Acronyms

VAE Virtual Acoustic Environment . . . 3
VSS Virtual Singing Studio . . . 3
RIR Room Impulse Response . . . 3
first: Click here

Second: Click here

Contents

1 Acronyms **1**
1.1 Software . . . 3
1.1.1 Software Overview . . . 3
1.1.1.1 The Original Patch . . . 3
1.1.1.2 Extended Patch Overview . . . 4
1.1.2 Location Selection . . . 5
1.1.2.1 File name JavaScript: 'loadFilesLogic.js' . . . 8
1.1.3 Mobility Implementation . . . 11
1.1.3.1 Iteration 1 . . . 11
1.1.3.2 Iteration 2 . . . 12
1.1.3.3 Iteration 3 (Final) . . . 13
1.1.4 Location Tracking . . . 18
1.1.4.1 User Interface . . . 19
1.1.4.2 Position Feedback . . . 20
1.1.5 Head-Tracking . . . 21
1.1.6 Settings Menu . . . 22
1.1.7 Software Issues . . . 22

1.2 Latency 23
1.3 RIR Trimming 24

Appendices **26**

A Appendix A **26**

B Appendix B **30**

C Appendix C **32**

D Appendix D **35**

E Appendix E **41**

1.1 - SOFTWARE

As a software patch was already in use with the Virtual Singing Studio (VSS), the idea was to extend this software patch to accommodate the newly proposed functionality. This section will first give a quick overview of the original max patch and then an overview of the newly produced software with a simple explanation of how it works, followed by a more detailed explanation of how the two main parts of the software work.

The software described was run on a 2012 Mac mini with 16GB of RAM, an IntelCore i7 processor running OSX 10.10.4.

1.1.1) Software Overview

1.1.1.1 The Original Patch

The original max patch was used to convolve a real time audio signal with a set of four Room Impulse Response (RIR)'s simultaneously, allowing the user to turn their head in the Virtual Acoustic Environment (VAE) through the use of an Oculus Rift as a head tracking device. Four positions within the VAE were available. To select one, the user (or an operator) selected an 'open' button which prompted a file navigation window. The RIR files then had to be found (in the correct order) and opened one at a time, with a new file window opening after each file had been selected. A screen shot of this process is shown in figure 1. This was the primary aspect of the original patch that needed extending to make the process automatic.

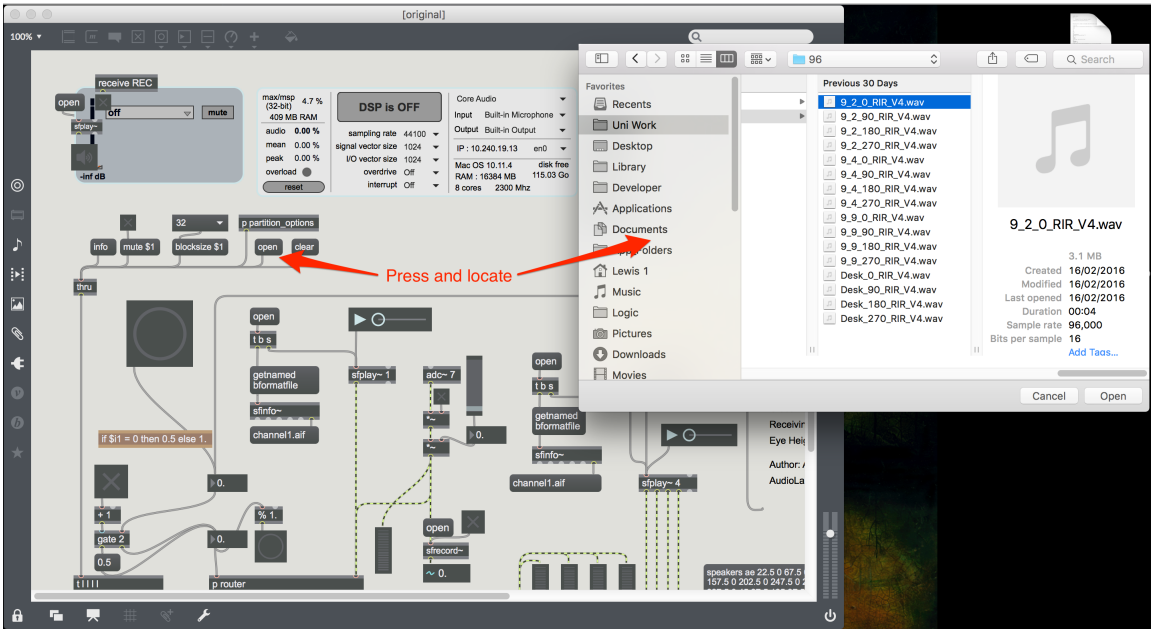


Figure 1: Original Max patch used in the VSS showing the file location window that pops up 4 times.

1.1.1.2 Extended Patch Overview

Figure 2 shows an annotated top level view of the Max patch produced to take a user input, load the appropriate RIR files and convolve with a real time audio input. The annotated sections can be described as follows:

1: Two buttons used to reset the system and start the timer used when moving around the space. The settings patch extends to provide a range of options including the density of

2: Here, an audio file can be loaded into the system and used instead of a real time audio input. This is used in user test #3.

3: A timer system used to synchronise the output of data transfer (from **4.2** to **4.3**), panning between convolved audio and the movement of the user interface feedback system (see section Location Tracking).

4.1: Patches that send a user interface to an iPad which allows a user to select a location within the VAE. User interaction is monitored (in the form of screen coordinates) and sent to **4.2**.

4.2: Takes the coordinates of the user input and calculates which (if any) RIR file should be loaded into the system, in section **4.3**.

4.3: Three patches (extended versions of the max explained in section The Original Patch) used to simultaneously convolve an audio signal (real time or audio file) with four directional RIR files. While one loads the next necessary RIR file the other two are used to simulate the movement of the user by panning the real time audio between the two currently running convolutions. This is how the user is moved along a path, explained in section Iteration 3 (Final).

5: An extended version of the real time head-tracking system used in the original patch, used to pan between the four directional RIR's to simulate head movement in the VAE.

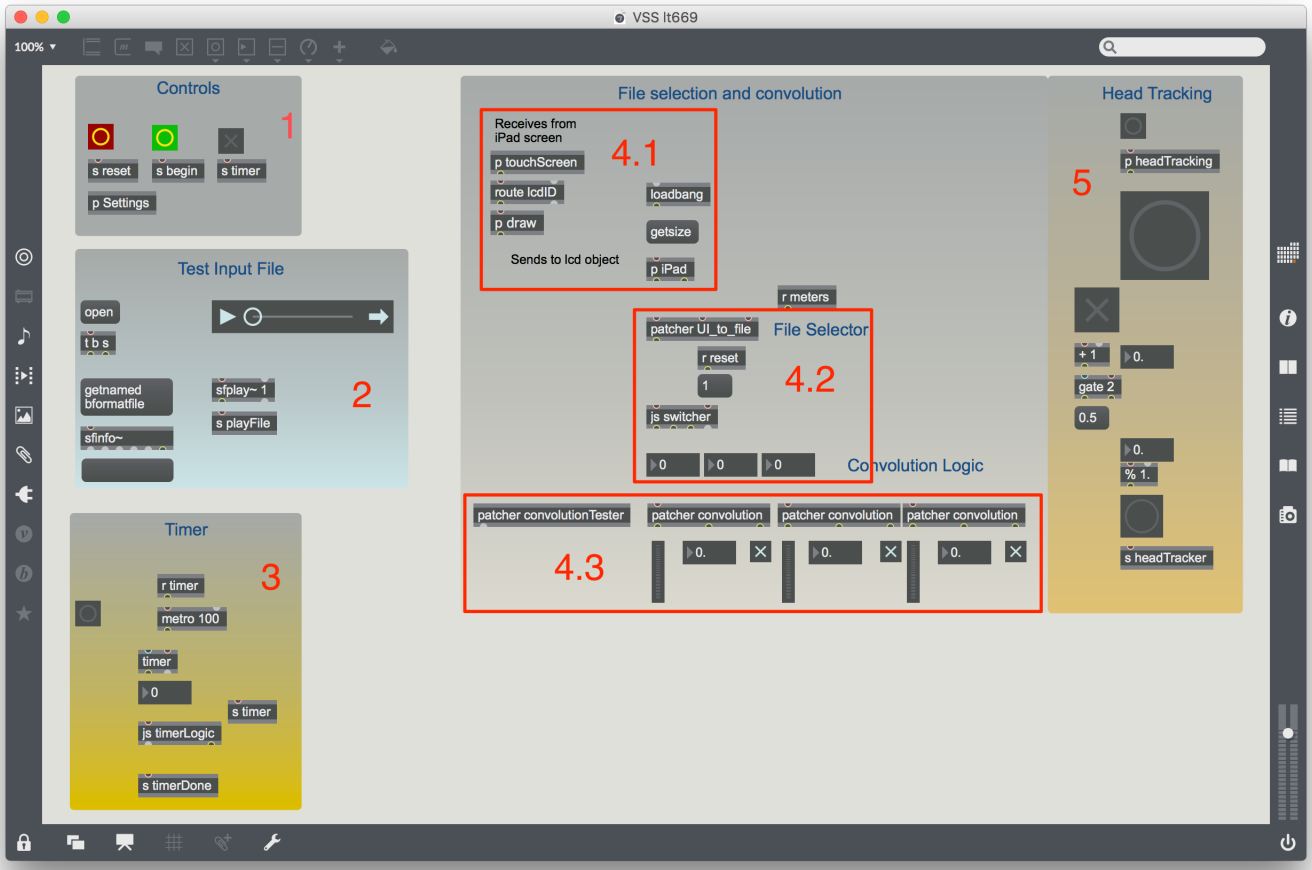


Figure 2: Top level Max patch

The software implementation was split into two main sections: **Location Selection** consisting of section **4.2** and **Mobility** mainly consisting of section **3 and 4.3**.

1.1.2) Location Selection

In order to allow the user to move themselves around the room, they had to be presented with a means of doing so. This involved presenting the user with an interface that resembled the space available to them on which they could select their location. This then had to output the coordinates selected by the user and convert them into a format which can be interpreted as a file name indicating which RIR in the available grid to use. This could then be passed to the rest of the system to load the appropriate files. A simple block diagram of the user interface part of the system is shown in figure 3.

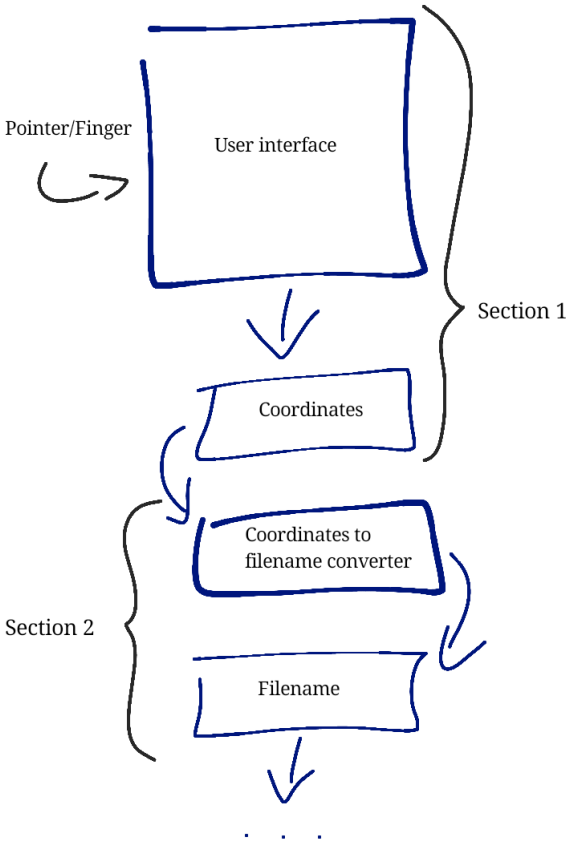


Figure 3: Flow diagram of the location selection software design. **Section 1** indicates the user interface section where coordinates are recorded and **Section 2** takes these coordinates and finds the appropriate RIR file.

In Max, the 'lcd' object is used for this function. This object presents a quadrilateral of variable length and height with the ability to output its dimensional information by sending a 'getSize' message to its input, as well as output the coordinates of a mouse click/drag. Figure 4 shows the lcd object with its inputs and outputs represented by section 1 in figure 3.

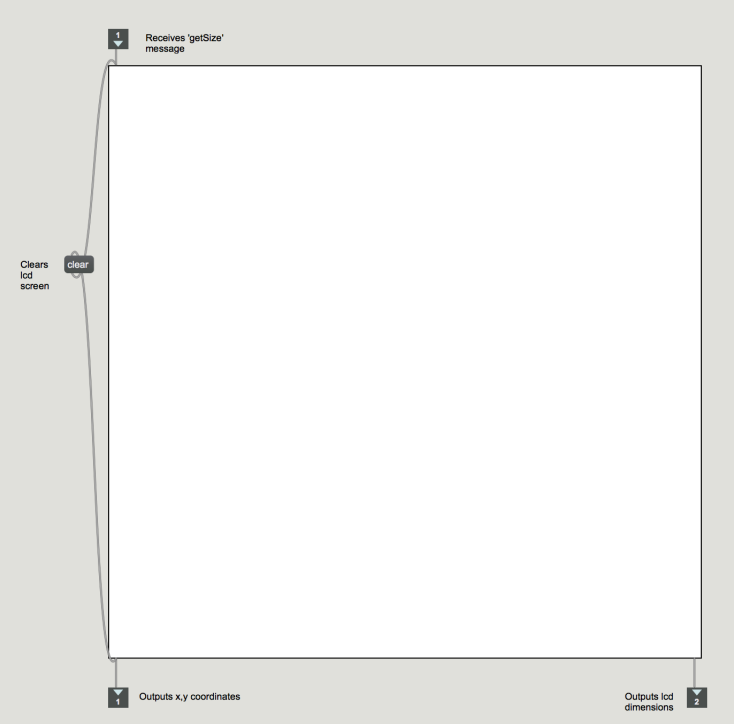


Figure 4: Flow diagram of the location selection software design

The outputs from the lcd object are sent to the patch 'UI_to_file' which contains a JavaScript file called 'loadFilesLogic'. This JavaScript file converts the (x,y) coordinates into an appropriate file name, by taking into account the size of the lcd screen and how many RIRs there are per meter.

This lcd screen could also be displayed on an iPad which was used as a remote user interaction. Initially, the more popular and supported application **Mira** [1] was going to be used. This allows for the mirroring of a selected portion of a Max patch to be displayed and interacted with on an iPad. However, the lcd object used to track coordinates was not supported by this application and thus did not appear on the iPad. Instead an alternative was found. Cycling74, the same company that produces Max/MSP and Mira also created an app for the iPhone and iPad called c74 [2] and allows for the creation of an independent interface that can control objects within the max patch. The operation of this section is explained further in section Location Tracking

The javascript takes 5 inputs: UI (x,y) coordinates , (x,y) lcd dimensions and a number representing how many rows and colums of RIR locations there should be available. This last value is calculated by sending a number from 1-5 (distance per RIR) to the third inlet of the UI_to_file patch (on the far right). As the maximum number of RIRs that will be available per length of the room is 15, 15 is divided by the input and rounded to the nearest integer, giving the number of rows and columns the lcd screen should be split into (the exact number of rows and columns is calculated later in the javascript file). This information can then be used to determine in which *'section'* (row and column) the user is currently located based on their coordinates, allowing it to load the appropriate RIRs that are available in that section. Figure 5 shows the section of

'UI_to_file' highlighting each section.

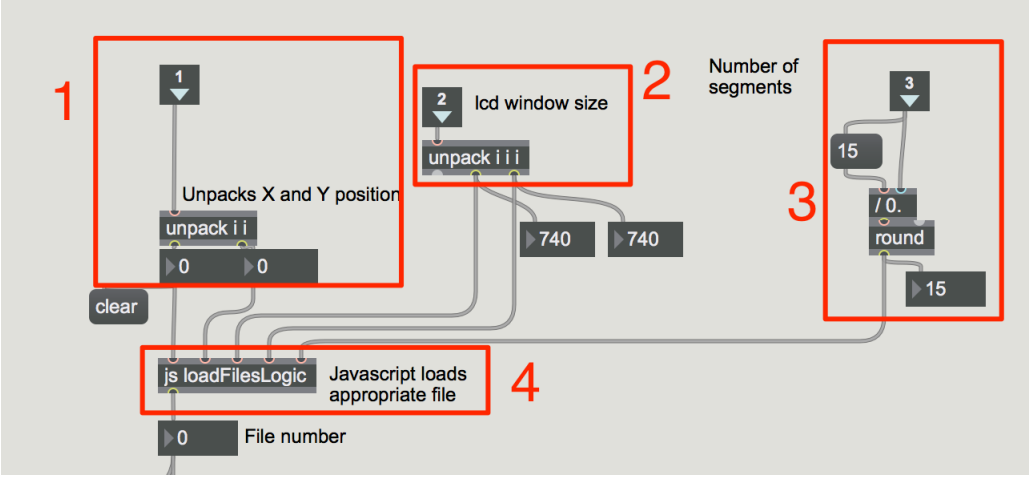


Figure 5: Screen shot from Max showing a section that converts the user interface coordinates into the appropriate file name. 1) Coordinates from lcd object 2) Dimensions of lcd object 3) Number of segments the lcd object should be split into due to the number of RIR's per meter 4) JavaScript file that produces an appropriate file name given the input data.

1.1.2.1 File name JavaScript: 'loadFilesLogic.js'

```
function msg_int(input){
 if(inlet == 0){
  xPos = input;
 } else if (inlet == 1){
  yPos = input;//Add off set to start at (0,1)
 } else if (inlet == 2){
  windowSize [0] = input;
 } else if (inlet == 3){
  windowSize [1] = input;
 } else if(inlet==4){
  numberOfMeters = input;
 }

```

The first section in the JavaScript simply stores the data from different inputs to different variables that are used throughout the rest of the code, where:

xPos	= user x coordinate
yPos	= user y coordinate
windowSize[0]	= lcd length
windowSize[1]	= lcd height
numberOfMeters	= Integer to calculate rows/columns

As can be seen in figures 25,26,27 and 28 in Appendix B, there are not always the same number of rows as there are columns in each of the grids. The if else statements in figure 6 calculate the users positions by taking the users x and y coordinates and dividing by the x and y dimensions of the lcd screen, giving a percentage of how far across the room they are. This allows the for lcd screen to be resized allowing it to be used of different sized screens in the future (see section **??**). This value is then multiplies by either the number of rows (x axis) or columns (y axis) there are, which is calculated by adding or subtracting 1, or using the value of numberOfRows which is calculated

before being input into the js object. The following table indicates how many rows and columns there are given the inputs to the js object:

Distance between RIR's (m)	Calculation	numberOfMeters	Rows	Columns
1	15/1	15	15	16
2	15/2	8	7	8
3	15/3	5	5	5
4	15/4	4	3	4
5	15/5	3	3	3

Lines 17 and 18, the calculated positions are rounded to the nearest integer in order to determine which RIR location the user is closest too. Lines 21 to 26, create an initial offset to prevent the lcd screen from starting at location (0,0), as this is not a physical point in the room itself thus preventing errors from occurring when searching for the appropriate RIR file.

```
1   //Split into sections
2   if(numberOfMeters == 3 || numberOfMeters == 5){
3    //Even grid for 3m and 5m
4    xPosition = (xPos/windowSize[0])*(numberOfMeters);
5    yPosition = (yPos/windowSize[1])*(numberOfMeters);
6   } else if (numberOfMeters == 4 || numberOfMeters == 8){
7    //4m separation requires different x,y coordinate scaling
8    xPosition = (xPos/windowSize[0])*(numberOfMeters-1);
9    yPosition = (yPos/windowSize[1])*(numberOfMeters);
10  } else{
11   //Extra row for others
12   xPosition = (xPos/windowSize[0])*(numberOfMeters);
13   yPosition = (yPos/windowSize[1])*(numberOfMeters+1);
14  }
15
16  //Round to nearest value
17  xSection = Math.round(xPosition);
18  ySection = Math.round(yPosition);
19
20  //Start the lcd grid sections from column 1 row 1 instead of column 0 row 0
21  if(xSection == 0){
22   xSection = 1;
23  }
24  if(ySection == 0){
25   ySection = 1;
26  }
27
```

Figure 6: Code used to calculate user location

The section the user is currently located in is then saved to the first address in a array and can be used to compare against their previous position to check whether a new file needs to be loaded.

This is necessary because each time the user touched the user interface, thus changing the x and y coordinates, javascript file is run. However, if there new location is still within the same 'section' then a new RIR file does not need to be loaded.

In figure 7, lines 6 to 26 contain one large if statement. This essentially prevents the program from calculating, searching for and trying to load a file if the same one is still in use, thus saving computation time. This is done by comparing the section of the grid the user is currently in with the previous section they were currently in and if it has changed the file name for the new location should be calculated and searched for, otherwise no new file should be calculated.

If the location of the user has changed, two more conditions are checked in lines 8 and 14. This ensures that only the section that has changed is updated, ie, if the user has moved to the left, only the X axis section is updated.

Finally, the appropriate file name is calculated. Two different algorithms are used depending on which RIR grid is being used. Both essentially take the section the user is located on the X axis, and add the number of locations there are due to how many rows down the room they are located.

This then outputs a number from 1 to the maximum number of locations in the selected grid.

```
1     //Store current location
2     xArray[0] = xSection;
3     yArray[0] = ySection;
4
5     //If either coordinate is changed search for new files
6     if(xArray[0] != xArray[1] || yArray[0] != yArray[1]){
7
8      if(xArray[0] != xArray[1]){
9       //Store previous value
10      xArray[1] = xArray[0];
11      X = xArray[0];
12     }
13
14     if(yArray[0] != yArray[1]){
15      yArray[1] = yArray[0];
16      Y = yArray[0]
17     }
18
19     //Output user location within grid
20     if(numberOfMeters == 4 || numberOfMeters == 8){
21      fileNumber = X + ((numberOfMeters-1)*(Y-1)); //Requires different algorithm for 2
    m and 4m due to different grid shapes
22     } else {
23      fileNumber = X + ((numberOfMeters)*(Y-1));
24     }
25     outlet(1,fileNumber);
26    }
27
```

Figure 7: Code used to search for appropriate file name

1.1.3) Mobility Implementation

The following sections describe two earlier iterations of the systems that were attempted and the issues faced, eventually leading to the third and final iteration which can been seen as a compromise between system speed and desired functionality. All three iterations are extensions of the original patch used in the VSS which use spat to load RIR files into the system, and to convolve a real time audio input with them.

1.1.3.1 Iteration 1

Initially, the idea was to pre-load a grid of the closest RIR locations that surrounded the user and simultaneously convolve the real-time audio with all of the RIR files. This is illustrated on the left in figure 8 which shows annotated grid positions on the lcd screen labeled 1 - 9. The patch on the right shows the corresponding volume bars (also labelled 1 - 9) used to automatically vary the

output level of each of the convolved signals. This required a panning algorithm for each of the positions in the defined grid, an overview of which is shown in figure 29 in Appendix C, based on where the user has moved relative to the centre position essentially allowing the user to move freely.

When the user reaches on of the other locations in the grid, that location becomes the centre of a new grid, and the appropriate files are loaded around that centre positions through the use of a javascript file. This requires the system to simultaneously load 4 RIR files per location, of which there are 9, meaning 36 files are loaded into the system at once. Due to the time taken to load all of the files, the system ran too slow and could not be used for real time movement.

A video demonstrating the functionality of the proposed system can be found link **HERE LINK!: Writeup/videios/iteration1Panning**.

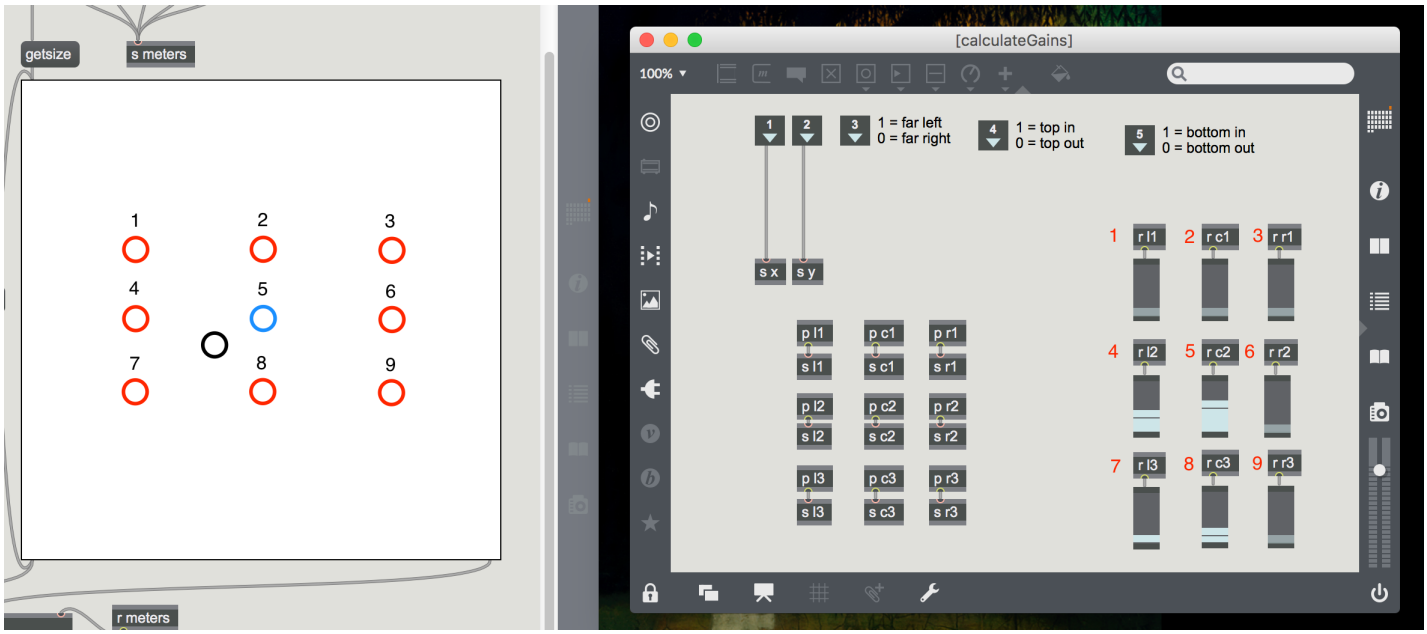


Figure 8

1.1.3.2 Iteration 2

In an attempt to maintain the current implementation and solve the file loading time issue, a patch was built to pre-load all the RIR files, meaning the files only had to be associated with a grid position as opposed to loaded every time a new position was used. This involved running a convolution patch for each file loaded into the system. As the files would no longer need to be loaded into the system, the user interface section described in section Location Selection is used to decide which of the convolution patched to bypass and mute, and which ones to run. This means that only the grid of RIR's being used would be convolved with the audio signal, saving computational resources. Figure 9 shows part of the patch used to pre-load all the RIR files.

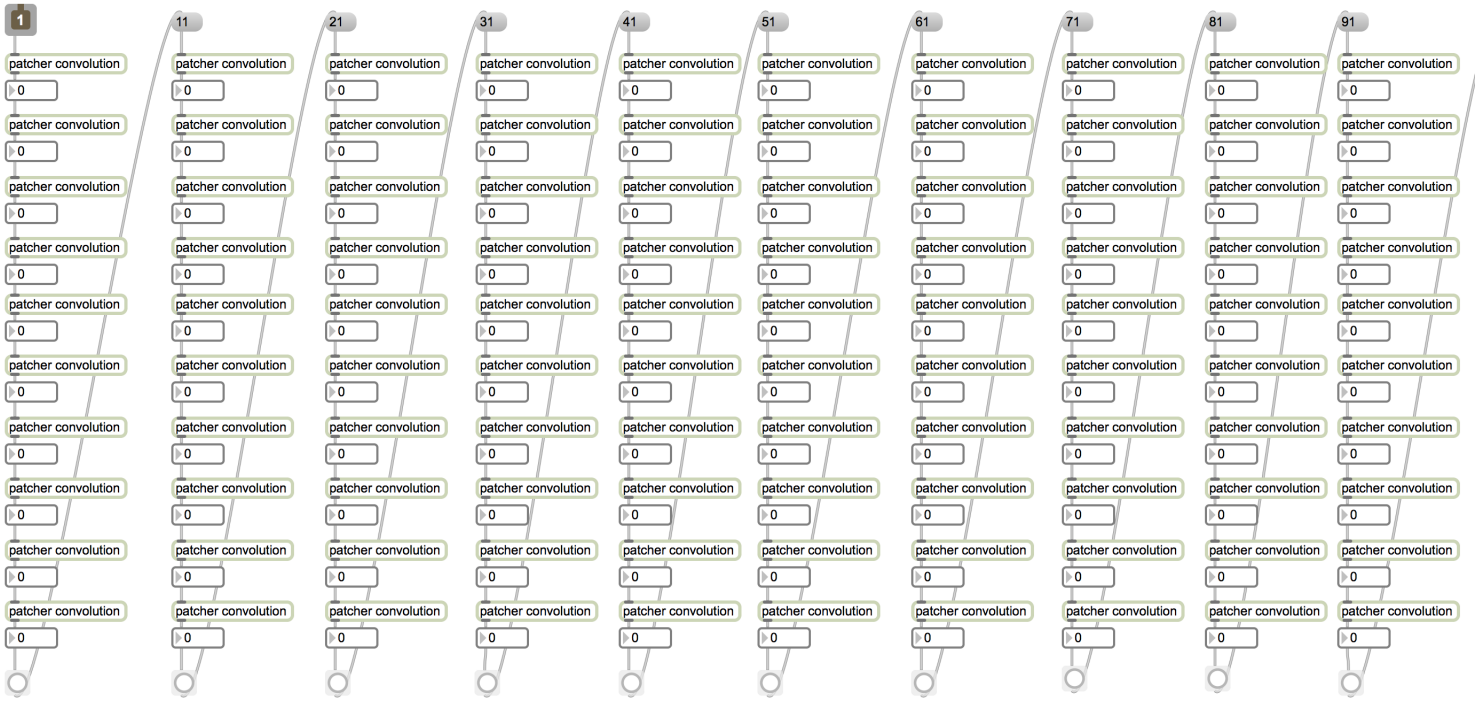


Figure 9: Chained convolution patches used to pre-load each RIR file. *Image taken from Max 6 as the version of this patch would not run in Max 7*

Though it was possible to load smaller grids, such as a 9x9 grid used when locations are separated by 5m, the larger grids caused the system to run too slowly due to the amount of RAM that was used.

1.1.3.3 Iteration 3 (Final)

The previous iterations attempted to allow the user to move anywhere within the VAE in real time as they moved their finger around the screen, however due to technological implications, this was not possible. It was therefore decided to find a compromise, where the user could draw themselves a path around the space and the system would move them along the path, one RIR location at a time at a rate determined by a timer, allowing the system the load all appropriate files with ample time to prevent the system from lagging.

The top half of figure 10 has been seen already in figure 5, however the bottom of figure 10 shows how the resulting file number is used. It gets sent to a javascript file that adds the file number to the end of a mutable array (an array that can be extended or truncated once created). As the file names are being stored, the corresponding files can be loaded when the system is ready, instead of trying to load them instantly as in Iteration 1. Once the js object receivers a message from the timer (highlighted as number 3 in figure 2), the number at the start of the array is sent to the output. The array is then truncated by 1, moving the next file number to the front of the array waiting for the next timer message. If a reset message is sent to the third inlet of this js object (by

pressing the first button in section 1 shown in figure 2), the arrays are cleared enabling the user to start drawing another path if the previous path had not been completely travelled.

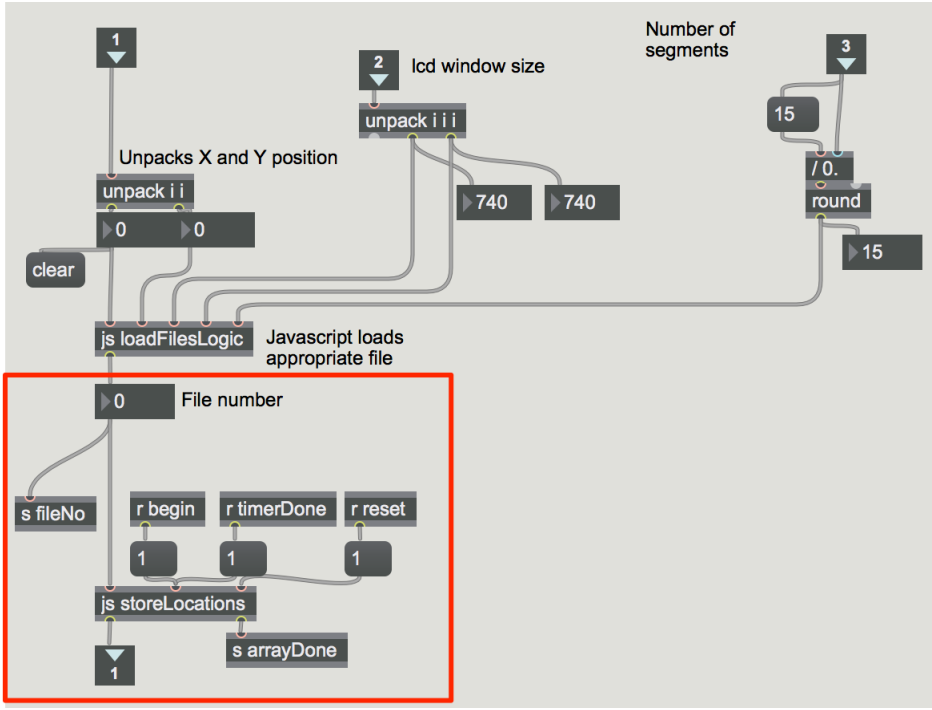


Figure 10: Highlighting the section of 'UI_to_file' that stores the file numbers and outputs them when a timer is done.

Figure 11 shows the output from the 'UI_to_file' patch in the red rectangle (the patch that contains the contents of figure 10) running into a new javascript called 'js switcher' in the blue rectangle. This simple script ensures that each new file number is sent to a different convolution patch (shown in the yellow rectangle). This means that while a new file is being loaded into one of the convolution patches, the other two can be used to pan between two pre-loaded positions. This is illustrated in figure 12

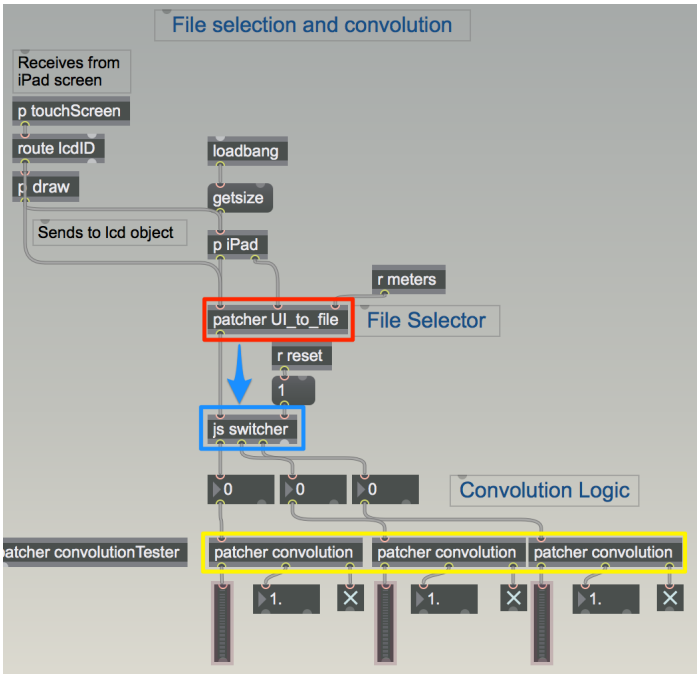


Figure 11: **Red:** Outputs file stored file number. **Blue:** Distributed file number to a different convolution patch each time. **Yellow:** Receives file number and loades appropriate file into the system.

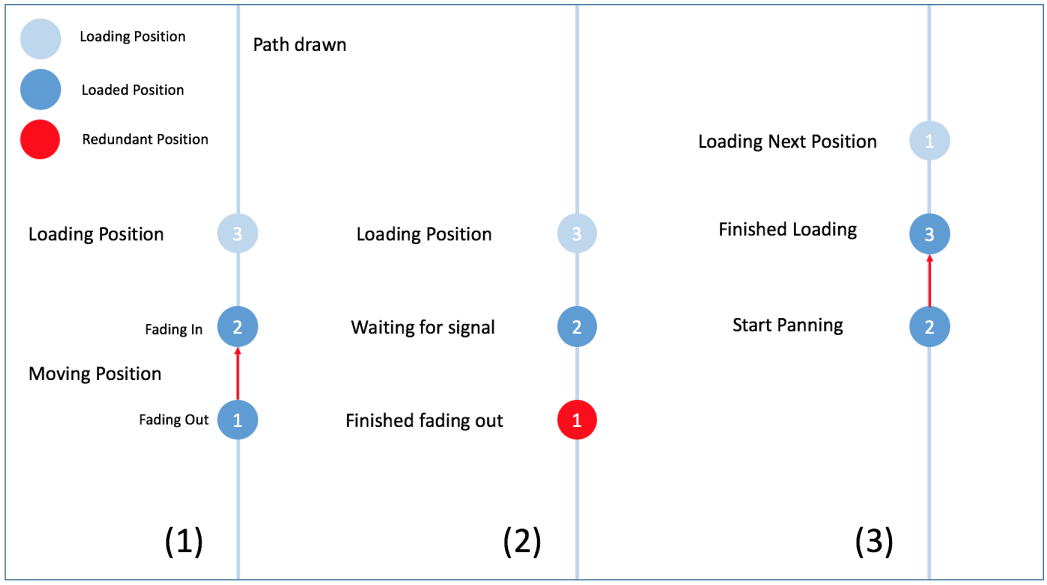


Figure 12: Illustration of how the three convolution patches work together to simulate movement, where the circles represent locations within the VAE and are numbered to indicate which convolution patch they are loaded in. **(1)** User is moved between two loaded positions while the third patch loads a new RIR fie. **(2)** The user is held in the new position until the next position has been loaded into the system. **(3)** User is moved between the two available positions while the next position is loaded into the system.

The convolution patches have three outputs connecting to:
1: A level meter used to see which patch is being used.
2: A number box to check the level of each patch.
3: A toggle box used to see which spat convolution algorithms have been bypassed (a function that was later removed, explained in section Software Issues) These were used to monitor the functionality of the patch while testing.

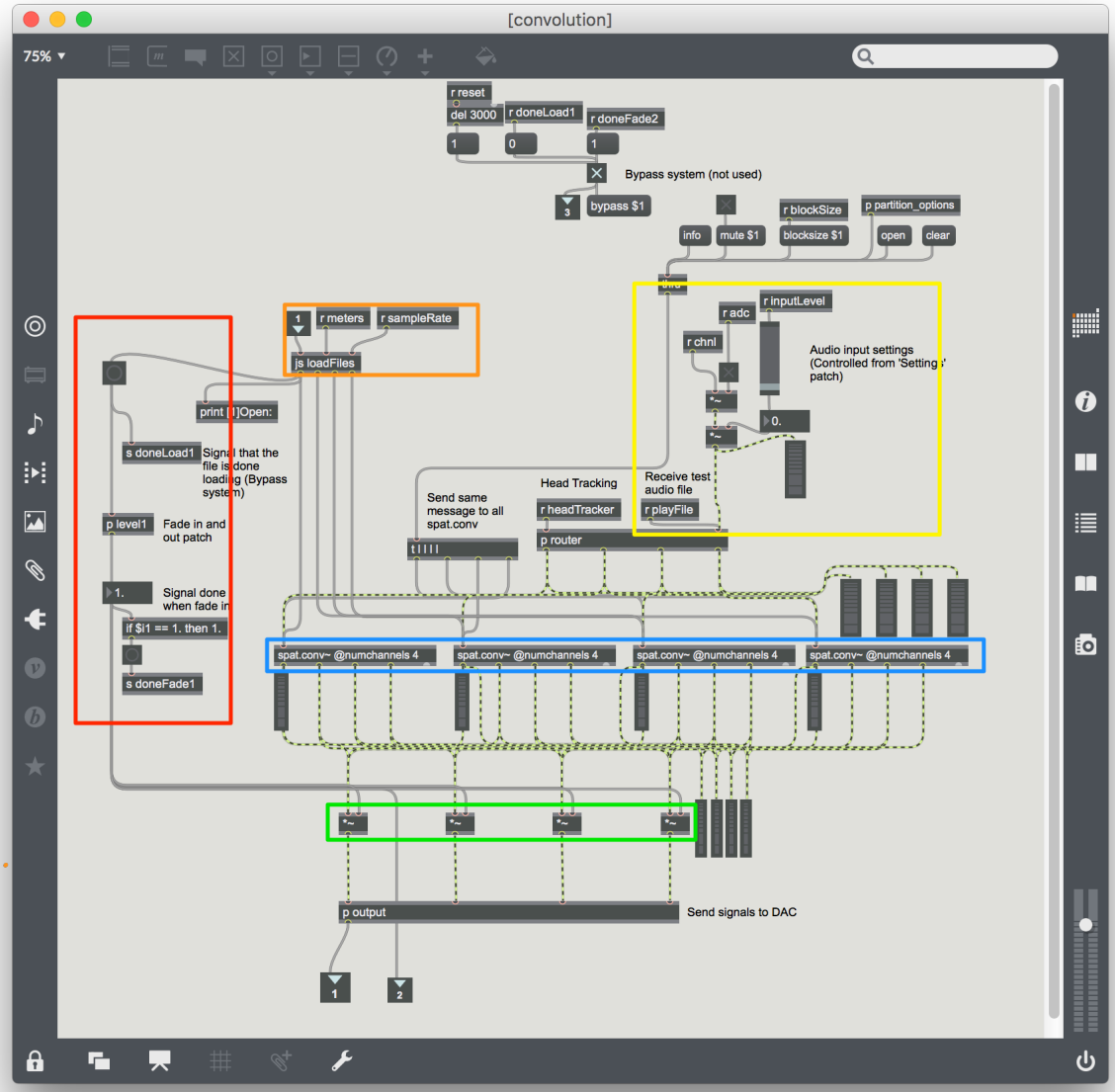


Figure 13

To achieve the path following functionality, each convolution patch contains its own level control algorithm. Figure 13 shows an overview of the first convolution patch which is a modified version of the original pathch used in the VSS. The input to the convolution patch (orange) feeds the file number directly into a 'loadFiles.js' javascript object. This simply prepends the number with the appropriate number of 0's (eg 1 becomes 001 and 28 becomes 028), then this number is concatenated with a file path pointing to where the audio files are located (see js file: loadFiles.js **LINK**). This outputs four 'open' messages (one for each directional RIR file) followed by a file paths to the spat.conv objects (blue). These are the objects that actually search for the file and load it into the system. By sending the prepending 'open' message, manually searching for a file is

not required, thus automating the process. These objects convolve the loaded RIR file with whatever audio input is given at its inlet, in this case the real time audio input or audio file (yellow). These outputs are then sent through multipliers (green), used to fade the signal in and out with an automated volume control (red).

The automated volume controls receives a 'bang' (signalling something has happened) when the 'open' message is sent form the 'loadFiles' js object (orange). This bang is also sent to the convolution patch that is currently at full volume. This prompts the volume of the current patch to increase while the volume of the previous patch decreases, thus panning between the two signal.

1.1.4) Location Tracking

Figure 14 shows the patch 'touchScreen' which contains two main elements. The left side is used to draw a dot on the user interface that indicated to the user where they currently are in the virtual space and the right side is used to draw the user interface. Both are described in the following sections.

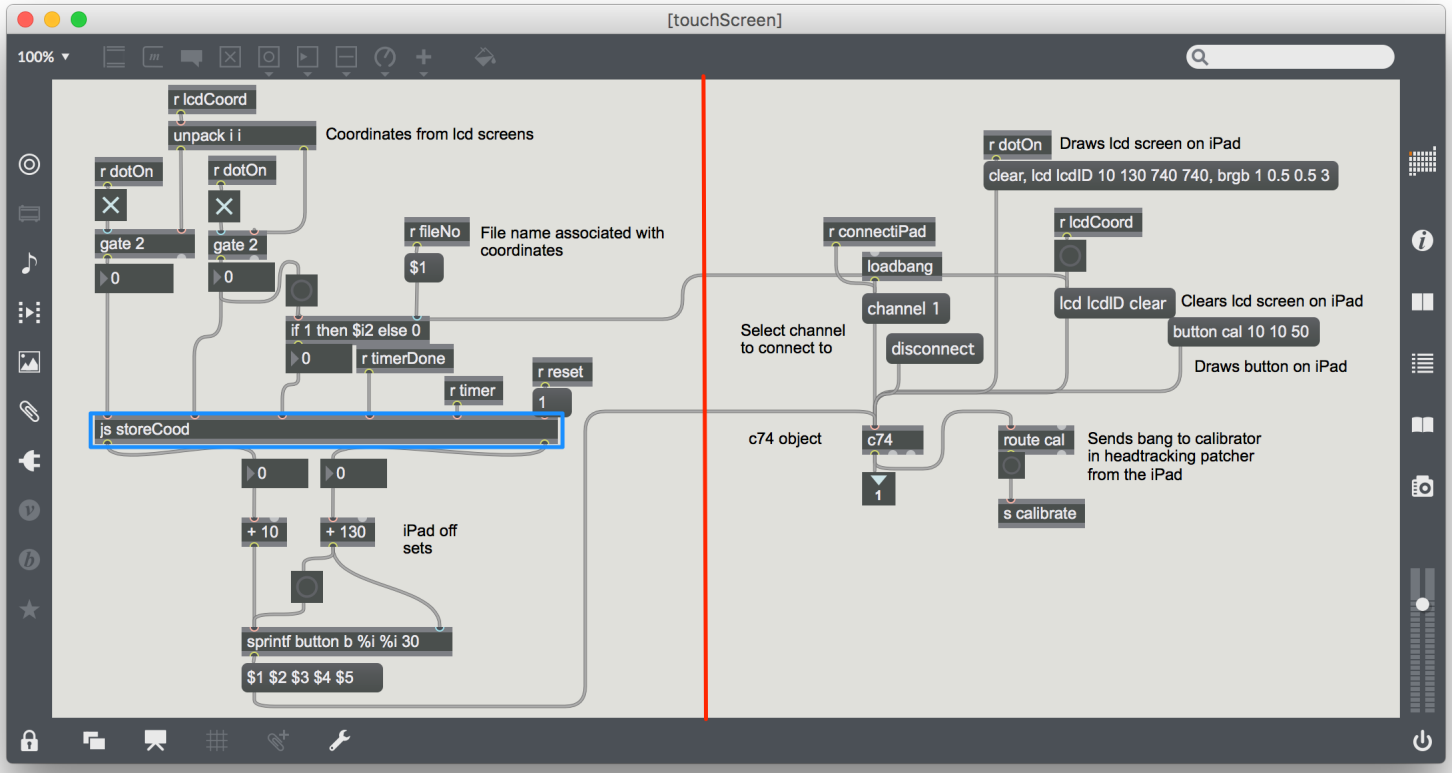


Figure 14: Screen shot of 'touchScreen', the patch used to produce the user interface on the iPad (right side) and to draw a dot on the iPad showing the user where they are located in the virtual space (left side). 'storeCood' javascript object highlighted in blue.

1.1.4.1 User Interface

As previously mentioned, the c74 application was used to display a user interface on an iPad, allowing the user to select their location within the space. This is created on the right side of figure 14, by sending the 'c74' object a message that contains the type of object that should be presented, its coordinates in the space as well as size and colour, the results of which are shown in figure 15. The output of the 'c74' object returns information regarding the objects created in the user interface. In this case, the positions touched on the lcd screen will be sent from the iPad back to this 'c74' object. This information is sent to the input of the 'UI_to_file' patch. This allows the iPad screen to be used the same way as the lcd screen in Max is used (show previously in figure 4).

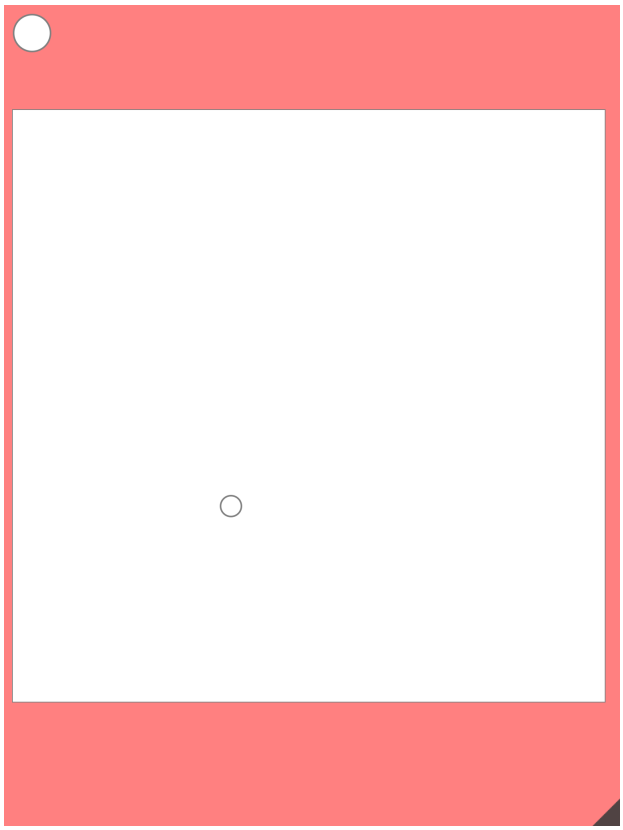
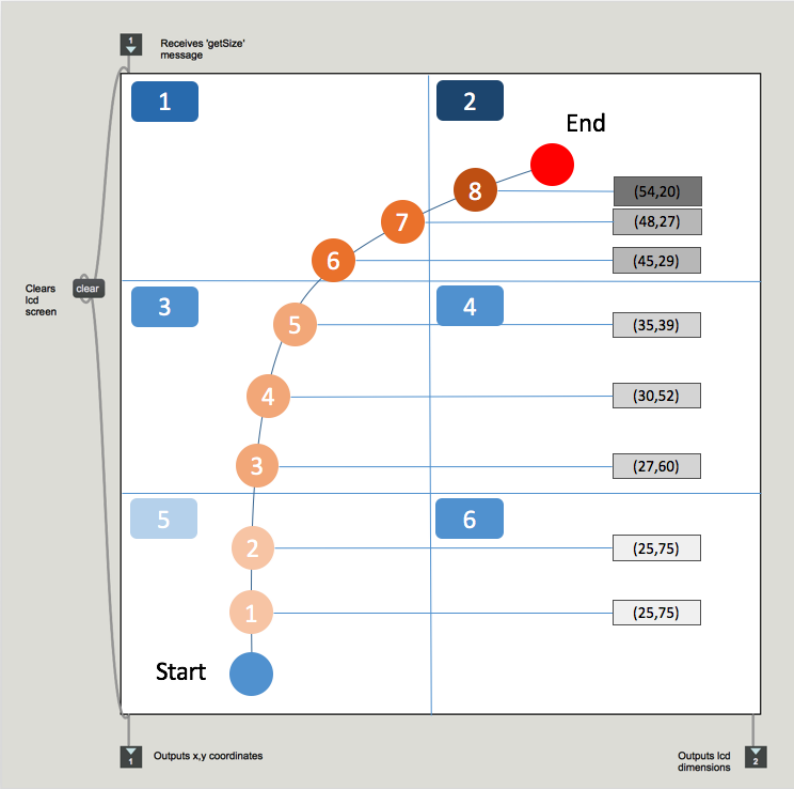


Figure 15: Screen shot of the user interface produced on the iPad. The large white rectangle represents the virtual space in which the user can place themselves, where the top of the rectangle is the front of the room (where the blackboards are in Hendrix Hall). The dot in the white rectangle is used to show the user where they are located in the space and the button in the top left corner is used for calibrating the head-tracking device.

1.1.4.2 Position Feedback

The left side of figure 14 contains a javascript file called 'storeCood' **Available here: LINK** (highlighted in blue) with a number of inputs. As mentioned above, the 'c74' objects can retrieve data from the lcd screen on the iPad. When the user selects a location, the (x,y) coordinates are sent to the first two inputs of the javascript file respectively. These coordinates are stored in a list along with the corresponding section that they are located in (third input), calculated using the method described in section 1.1.2 Location Selection. Figure 16 shows a simple example, where the coordinates of a path spanning across 4 sections are stored in groups.

Input 4 and 5 receive two different signals routed from the main timer (labelled 3 in figure 2). Input 4 receives a signal every time the timer is done (ever 2.5 seconds) and input 5 receives a message every 100ms (the rate at which the timer increments). These are used to call the two main functions that are used to output the coordinates at the correct times, `findLength()` and `outputCoords()`. Each time the timer finishes (after 2.5 seconds), `findLength()` starts from the beginning of the list of stored coordinates and scans through them, calculating how many coordinates are located in the same section. For the example in figure 16, this would find a length of 2 for positions 1 and 2 located in section 5. As the user is moved to a new RIR location every 2.5 seconds, `outputCoords()` will output the coordinates that were travelled in that section evenly within that amount of time. This is done by dividing 2500(ms) by the number of points within that section. For the example in figure 16, this would be $2500/2 = 1250$. Therefore, every 100ms the function checks to see whether it is time to output the stored coordinates, meaning the coordinates of position 1 and 2 will be output at times 1250ms and 2500ms. If we were to continue with the example, the next three positions, 3, 4 and 5 would be output at times 833ms, 1666ms and 2500ms, thus evenly spreading out the movement of the dot on the user interface screen *meaning the dot should travel around the path at a continuous rate instead of in staggered increments*. An example of the dot movement can be found **LINK TO DOT MOVEMENT VIDEO**.



Point	Section	(x,y)
1	5	(25,80)
2	5	(25,75)
3	3	(27,60)
4	3	(30,52)
5	3	(35,39)
6	1	(45,29)
7	1	(48,27)
8	2	(54,20)

Figure 16: Simple example of the coordinates along a path being stored in groups depending on which section of the lcd screen they are located. *(The smallest number of sections possible is actually 9, but 6 are used here for simplicity)*

1.1.5) Head-Tracking

The original VSS patch used an Oculus Rift as a head-tracking device which was also used to provide visuals of the VAE. As this project does not provide visuals, a smaller, less obtrusive device was sought. Initially the YEI 3-Space Sensor [3] was investigated, a small device that can track angular rotation and output results into Max. It was found however that the software used to interface between the device and the computer (a Mac) was only in a Beta stage. As a consequence, the output from the sensor did not function correctly or provide useful information.

Instead, the c74 application already being used for the iPad user interface (see section 1.1.2 Location Selection) was used to extract rotational data from an iPhone. The iPhone data retrieved was accurate and provided a steady wireless connection, as opposed to the USB connection required for the YEI sensor. For user testing, a low-tech head-mounting solution was found by wrapping the iPhone in the front of a cotton hat. This was integrated with the existing head-tracking by creating a new patch that reads data from the iPhone (much like the patch used for the iPad) and maps the data to an angle from 0° to 360°, including a calibration system ensuring that each user is facing the same way in the VAE. The output of this was sent to the input of the existing head-tracking system where the Oculus rift data would have been sent. This data is used to pan

between the four direction RIR files simulating the effect of turning the the VAE.

1.1.6) Settings Menu

Figure 17 shows the settings patch 'settings'. This can be used to determine which RIR grid to use by selecting the number 1-5 on the top left of the patch (green buttons) as well as the sample rate of the RIR files. The audio input settings can be used to change which audio input is used. The panning settings determine the speed at which the user is moved between RIR locations.

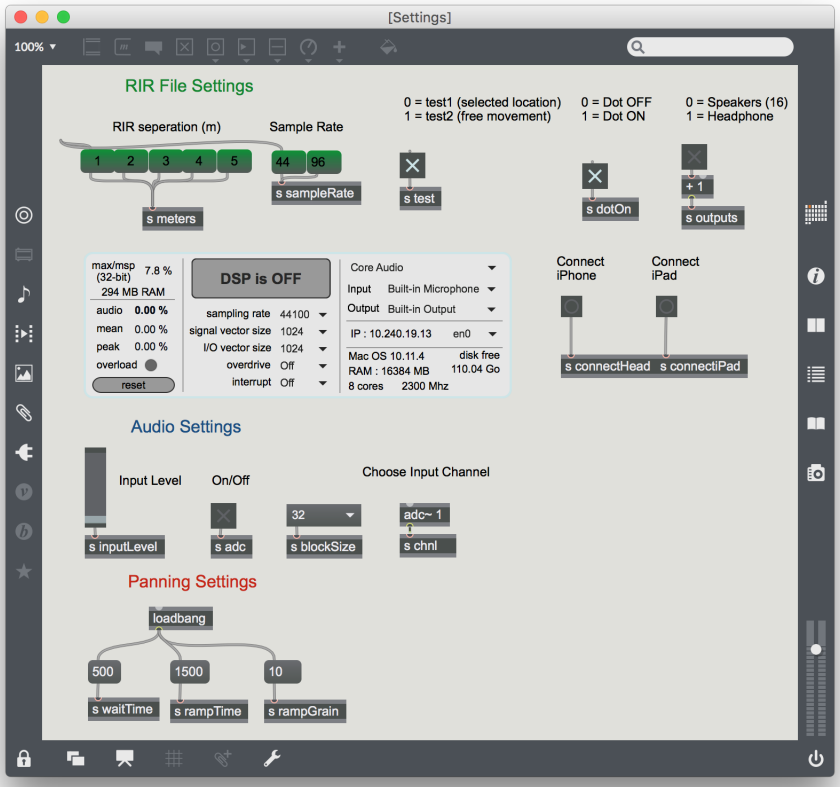


Figure 17: Settings patch used to control parts of the overall patch from one location

1.1.7) Software Issues

One issue raised when implementing iteration 3 was the way in which the user is moved through the virtual space. As only two RIR locations are used at a time as opposed to using 4 RIR locations for interpolation, the user is moved in a 'zigzag' pattern. Figure 18 shows two images illustrating how the user is moved through the virtual space using the final iteration of the software against the initial two iterations, where the curved blue line represents the path drawn by the user. The image on the right shows how the 4 RIR locations are used together to make it sound like the user

is placed between them. The image on the right shows how the user is passed between locations one at a time, causing the 'zigzag' pattern.

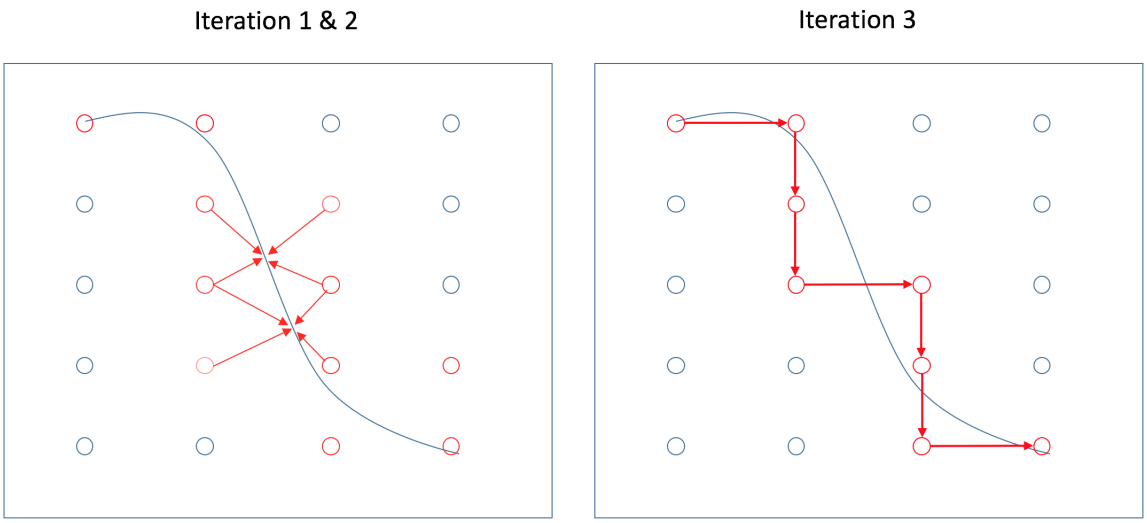


Figure 18: Illustration of how the user is moved through the virtual space using different numbers of RIR locations where the image on the **left** shows how 4 RIR's are used at a time and the image on the **right** shows the user being passed between two RIR locations at a time. The blue line represents the paths drawn by the user and the circles represent RIR locations.

As previously explained in section 1.1.3.3 Iteration 3 (Final), three convolution patches were used to move the user between two RIR locations while the third loads the next location. In an attempt to make the system less computationally expensive, if any of the convolution patched were loading a file as opposed to convolving an audio signal with one, the convolution algorithm was bypassed. However it was found that any delay in turning the bypass on or off produced a 'popping' noise. Small delays in turning the bypass on and off often occurred, therefore this bypass functionality was removed.

1.2 - Latency

The time taken for the input signal to run through the system causes the room reflection to arrive at the user at the incorrect times. To determine the length of the delay, a latency test was conducted. The test procedure was similar to that in the production of the original VSS [4] with some slight differences. Two Earthwork M30 reference microphones [5] were placed in the centre of the speaker array. One was connected to the input of the max patch and the other was connected to a Mac running reaper through a MOTU audio interface [6] to record a reference input. An output channel running to one of the Genelec speakers was routed to the second input of the MOTU interface. This was used to record the signal after it had run through the Max patch, thus allowing the two signals to be compared to find the latency time. Both systems were running at a sample

rate of 44.1kHz. To identify the impulse clearly, a pair of drumsticks were used to produce an impulse. This was then convolved with a 4-channel Dirac impulse (1 sample of amplitude 1) of approximately 3 seconds long. Figure 19 shows a diagram of the latency measurement process.

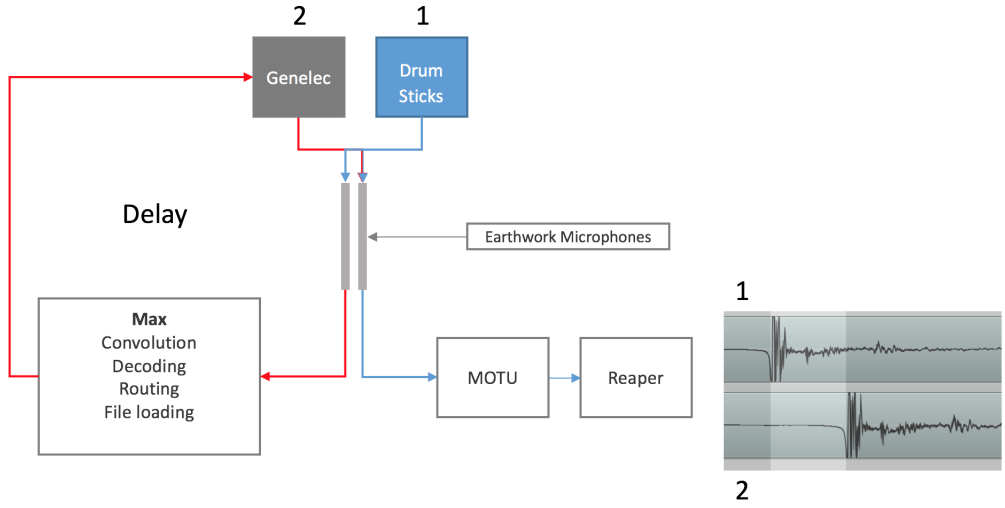


Figure 19: Diagram showing the latency measurement process. Red arrow shows the signal being delayed due to running through the max patch, eventually being output from the Genelec and running into reaper. Blue arrow indicates the direct signal to Reaper. The numbers indicate which of the audio waves comes from which source, 1 being the direct sound and 2 being the delayed sound.

Previous testing of the system had shown that the system ran smoothly when using an input block size of 32 in Max, however for future records (and in case the system speed were to slow down due to potential future changes) the latency was measured for varying block sizes. The following table shows the resulting latency time due to the different block sizes used in Max. The total latency is calculated by adding 5ms to the system latency. This accounts for the time taken to travel from the loudspeaker to the centre of the speaker array where the user will be present.

Sample Rate (kHz)	Block Size	System Latency (ms)	Total Latency (ms)
44.1	512	27	32
44.1	256	20	25
44.1	128	18	23
44.1	64	17	22
44.1	32	17	22

1.3 - RIR Trimming

Due to the latency present in the system, the first 22ms of the RIR's had to be trimmed. Figure 20 shows two of the synthetic RIR's, the top shows one from the centre of the room (8.85m from the left wall) and the bottom shows an RIR taken 1.85m away from the left wall which are two of the

locations that were also taken in Hendrix Hall for user test #1. Both of the RIR's being analysed were produced with the sound source facing the left wall to emphasis the early wall reflection making the following point more obvious.

Points (1), (2) and (3) are identical in both RIR's showing the start of the impulse, the direct sound and the floor reflection respectfully. The main difference in the RIR's are the direct reflection times from the left wall. (4) shows the wall reflection occurring at 0.04084s, indicating a travel distance of 3.75m (following the same patch described in figure **??** in section **?? ??**). (5) shows the reflection from the same wall, however occurring much later due to the greater distance from the wall. The point at which the RIR's will be trimmed is indicated by the red intersecting line, occurring at approximately 0.052s. It can be seen that the direct reflection from the left wall (4) is removed from the impulse, whereas in the RIR in the centre of the room, the direct wall reflection (5) remains in the trimmed version of the RIR. Due to the 22ms delay, any sound that travels less than 7.568m (if the source and receiver were placed 3.784 away from a surface) before reaching the receiver will be removed from the RIR.

The necessity for direct sound has been previously discussed in [7], stating that as reverb builds up over time eventually masking reflections, the direct sounds are used for sound source localisation. As the user in the VAE is both the sound source and the receiver, they will rely on direct reflections from the walls to determine their location within the space, however, the direct reflections from walls have had to be removed in some cases. Therefore, when moving closer to a wall, it may become difficult for the user to determine their locations.

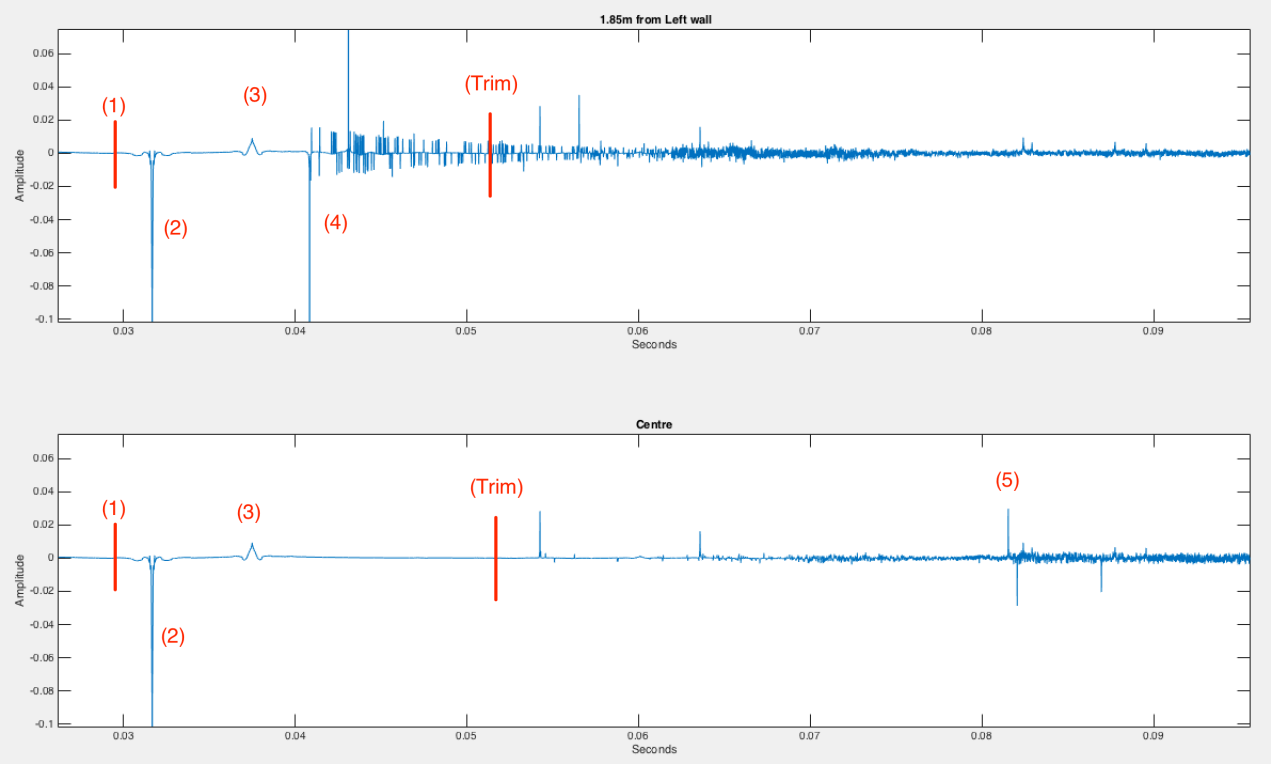


Figure 20

The Matlab script used from audio file trimming can be found: RIRtrim.m

This was done for the real RIR measurements as well. Figure **??** shows two RIR's in the same positions as mentioned above, where the early wall reflection in the bottom RIR (3) is trimmed.

Appendices

APPENDIX A

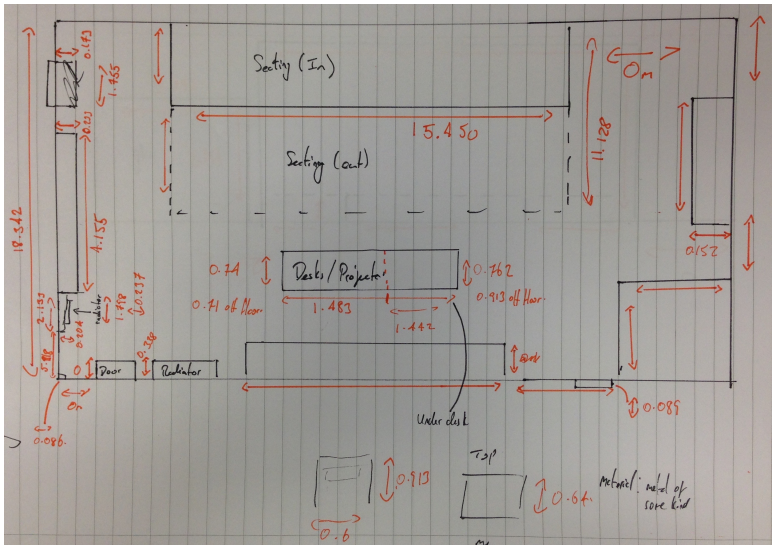


Figure 21: Annotated blueprint of Hendrix Hall from a birds-eye view

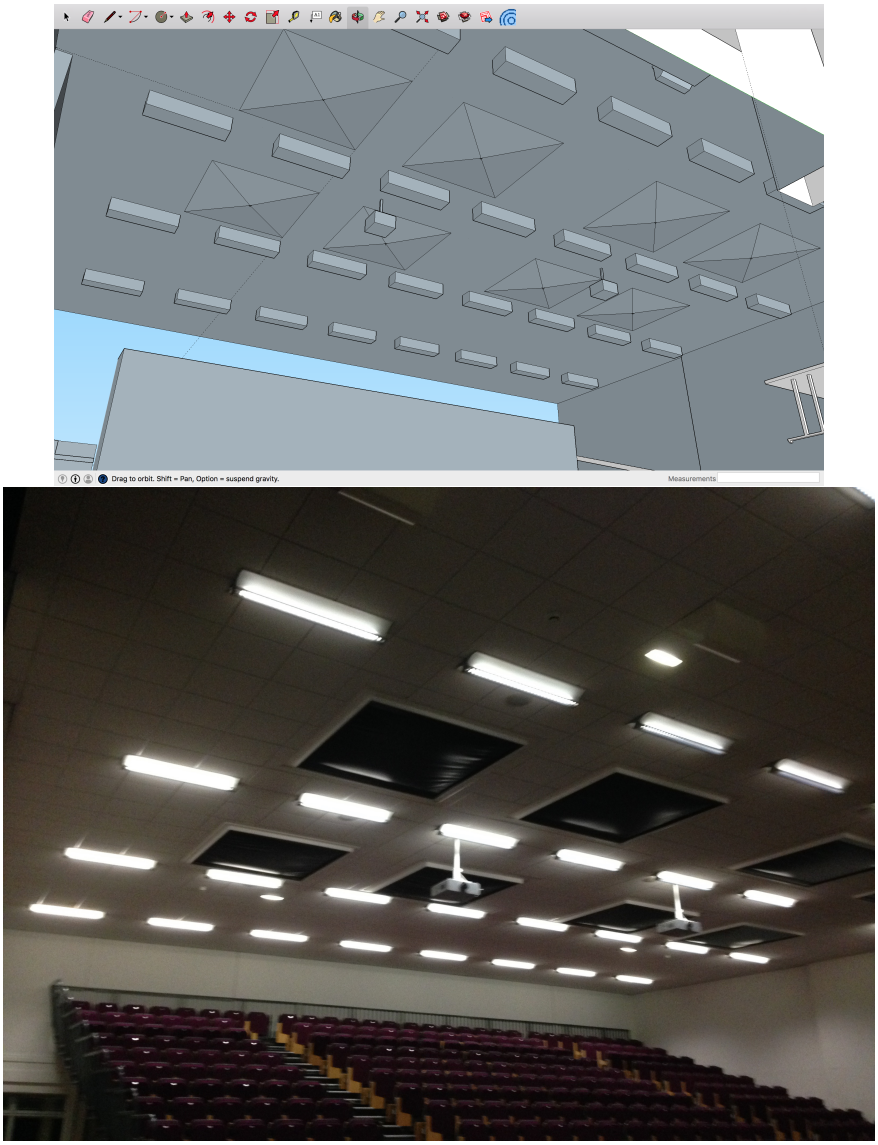


Figure 22: Real Vs SKU Roof

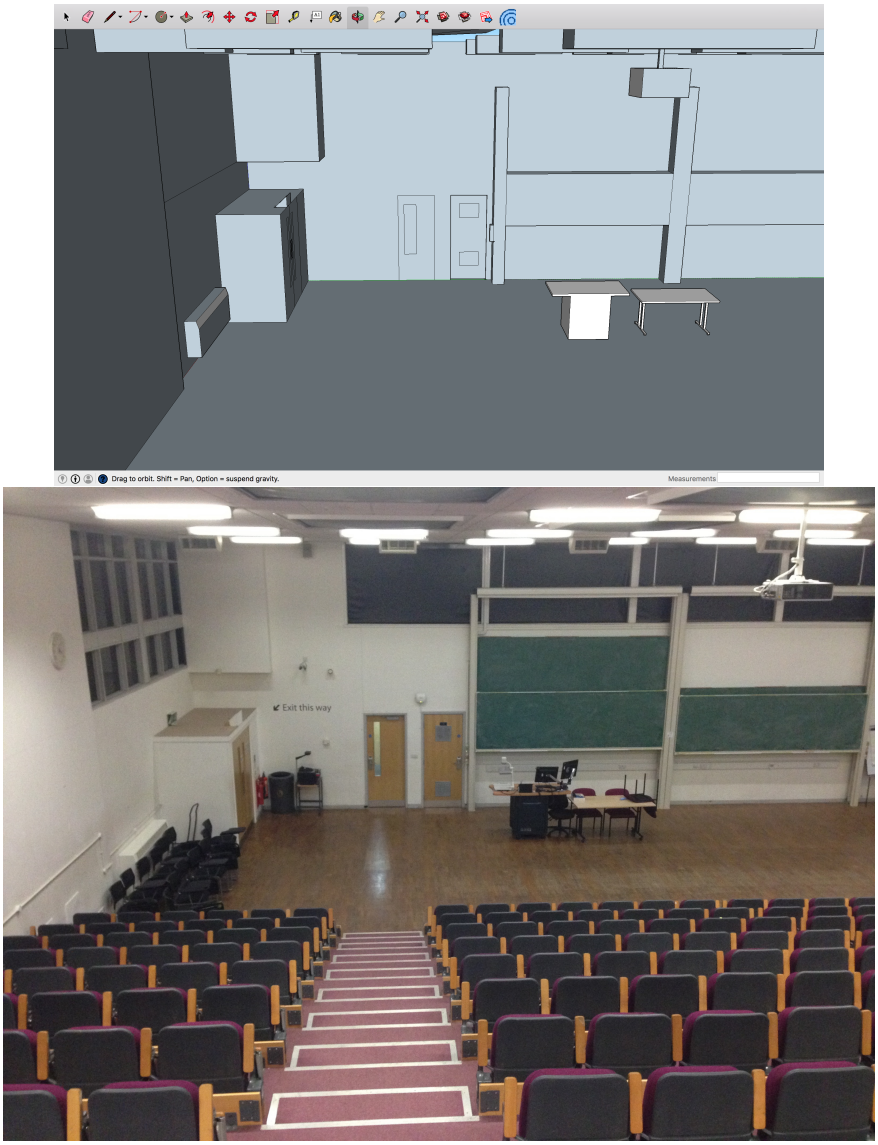


Figure 23: Real Vs SKU Seating Area

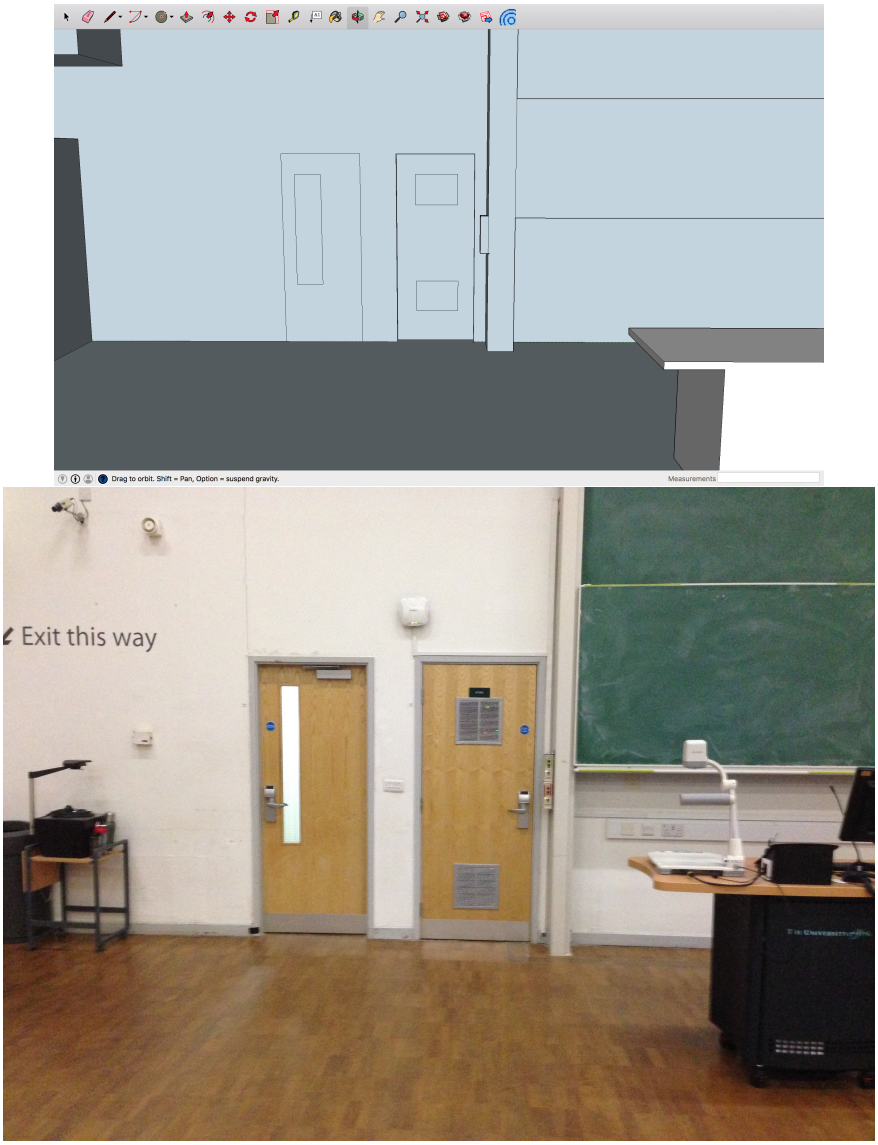


Figure 24: Real Vs SKU Door

APPENDIX B

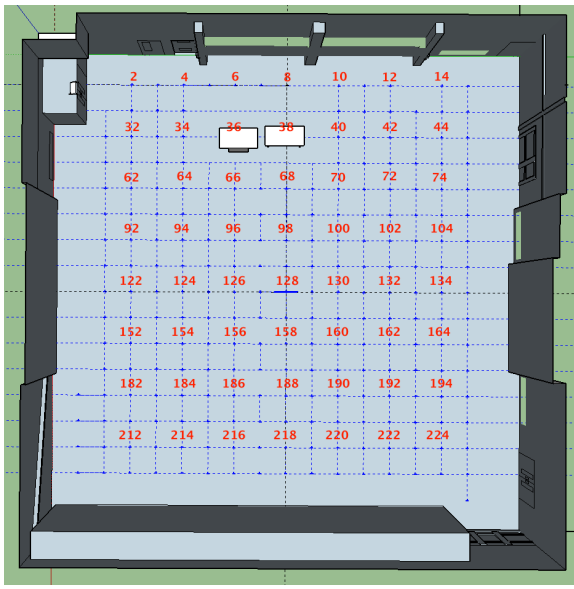


Figure 25: RIR grid with 2m separation

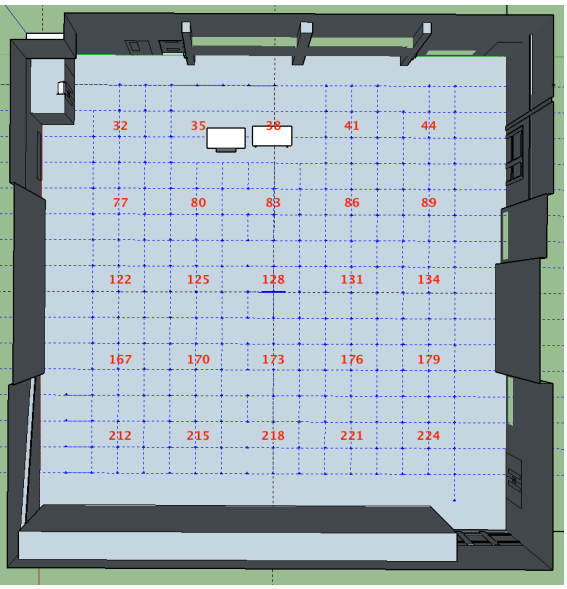


Figure 26: RIR grid with 3m separation

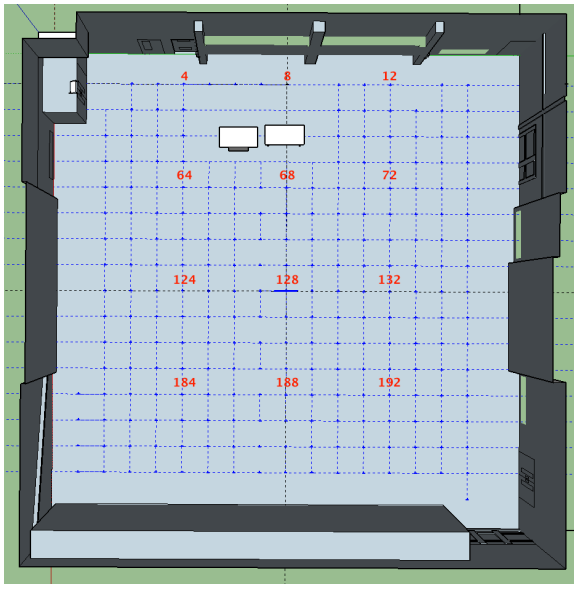


Figure 27: RIR grid with 4m separation

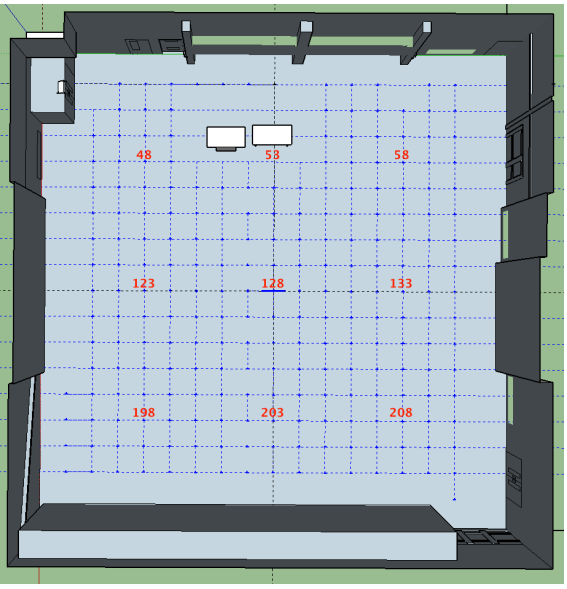


Figure 28: RIR grid with 5m separation

APPENDIX C

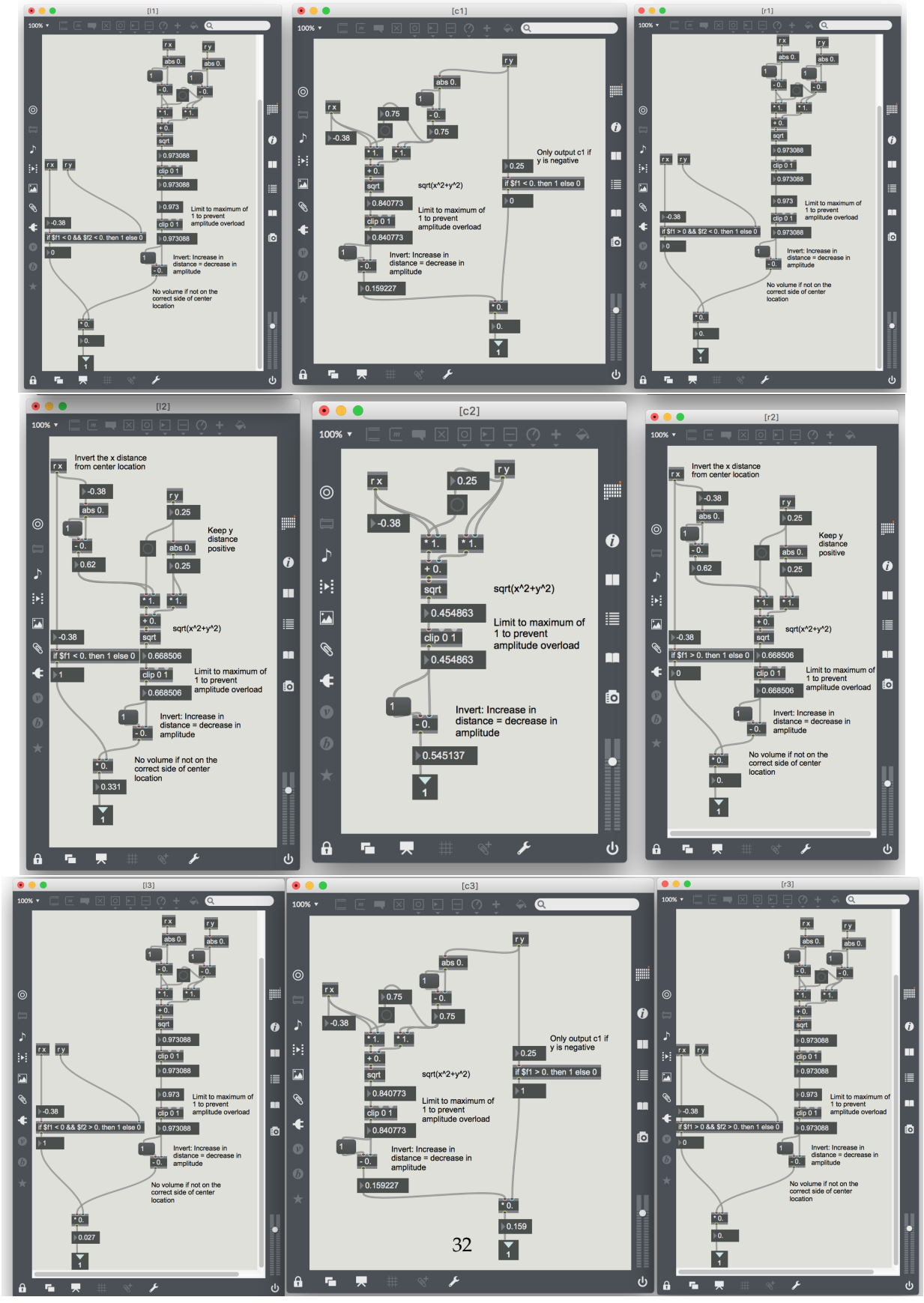


Figure 29: Overview of the individual panning algorithms used in iteration 1.

```
inlets = 5;
outlets = 5;

//Create arrays to store previous positions
var xArray = new Array(2);
var yArray = new Array(2);

var windowSize = new Array(2);

//Variables to use for file searching
var fileX, fileY, search;

//Defines how to split up the grid
var numberOfMeters;

//Loads appropriate files given users finger coordinates
function msg_int(input){
 if(inlet == 0){
  xPos = input;
 } else if (inlet == 1){
  yPos = input;//Add off set to start at (0,1)
 } else if (inlet == 2){
  windowSize [0] = input;
 } else if (inlet == 3){
  windowSize [1] = input;
 } else if(inlet==4){
  numberOfMeters = input;
 }

 //Split into sections
 if(numberOfMeters == 3 || numberOfMeters == 5){
  //Even grid for 3m and 5m
  xPosition = (xPos/windowSize[0])*(numberOfMeters);
  yPosition = (yPos/windowSize[1])*(numberOfMeters);
 } else if (numberOfMeters == 4 || numberOfMeters == 8){
  //4m separation requires different x,y coordinate scaling
  xPosition = (xPos/windowSize[0])*(numberOfMeters-1);
  yPosition = (yPos/windowSize[1])*(numberOfMeters);
 } else{
  //Extra row for others
  xPosition = (xPos/windowSize[0])*(numberOfMeters);
  yPosition = (yPos/windowSize[1])*(numberOfMeters+1);
 }

 //Round to nearest value
 xSection = Math.round(xPosition);
 ySection = Math.round(yPosition);

 //Start the lcd grid sections from column 1 row 1 instead of column 0 row 0
 if(xSection == 0){
  xSection = 1;
 }
```

```
53 if(ySection == 0){
54  ySection = 1;
55 }
56
57 //Distance in % away from center of section
58 xBetween = 2*(xPosition - xSection); //x2 to get 100%
59 yBetween = 2*(yPosition - ySection);
60
61 //Which RIR to load in centre location
62 outlet(0,xSection);
63 outlet(1,ySection);
64
65 //Output panning values
66 outlet(2,xBetween);
67 outlet(3,yBetween);
68
69 //Store current location
70 xArray[0] = xSection;
71 yArray[0] = ySection;
72
73 //If either coordinate is changed search for new files
74 if(xArray[0] != xArray[1] || yArray[0] != yArray[1]){
75
76  if(xArray[0] != xArray[1]){
77   //Store previous value
78   xArray[1] = xArray[0];
79   X = xArray[0];
80  }
81
82  if(yArray[0] != yArray[1]){
83   yArray[1] = yArray[0];
84   Y = yArray[0]
85  }
86
87  //Output user location within grid
88  if(numberOfMeters == 4 || numberOfMeters == 8){
89   fileNumber = X + ((numberOfMeters-1)*(Y-1)); //Requires different algorithm for 4m
      due to different grid shape
90  } else {
91   fileNumber = X + ((numberOfMeters)*(Y-1));
92  }
93  outlet(4,fileNumber);
94 }
95}
```

Sections/Appendix/AppendixA/Code/loadFilesLogic.js

APPENDIX D

Test Participant Form

You have volunteered to partake in two user tests that should take no longer than 30 minutes to complete.

Test Descriptions

The VSS (virtual singing studio) is a system that is used to simulate the acoustics of another room. The system can be used by standing in the centre of the speaker array and singing into a head mounted microphone. By wearing the provided head-tracking device, you can turn in the virtual space by turning your head/body.

Test #1

This test aims to investigate the perception of movement within the virtual acoustic environment when using two different methods: Method **A** and Method **B**. You will be asked to step inside the VSS and say the word "Bob". Your location within the virtual space will then be changed and you will be asked to produce another sound. This process will then be repeated a second time but this time using method **B**. You will then be asked to state whether method **B** felt like you had:

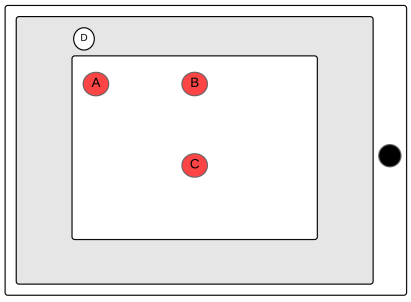
Moved a **shorter** distance than I had in **A**
Moved the **same** distance as I had in **A**
Moved a **further** than I had in **A**
I don't know

This process will be repeated 5 times in total.

Test #2

Part 1: You will be asked to step inside the VSS and to sing or produce a noise. After a short amount of time you will be asked to do the same again. You will then be asked whether you feel you have changed location or not with a simple Yes/No answer. This will be repeated 8 times.

Part 2: In this part of the test, you will be asked to change your location within the virtual space yourself by tapping on a location or dragging your finger around the iPad provided for you. You will be asked to rate on a scale of **1 - 10** how free you feel you can move about the room with **1** being a *jumpy movement* and **10** being *complete freedom to move without limitations*. You will also be given the opportunity to add comments to further explain you score if you wish.



To the left is a diagram of an iPad. **A**, **B**, and **C** indicate where parts of the room can be located. When situated in the VSS, you will start in the center of the room **(C)** facing towards the front of the room **(B)**.

A = Top left corner of the room
B = Front of the room
C = Centre of room
D = Button to calibrate head tracker

Answering Question

Note that when you're within the VSS it will be difficult to write down your answers to the questions asked. Therefore you will be asked to answer verbally and your answers will be taken down for you. You will be asked at the end of the test to check that your answers have been taken down truthfully.

Information and Consent

Experimenter: _____

Please read the following statements and tick the boxes on the right hand side to indicate that you understand and agree.

I understand that at any point I may choose to withdraw from the experiment	☐
I understand that I may omit answers to any questions	☐
I agree that I am here voluntarily	☐
I understand and agree that the experimenter conductor will be observing the experiment	☐
I agree that the system being used has been explained to me	☐
I agree that the point of this experiment has been explained to me	☐

Participant Signature: _____

Answer Sheet

Participant Number: _____ Date: _____

Test #1

Question 1: Please state whether you feel you have:

Moved a **shorter** distance than I had in **A**
Moved the **same** distance I had in **A**
Moved a **further** distance than I had in **A**
I don't know

Trial	**Score**			
	Shorter	Same	Further	Don't Know
1	○	○	○	○
2	○	○	○	○
3	○	○	○	○
4	○	○	○	○
5	○	○	○	○

I agree that the answers that have been taken down on my behalf are correct □

Participant Signature: _____

Participant Number: _____ Date: _____

Test #2 - Part 1

Question 2: Do you feel you have changed location within the room?

Trial	**Answer**
1	YES/NO
2	YES/NO
3	YES/NO
4	YES/NO
5	YES/NO
6	YES/NO
7	YES/NO

Test #2 - Part 2

Question 3: Please rate on a scale of **1 - 10** the mobility within the virtual space where **1** = Extremely "jumpy" movement and **10** = Completely smooth movement or please select "N/A" if you can not tell you are moving.

Trial	**Score**										**N/A**
1	1	2	3	4	5	6	7	8	9	10	
	○	○	○	○	○	○	○	○	○	○	○
2	1	2	3	4	5	6	7	8	9	10	
	○	○	○	○	○	○	○	○	○	○	○
3	1	2	3	4	5	6	7	8	9	10	
	○	○	○	○	○	○	○	○	○	○	○
4	1	2	3	4	5	6	7	8	9	10	
	○	○	○	○	○	○	○	○	○	○	○
5	1	2	3	4	5	6	7	8	9	10	
	○	○	○	○	○	○	○	○	○	○	○

Comments:

I agree that the answers that have been taken down on my behalf are correct □

Participant Signature: _____

Question 4: Please rate on a scale of **1 - 10** the mobility within the virtual space where **1** = Extremely staggered movement and **10** = Completely smooth movement or please select "N/A" if you can not tell you are moving.

Trial	**Score**										**N/A**
1	1	2	3	4	5	6	7	8	9	10	
	○	○	○	○	○	○	○	○	○	○	○
2	1	2	3	4	5	6	7	8	9	10	
	○	○	○	○	○	○	○	○	○	○	○
3	1	2	3	4	5	6	7	8	9	10	
	○	○	○	○	○	○	○	○	○	○	○
4	1	2	3	4	5	6	7	8	9	10	
	○	○	○	○	○	○	○	○	○	○	○
5	1	2	3	4	5	6	7	8	9	10	
	○	○	○	○	○	○	○	○	○	○	○

Comments:

I agree that the answers that have been taken down on my behalf are correct □

Participant Signature: _____

pagebreak

APPENDIX E

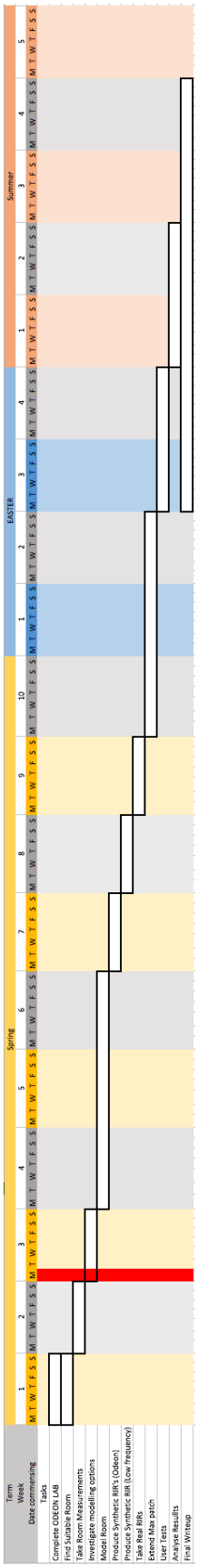


Figure 30: Initial Gantt chart showing the approximate time allocated to each of the tasks required to complete the project. The red line shows the time at which the Gantt chart was abandoned for a more appropriate planning method

REFERENCES

[1] (2016). Mira, Cycling74, [Online]. Available: https://cycling74.com/products/mira/ (visited on Apr. 25, 2016).

[2] (2016). C74, Cycling74, [Online]. Available: https://cycling74.com/project/project43-c74-iphone-app/#.Vv1FhRIrJE4 (visited on Apr. 25, 2016).

[3] Y. Technology, *3-space sensor embedded*, 630 Second Street, Portsmouth, Ohio 45662. [Online]. Available: http://www.solarsystemexpress.com/uploads/5/0/6/0/5060129/3-space_sensor_users_manual_embedded_1.1_r13.pdf (visited on May 2, 2016).

[4] J. S. Brereton and B. A. Hons, "Singing in space (s): singing performance in real and virtual acoustic environments — singers ' evaluation , performance analysis and listeners ' perception .," no. August, 2014.

[5] *M30 high definition measurement microphone*, Data Sheet, Earthworks, 2016. [Online]. Available: http://www.earthworksaudio.com/wp-content/uploads/2016/03/M30-Data-Sheet-2016.pdf (visited on May 3, 2016).

[6] (2016). Motu ultralite-mk3, MOTU, [Online]. Available: http://motu.com/products/motuaudio/ultralite-mk3 (visited on May 3, 2016).

[7] S. Devore, A. Ihlefeld, K. Hancock, B. Shinn-Cunningham, and B. Delgutte, "Accurate sound localization in reverberant environments is mediated by robust encoding of spatial cues in the auditory midbrain," *Neuron*, vol. 62, no. 1, pp. 123–134, 2009, ISSN: 08966273. DOI: 10.1016/j.neuron.2009.02.018.