# Acronyms

**VAE** Virtual Acoustic Environment . . . 3
**VSS** Virtual Singing Studio . . . 3
**RIR** Room Impulse Response . . . 3

# Contents

# 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.

## 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. Figure 1 shows this.

This was the primary aspect of the original patch that needed extending to make the process automatic.

The software was run on a 2012 Mac mini with 8GB of RAM running OSX 10 **FIND SPECS**

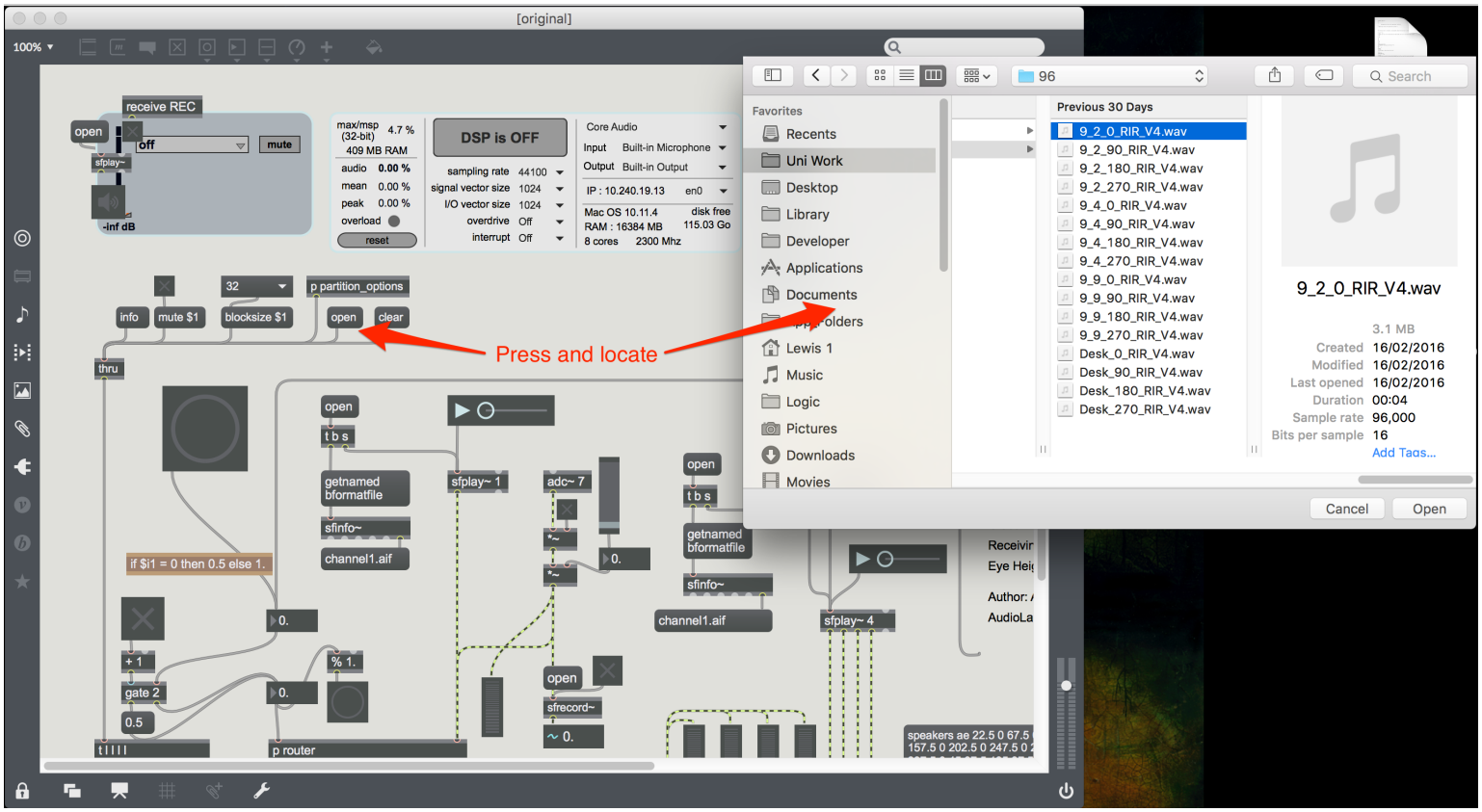


Figure 1: Original Max patch used in the VSS showing the file location window that pops up 4 times.

#### 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 used to load new RIR files when appropriate.

**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.

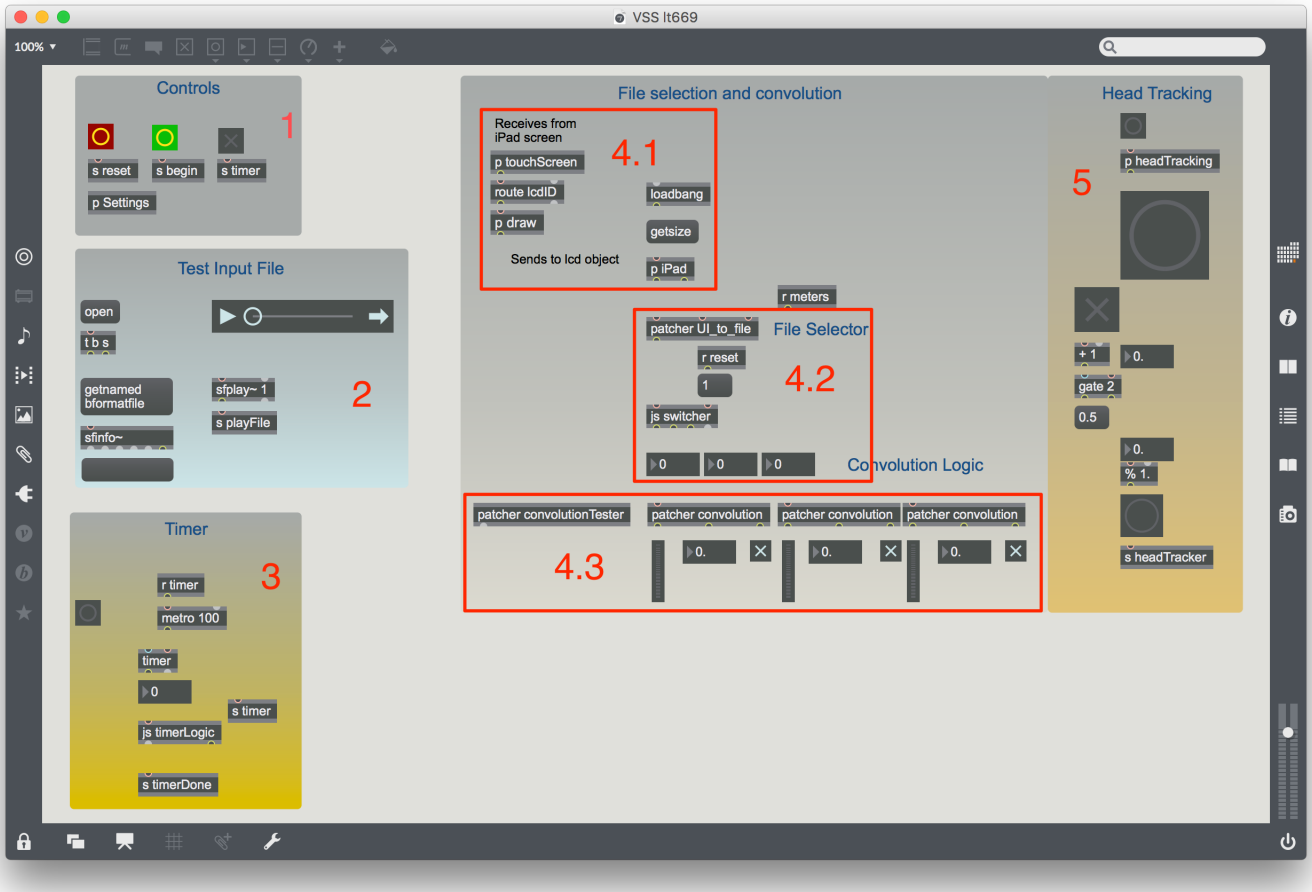


Figure 2: Top level Max patch

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.

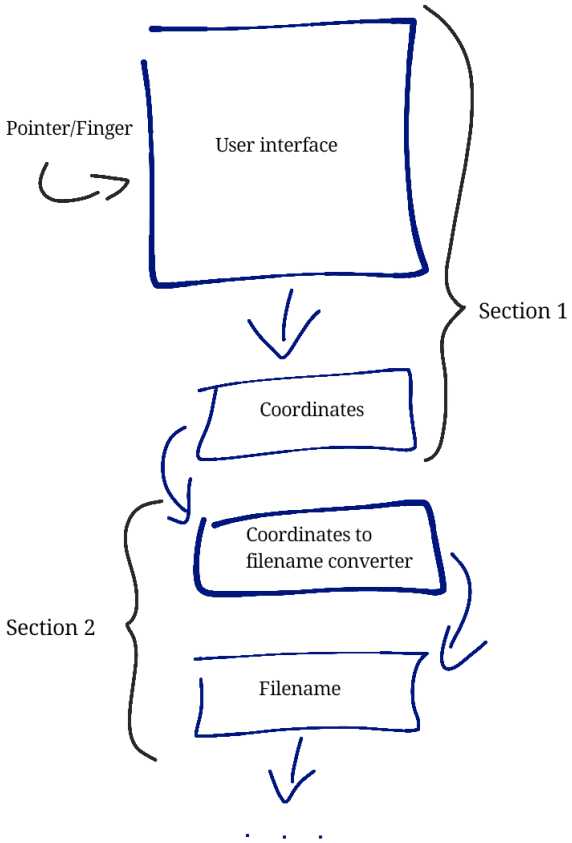


Figure 3: Flow diagram of the location selection software design. **Section 1** indicates the user interface section where coordinates are recorded and **Section 2** takes these coordinates and finds the appropriate RIR file.

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.

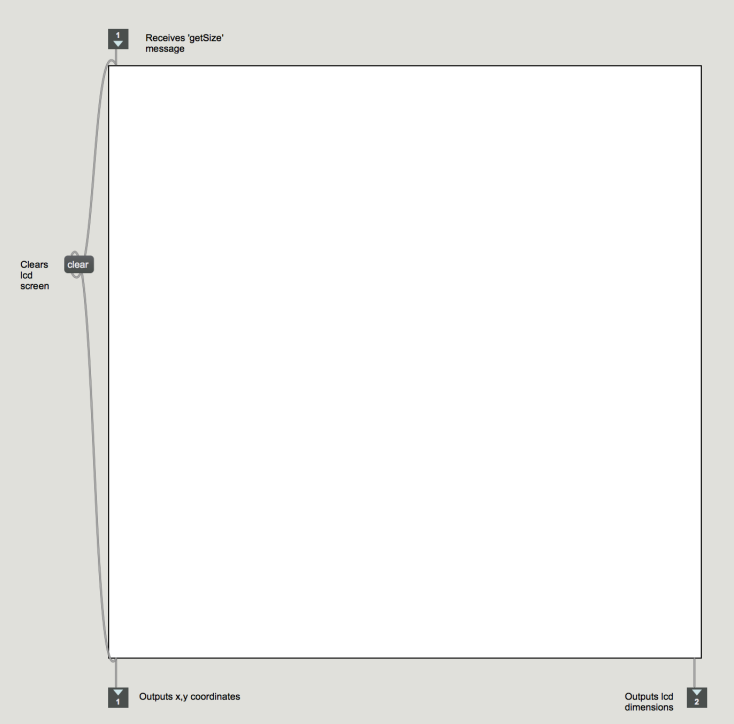


Figure 4: Flow diagram of the location selection software design

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 disapled 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.

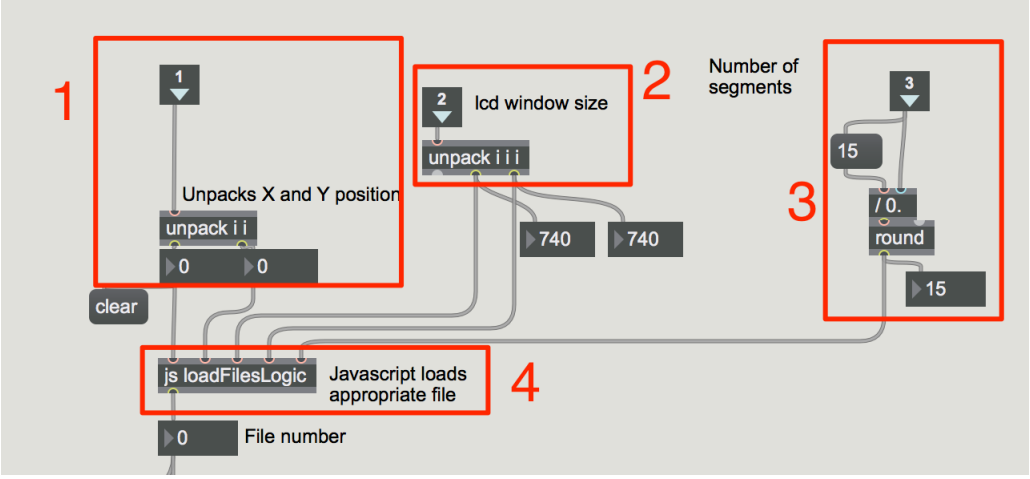


Figure 5: Screen shot from Max showing a section that converts the user interface coordinates into the appropriate file name. 1) Coordinates from lcd object 2) Dimensions of lcd object 3) Number of segments the lcd object should be split into due to the number of RIR's per meter 4) JavaScript file that 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 18,19,20 and 21 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 22 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 **HERE LINK!: Writeup/videios/iteration1Panning**.

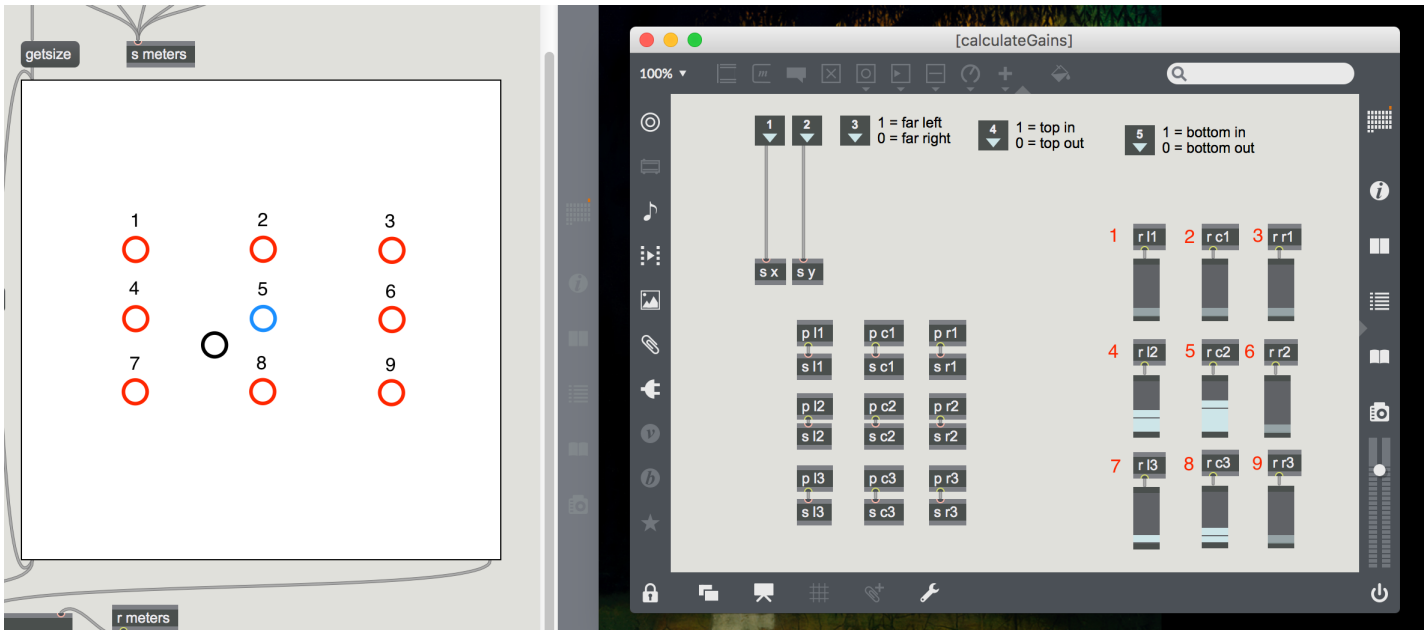


Figure 8

#### 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.

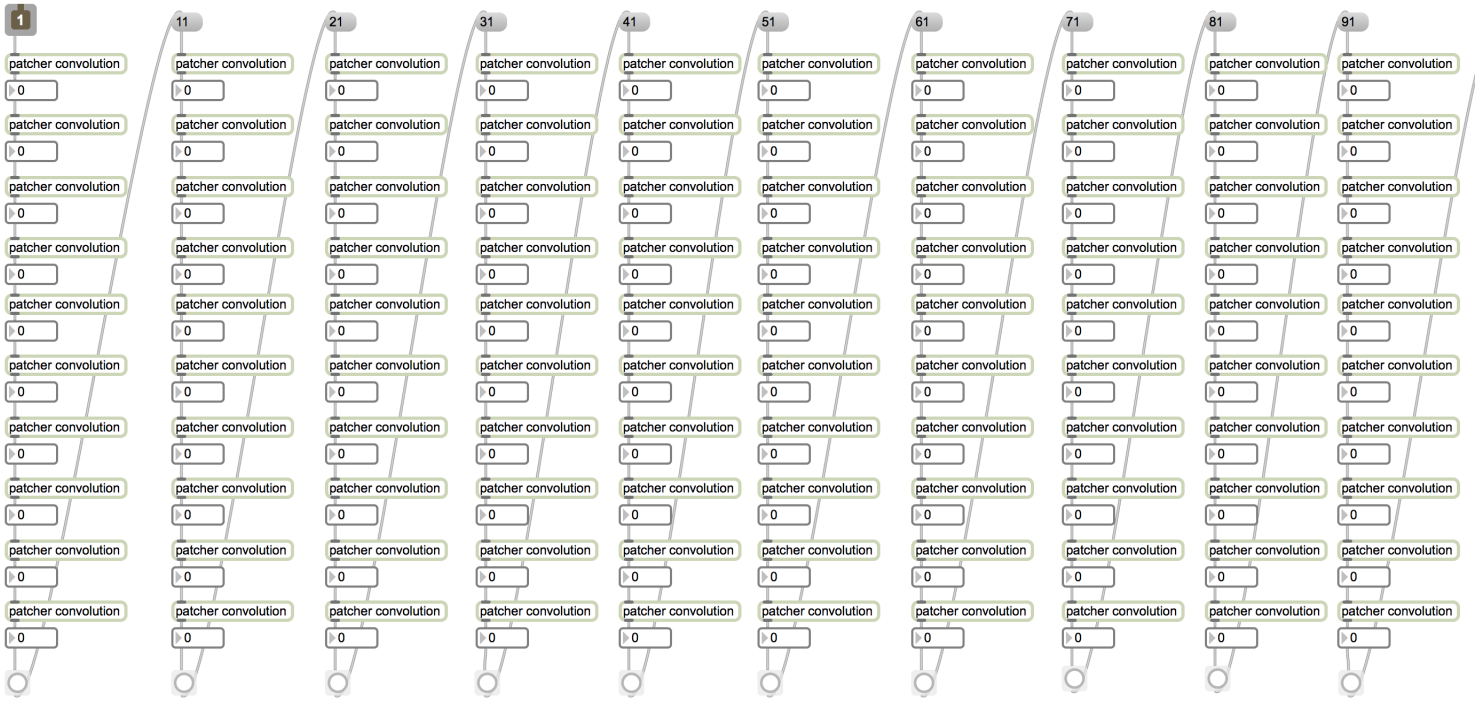


Figure 9: Chained convolution patches used to pre-load each RIR file. *Image taken from Max 6 as the version of this patch would not run in Max 7*

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 when the appropriate files had been loaded.

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 [1]. 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.

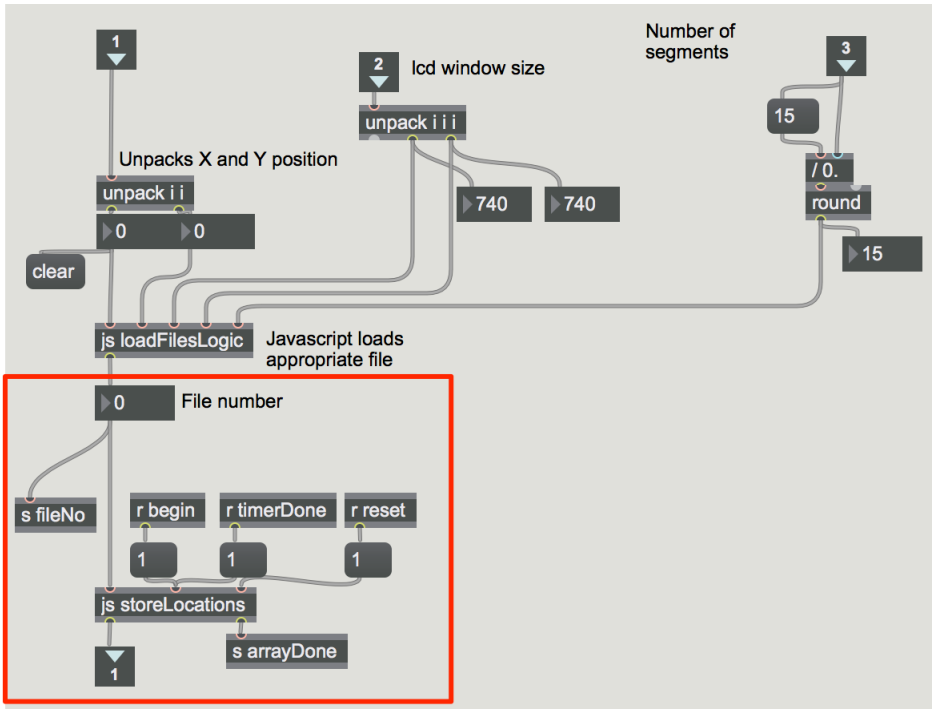


Figure 10: Highlighting the section of 'UI_to_file' that stores the file numbers and outputs them when a timer is done.

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

[1] An array that can be extended or truncated once created

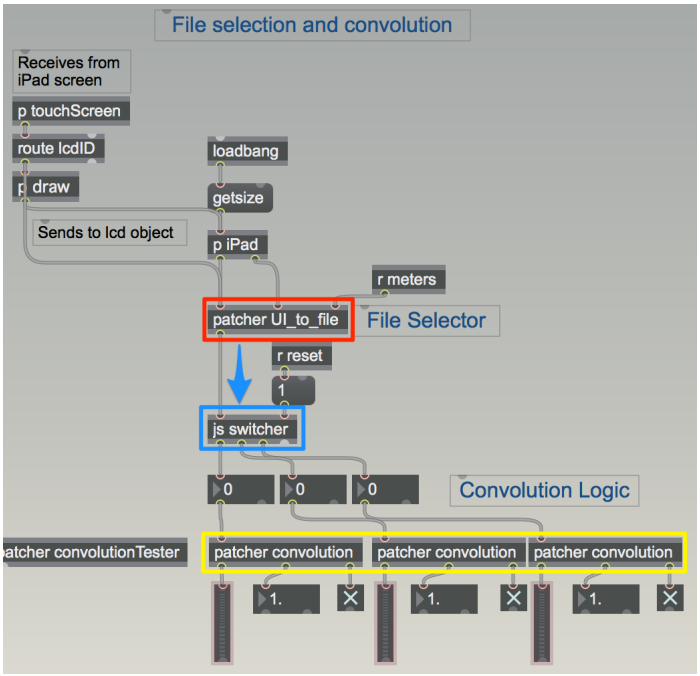


Figure 11: **Red:** Outputs file stored file number. **Blue:** Distributed file number to a different convolution patch each time. **Yellow:** Receives file number and loades appropriate file into the system.

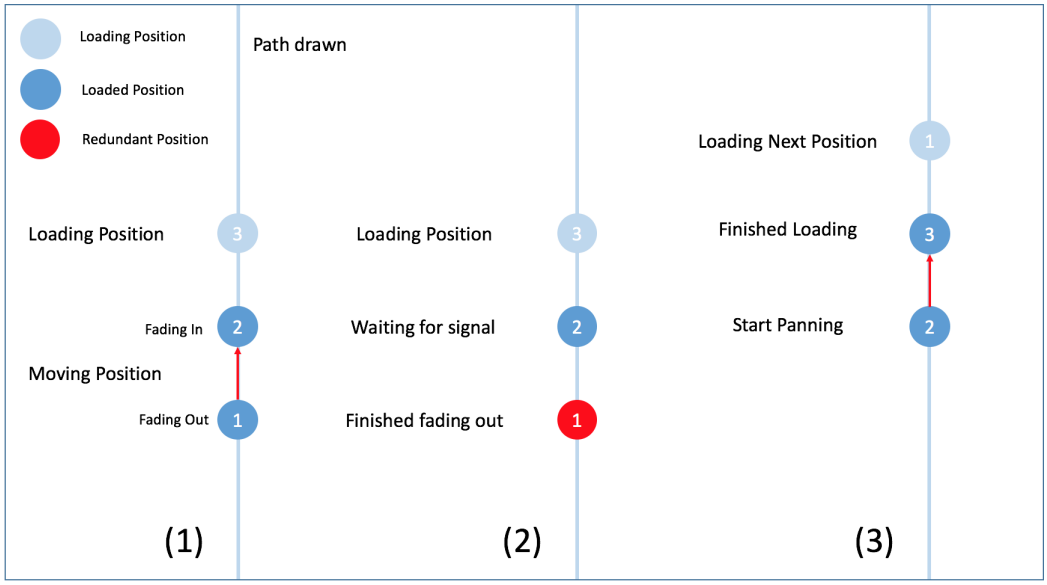


Figure 12: Illustration of how the three convolution patches work together to simulate movement, where the circles represent locations within the VAE and are numbered to indicate which convolution patch they are loaded in. **(1)** User is moved between two loaded positions while the third patch loads 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.

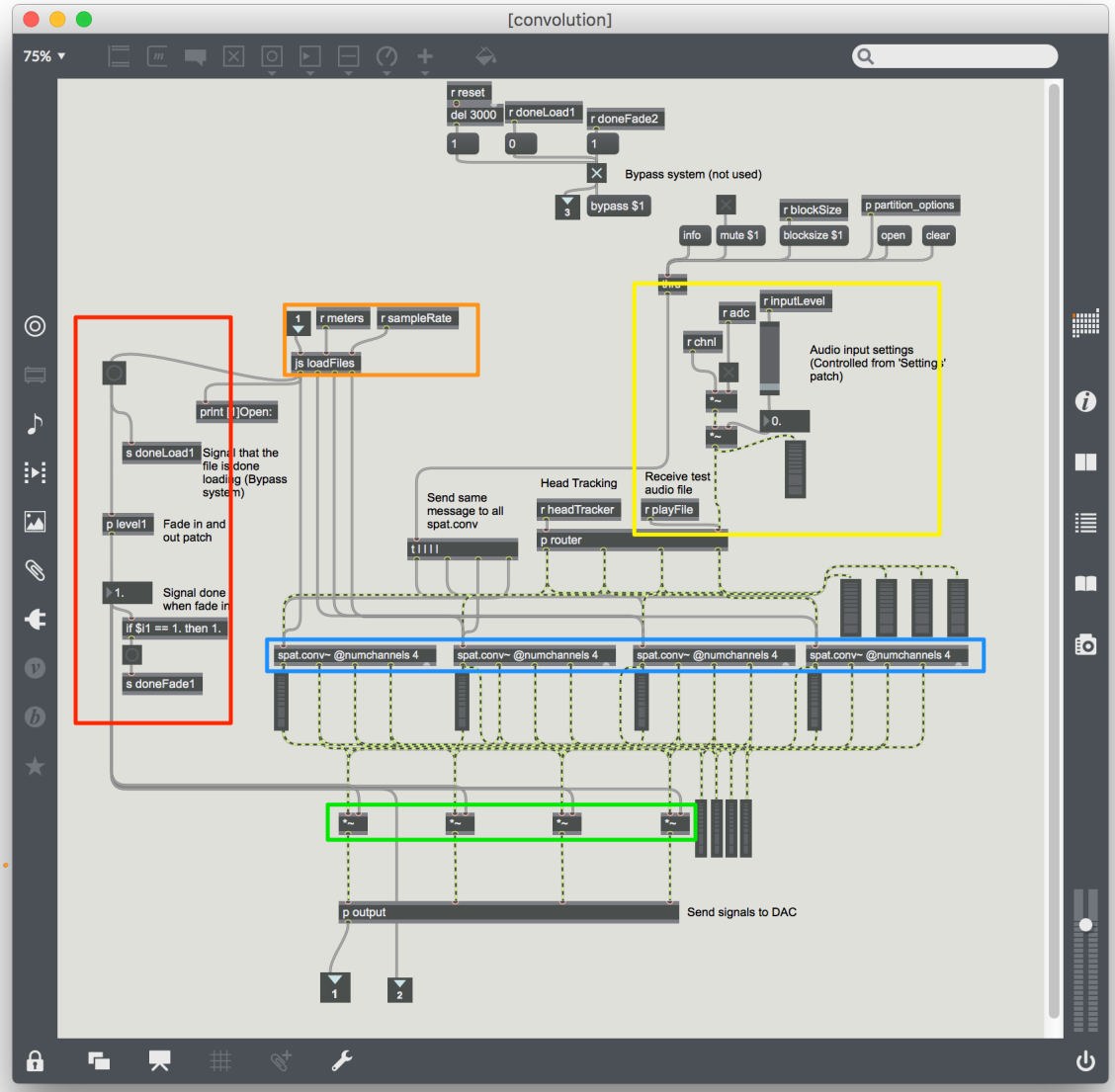


Figure 13

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 patch 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 pre-pending 'open' message, manually searching for a file

is not required, thus automating the process. These objects convolve the loaded file with what ever audio input it 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) Head-Tracking

The original patch used in the VSS 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. [3]

The YEI sensor YEI Sensor was not good.

#### 1.1.5) Location Tracking

#### 1.1.6) Software Issues

### 1.2 - Latency Test

### 1.3 - RIR Trimming

# Appendices

---

## Appendix A

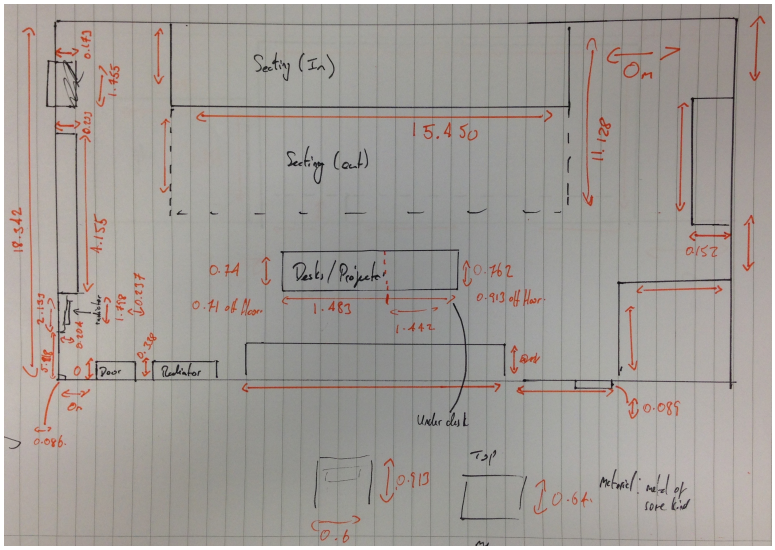


Figure 14: Annotated blueprint of Hendrix Hall from a birds-eye view

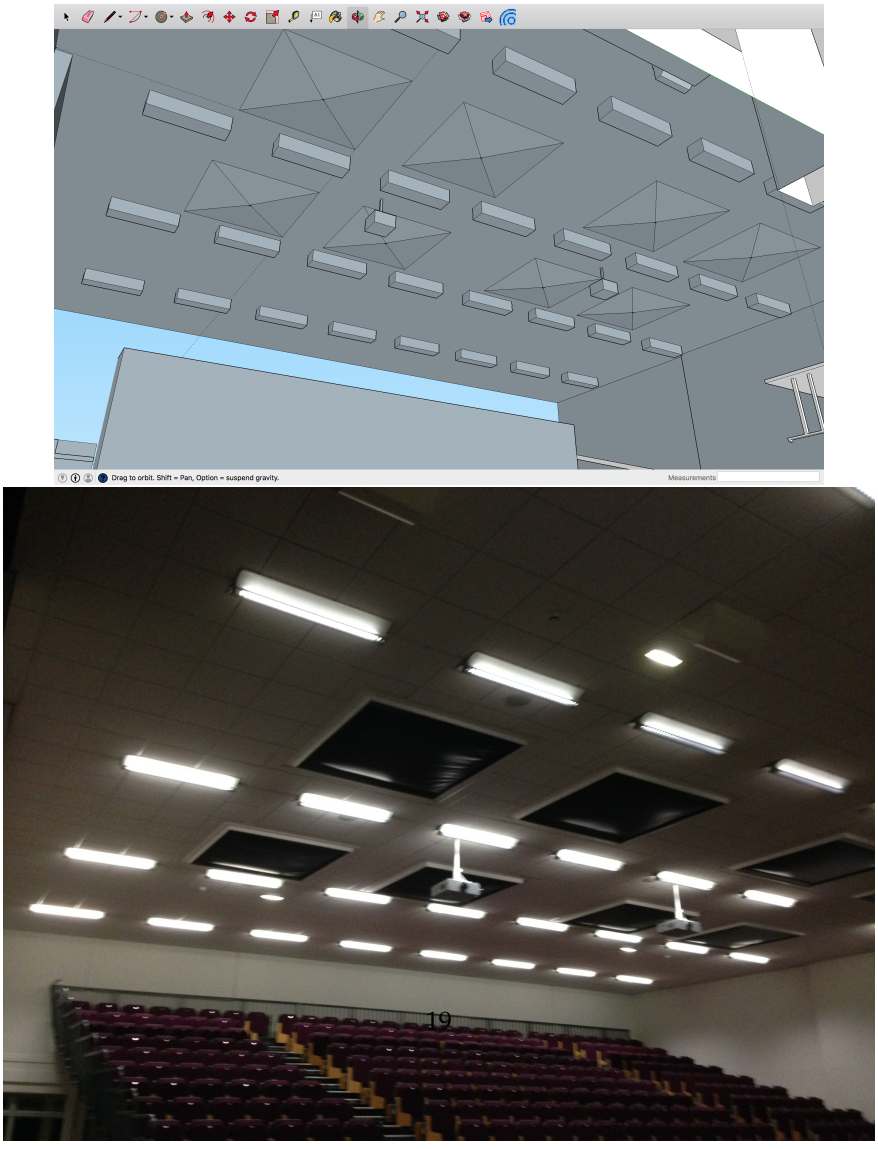


Figure 15: Real Vs SKU Roof

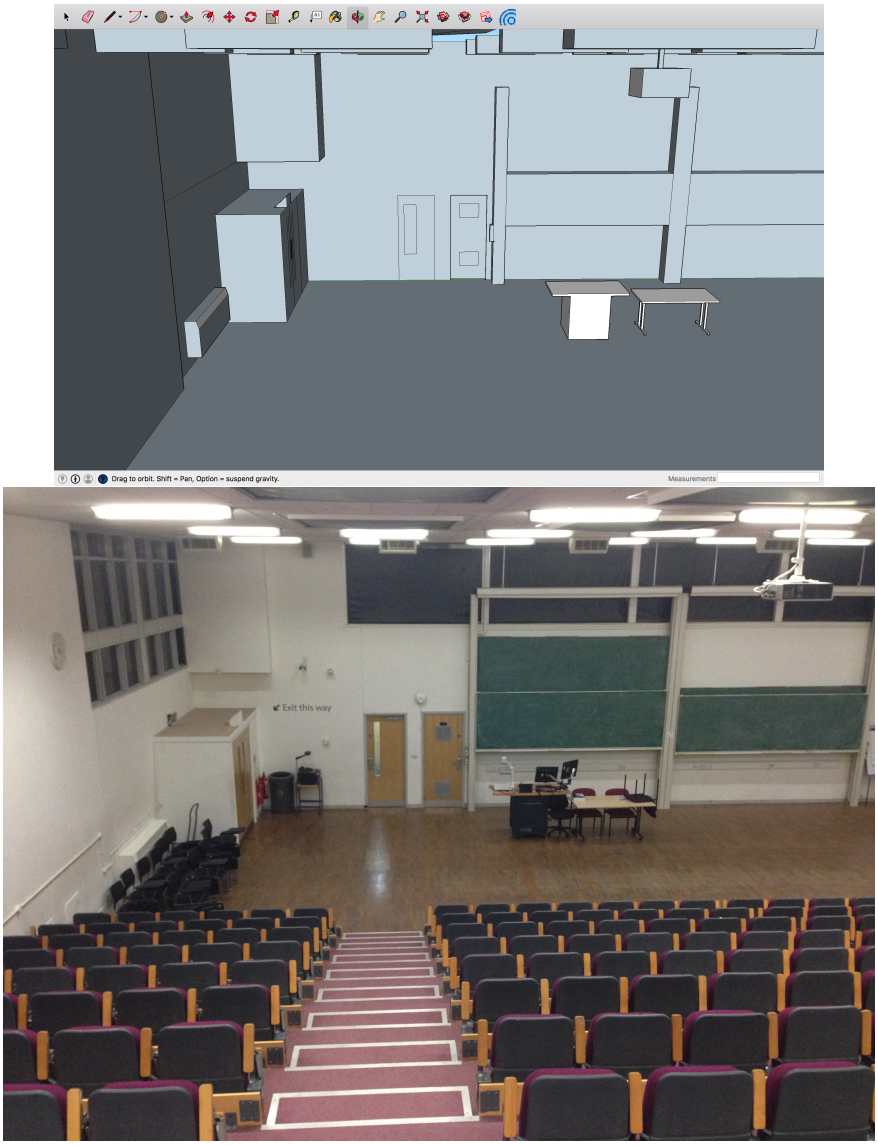


Figure 16: Real Vs SKU Roof

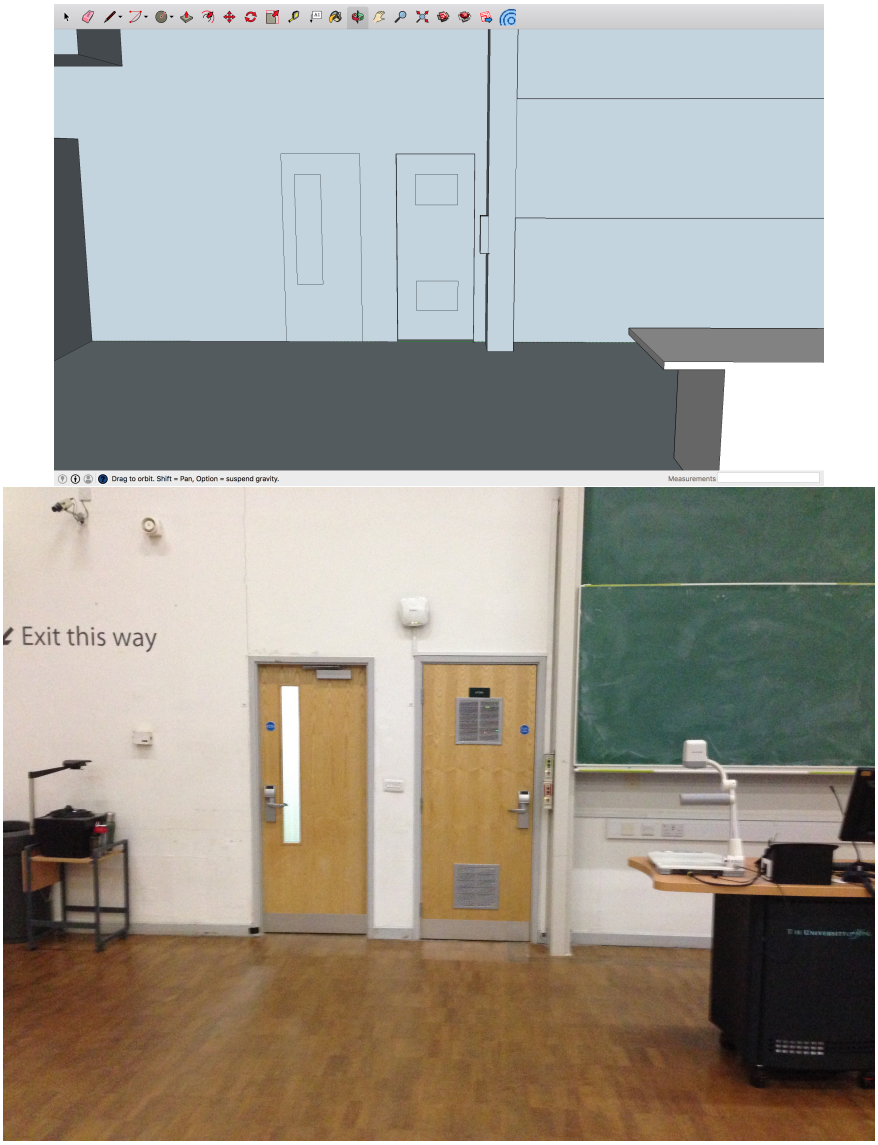


Figure 17: Real Vs SKU Roof

# APPENDIX B

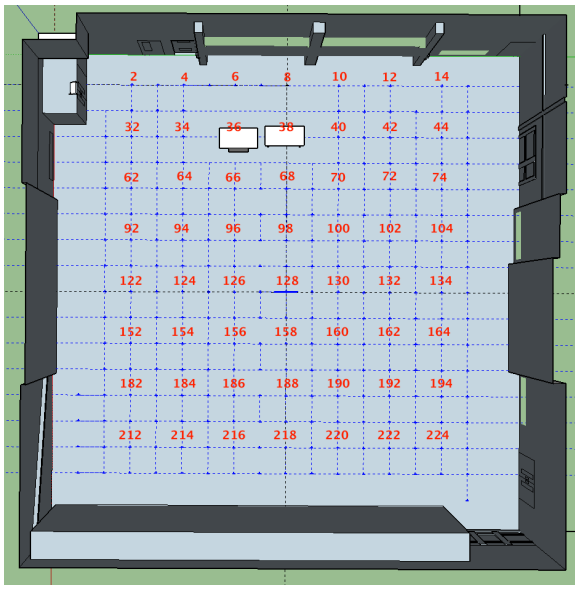


Figure 18: RIR grid with 2m separation

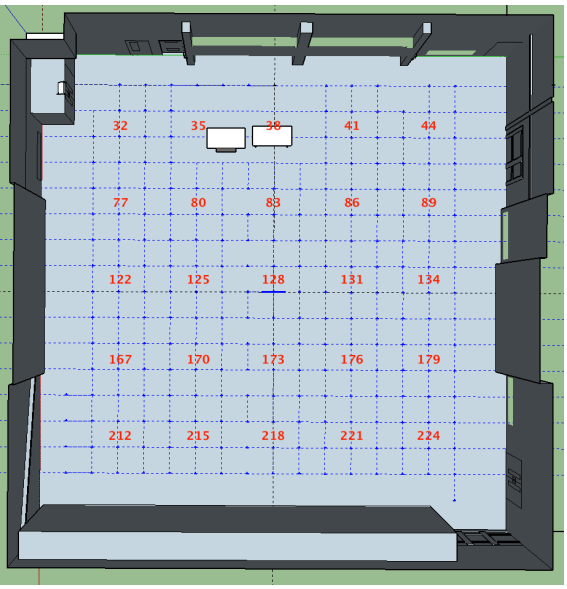


Figure 19: RIR grid with 3m separation

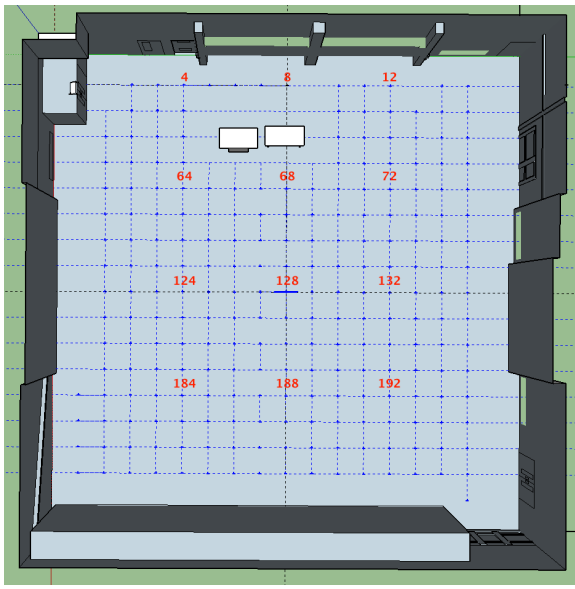


Figure 20: RIR grid with 4m separation

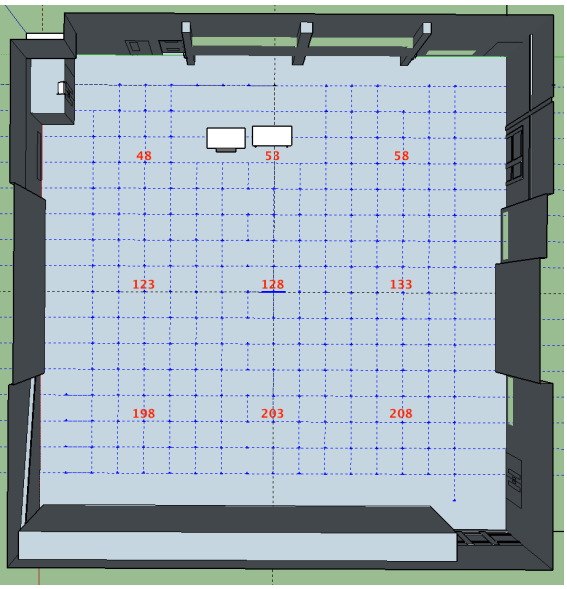


Figure 21: RIR grid with 5m separation

# APPENDIX C

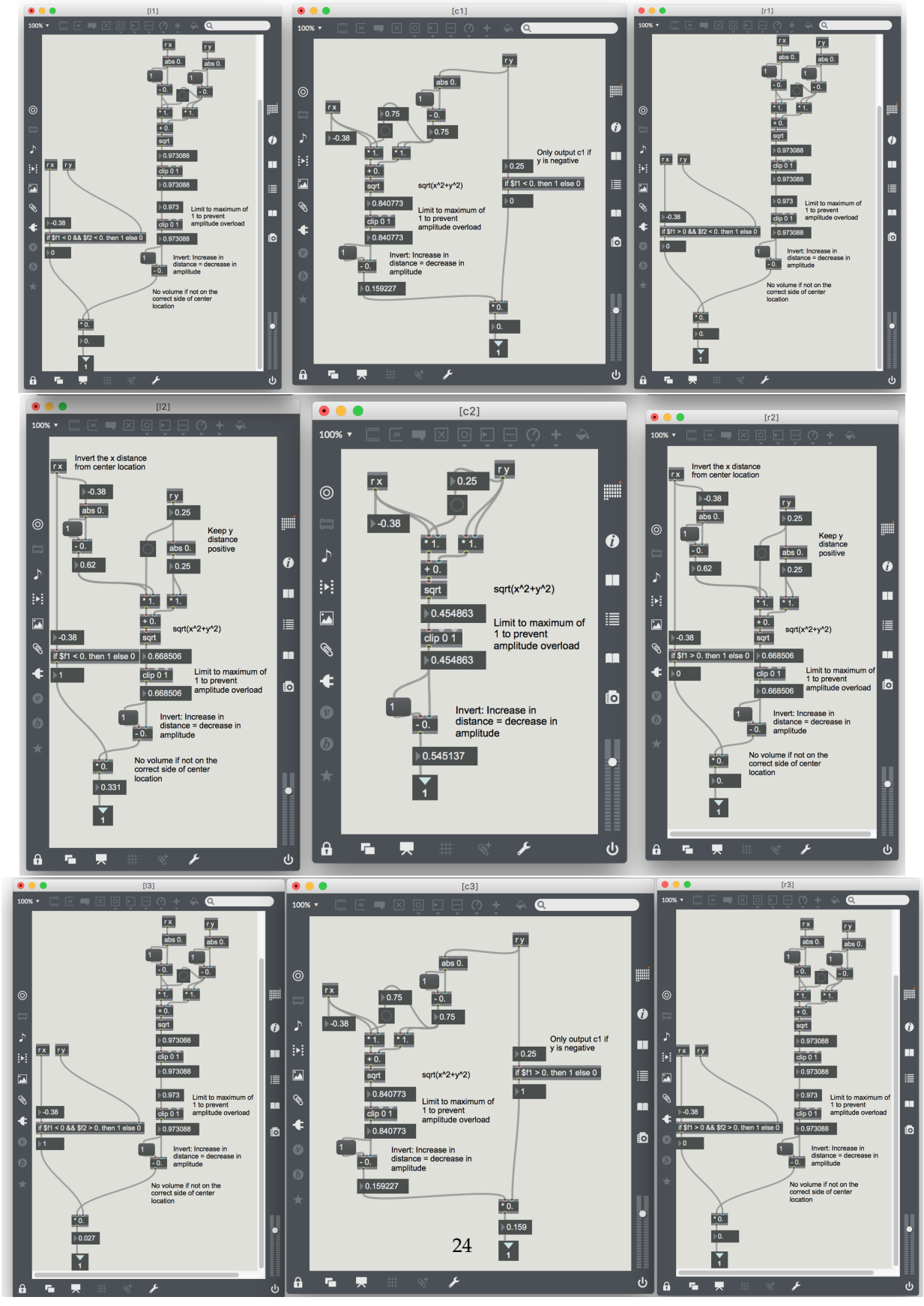




Figure 22: Overview of the individual panning algorithms used in iteration 1.

```
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

# 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).