f'Listening for connections on {IP}:{PORT}...')


def receive_message(client_socket):
    # Handles message receiving
    try:
        message_header = client_socket.recv(HEADER_LENGTH)
        if not len(message_header):
            return False
        message_length = int(message_header.decode('utf-8').strip())

        return {'header': message_header, 'data': client_socket.recv(
    message_length)}

    except:
        return False
```

```python


while True:
    read_sockets, _, exception_sockets = select.select(sockets_list, [], sockets_list)
    for notified_socket in read_sockets:
        if notified_socket == server_socket:
            client_socket, client_address = server_socket.accept()

            user = receive_message(client_socket)
            if user is False:
                continue

            sockets_list.append(client_socket)

            clients[client_socket] = user

            print(f"Accepted new connection from {client_address[0]}:{client_address[1]} username:{user['data'].decode('utf-8')}")
        else:
            message = receive_message(notified_socket)

            if message is False:
                print (f"Closed connection from {clients[notified_socket]['data'].decode('utf-8')}")
                sockets_list.remove(notified_socket)
                del clients[notified_socket]
                continue
            user = clients[notified_socket]
            print(f"Received message from {user['data'].decode('utf-8')}: {message['data'].decode('utf-8')}")

            message_data_split = message['data'].decode('utf-8').split(" ")
            if message_data_split[0].startswith('/user'):
                client_target = message_data_split[1]

            for client_socket in clients:
                if client_socket != notified_socket:
                    if message_data_split[0].startswith('/user') and clients[client_socket]['data'].decode('utf-8') != client_target:
                        continue
                    client_socket.send(user['header'] + user ['data'] + message['header'] + message['data'])

    for notified_socket in exception_sockets:
        sockets_list.remove(notified_socket)
        del clients[notified_socket]
```

Listing A.1: Server

Appendix B

Appendix B Simulator Commands

B.1 Intro

Commands start with "/" followed by a word in lowercase e.g. /move. Arguments are separated from the command and each other by a single space e.g. /move ZoneRed (2.9, 0.9, -2.8). Object positions must be sent as Vector3, three floating point numbers surrounded by parentheses. Note that the spelling of the object names is important as the system will search for the exact name entered.

B.2 Local commands

Local commands are executed only on the local device and are not synchronised over the network.

- /data [object name] - The command takes one argument. [object name] – the name of the object data is requested from. Case sensitive. Sometimes it is necessary to know the position of an object in the world. This function will write in the chat the world position of the specified object if it exists.
- /hide [object name] - The command takes one argument. [object name] – the name of the object that is to be hidden. Case sensitive. Hides the specified object from view. In some cases that might stop the object form functioning.
- /username [new username] - The command takes one argument. [new username] – the new username. Changes the username of the device. Only works if not currently connected to a server.
- /colour [object name] [red value] [green value] [blue value] - The command takes three arguments. [object name] – the name of the object which colour will be changed. Case sensitive. Can only be a Sound Zone. [red value]

[green value] [blue value] – RGB values in the range 0-255. Change the colour of the specified Sound Zone.

- /conto [IP address] [port] [username*] - The command takes four arguments. [IP address] – the IP address of the server e.g. 127.0.0.1 [port] – the port to which to connect e.g. 1234 [username*] – optional if change of the username is also desired.Connects to the specified server and port. Optionally, username can also be changed. This can be used to reconnect to a server.

B.3 Global commands

Global commands are executed on all connected devices that are running the companion app.

- /move [object name] [x] [y] [z] - The command takes four arguments. [object name] – the name of the object which will be moved. Case sensitive. Can only be a Sound Zone. [x] [y] [z] – the new position of the object in world coordinates. Move the specified Sound Zone to the specified position.
- /scale [object name] [value] - The command takes two arguments. [object name] – the name of the object which will be resized. Case sensitive. Can only be a Sound Zone. [value] – the new size of the object. Change the size of the specified Sound Zone.
- /smoothing [object name] [value] - The command takes two arguments. [object name] – the name of the object which will be changed. Case sensitive. Can only be a Sound Zone. [value] – the new smoothing value. Change the smoothing coefficient of the specified Sound Zone. Smoothing determines how fast the zone will be moving. Higher value == faster movement.
- /play [object name] [time] - The command takes two arguments. [object name] – the name of the object which will be played. Case sensitive. Can only be a Sound Zone. [time] – milliseconds since the start of the epoch at which the command has been executed on the sending device. Play the audio clip on the specified Sound Zone. The start time will be offset by the latency to ensure all playbacks are synchronised.
- /pause [object name] [time] - The command takes two arguments. [object name] – the name of the object which will be paused. Case sensitive. Can only be a Sound Zone. [time] – at which second of the song the pause command was executed on the sending device. Pauses the audio clip on the specified Sound Zone. The time of pause is synchronised.

- /volume [object name] [value] - The command takes two arguments. [object name] – the name of the object which will be changed. Case sensitive. Can only be a Sound Zone. [value] – new volume value. Change the playback volume of the audio clip on the specified Sound Zone.

- /zone [object name] [value] - The command takes two arguments. [object name] – the name of the object which will be changed. Case sensitive. Can only be a Sound Zone. [value] – set the maxDistance of the specified Sound Zone. Make the specified Sound Zone act as a regular speaker.

- /userpos [username] [time] [keypoint] [data]. . . - The command takes seven to eleven arguments. [username] – the name of the user whose position is send. Case sensitive. [time] – milliseconds since the start of the epoch at which the command has been executed on the sending device. [keypoint] – the keypoint that is send, see OpenPose keypoints for more information. [data] – up to four pairs of (x y) coordinates in camera view. Read the position send from OpenPose and calculate the position in world coordinates. At least views form two cameras must be sent.

- /userinf [username] [gesture] - The command takes two arguments. [username] – the name of the user issuing the command. Case sensitive. [gesture] – the name of the coroutine that will be started. When a user performs a gesture in the real world, the gesture is sent to the device using this format. The gesture cannot carry data, other than the coroutine name itself.

Appendix C

Appendix C Simulation

```
public class CamToRay : MonoBehaviour
{
    private Client1 client;
    private string data_in;
    string pattern = " +";
    Regex re;
    private List<Ray> rays = new List<Ray>();
    public List<Vector3> monitor;

    public Camera[] cams;

    private User user;
    public string Data_in
    {
        get { return data_in; }
        set
        {
            data_in = re.Replace(value, " ");
            //Debug.Log(data_in);
            StartCoroutine(CastRays());
        }
    }

    // Start is called before the first frame update
    void Start()
    {
        client = GameObject.Find("Client").GetComponent<Client1>();
        re = new Regex(pattern, RegexOptions.IgnoreCase);
        user = client.User.GetComponent<User>();
    }

    // Update is called once per frame
    void Update()
    {

    }

    private IEnumerator CastRays()
    {
        rays.Clear();
        // data in format: /userpos {target name} {time of execution} {keypoint} {up to 4 pairs of (x y) coordinates}
        List<string> command = data_in.Split(' ').ToList();
        List<string> data = new List<string>(command);
        data.RemoveRange(0, 4);

        // Check if the command is for this user
        if (command[1] != user.name)
            yield break;

        for (int i = 0; i < data.Count - 1; i += 2)
        {
            //Debug.Log(string.Format("{0}   {1}", data[i], data[i+1]));
            // subtract the y value from 480 because in Unity origin is bottom left
            Vector3 newPoint = new Vector3(float.Parse(data[i], CultureInfo.InvariantCulture), float.Parse(data[i + 1], CultureInfo.InvariantCulture), 0);
            int camID = ReturnImageID(newPoint);
            newPoint = TrimNewPoint(newPoint);
            newPoint.z = camID;
            newPoint.y = 480 - newPoint.y;

            rays.Add(cams[camID].ScreenPointToRay(TrimNewPoint(newPoint)));
            Debug.DrawRay(rays[i/2].origin, rays[i/2].direction * 100, Color.red);
        }

        List<Vector3> closestPoints = new List<Vector3>();

        for (int i = 0; i < rays.Count; i++)
        {
            for (int j = 1 + i; j < rays.Count; j++)
            {
                Math3d.ClosestPointsOnTwoLines(out Vector3 cp1, out Vector3 cp2, rays[i].origin, rays[i].direction, rays[j].origin, rays[j].direction);

                closestPoints.Add(cp1);
                closestPoints.Add(cp2);
            }
        }
        monitor = closestPoints;

        // Move the desired keypoint to it's new location. Further
    actions are taken in the User class.
        // Each keypoint needs to be hard coded in this implementation
    , add the needed functions to the User class.
        user.StopCoroutine(command[3]);
        user.StartCoroutine(command[3], MidPoint(closestPoints));

        yield return new WaitForSeconds(0);
    }

    public Vector3 MidPoint(List<Vector3> ListOfPoints)
    {
        Vector3 sum = new Vector3(0, 0, 0);
        foreach (Vector3 item in ListOfPoints)
        {
            sum += item;
        }
        return sum / ListOfPoints.Count;
    }

    public int ReturnImageID(Vector3 person)
    {
        if (person.x > client.w && person.y <= client.h)
            return 1;
        else if (person.x <= client.w && person.y > client.h)
            return 2;
        else if (person.x > client.w && person.y > client.h)
            return 3;
        return 0;
    }

    public Vector3 TrimNewPoint(Vector3 point)
    {
        if (point.x > client.w && point.y <= client.h)
            return point - new Vector3(client.w, 0, 0);
        else if (point.x <= client.w && point.y > client.h)
            return point - new Vector3(0, client.h, 0);
        else if (point.x > client.w && point.y > client.h)
            return point - new Vector3(client.w, client.h, 0);
        return point;
    }
}
```

Listing C.1: Simulation