De La Salle University – Manila
Gokongwei College of Engineering
Electronics and Computer Engineering Department

DIGITAL SIGNAL PROCESSING LABORATORY

LBYCPA4

Term Project Documentation

Generating Adjusted Spatial Audio from Customized HRTF using 3D3A Lab HRTF Database using MATLAB

by

BAYETA IV, Reginald Geoffrey L. - 11811269
TUPAL, Isaiah Jassen L. - 11847115

LBYCPA4 – EQ2

February 21, 2022

Contents

I. Abstract

This paper presents a method of generating adjusted spatial audio from customized HRTF using MATLAB, with inputs of images of ear pairs and audio file, and adjusted spatial audio as output. The custom HRTF is based on the publicly-available 3D3A Lab Head-Related Transfer Function Database. The custom HRTF created is then interpolated with respect to the input desired source position azimuth and elevation values. The adjusted spatial audio is saved into the current workspace directory and the performance graphs are shown as output. The output audio results is accompanied with frequency response, ITD, HRIR, and output audio plots which corresponds to the input azimuth and elevation values, as the desired source position. Six source positions are tested for this study, front, back, right, left, top, and bottom positions relative to the subject's position. To ensure the validity of the results, the project is made publicly available as a GitHub repository which can be improved and scaled by anyone.

Keywords —custom HRTF, spatial audio, 3D3A, MATLAB

II. Introduction

In generating interactive 3D spatial audio, a personalized or custom Head-Related Transfer Function (HRTF) is needed that is tailored for a given individual. An HRTF is defined as the distinct filtration process of sound inputs processed by our body and is utilized to arrange the output sounds as a best-fit spatial audio [1], [2]. Essentially, it's a mathematical model that represents how we actually hear sounds as an individual. HRTFs are measured via different methods such as anthropometric data [3], numerical methods [4], and machine learning [5], among others.

While the term HRTF is relatively new to most people, there are already existing commercial applications of it such as in HyperX Cloud Orbit [6] and Embody: Personalized Spatial Audio [7]. Both products present an easy-to-follow and understand setup to maximize the users' audio experience. It should be a given that both products will cost the user a certain fee to use their service, but there exists a method that will give you a similar experience for free! — and that is *doing it yourself*.

There are a lot of publicly available databases filled with HRTFs as listed in [8], with the file format exported in 'spatially oriented format for acoustics (SOFA). In this paper, we utilize the 3D3A lab HRTF database [1] to generate adjusted spatial audio from customized HRTF in MATLAB. The SOFA MATLAB/Octave API would be used to read and load *.sofa* files in MATLAB [9]. This project is publicly available at GitHub with the following link: **https://github.com/rglbiv/custom-hrtf**.

Table 1. MATLAB functions used for this project

Function	Definition and Usage
figure	Used to create a figure in MATLAB
subplot	By default, figures in MATLAB are in a 1x1 window size, and subplot function is used to divide it into given dimensions
plot	Used to generate a two-dimensional line plot of Y and X inputs
freqz	Used to get the frequency response of a given digital filter
im2double	Used to convert an image input to double form
rgb2gray	Used to convert a RGB image to its equivalent grayscale form
imread	Used to read a given image file
imresize	Used to resize an image file into a given dimension
imadjust	Used to adjust the levelling of a given image
imgaussfilt	Used to apply gaussian filtering to an image
medfilt2	Used to apply medial filtering to a two-dimensional image
edge	Used to apply edge filtering to a given image
imshowpair	Used to show a pair of images in a figure
linspace	Used to create a matrix of linear size n with values from the given range input
sin	Used to declare a sine function in MATLAB

yline	Used to create a horizonal (y) line graph
abs	Used to get the absolute value of the input
permute	Used to swap or rearrange the dimensions of a matrix
length	Used to get the length of a given matrix
max	Used to get the maximum value or element in a given matrix
append	Used to combine multiple strings
int2str	Used to convert integer to string
audioread	Used to read a given audio file
audiowrite	Used to write or save a given audio file
dir	Used to define a directory path
str2double	Used to convert string to double
erase	Used to remove a substring from a string
cell2mat	Used to convert a cell structure to matrix format
isempty	Used to check if matrix is empty or not
ssim	Used to generate the score and map of the structural similarity index of two images, input and reference
maxk	Used to get the Top *k* elements of a given matrix
intersect	Used to get the intersection between two given matrices
contains	Used to check if matrix contains the input element
SOFAload	Used to read and load *.sofa* files by using sofacoustics API [9]
interpolateHRTF	Used to adjust the input HRTF with respect to the desired source position values, in relation to known source position
dsp.FIRFilter	Used to generate an FIR filter via MATLAB DSP System Toolbox
squeeze	Used to squeeze given matrix which will result to reducing the dimension length by 1
conv	Used to apply convolution to input x and y vectors
sound	Used to play sound via computer speaker
*getSoundCHRTF**	Used to generate audio file adjusted using the custom HRTF
*getEarScans**	Read all scanned left/right ear images. Generating this is done by opening all .ply files from [2] and load it in Blender. Next is to crop the left/right ear images with methods listen in [10].
*getSimilarEars**	Get similar ears from scanned ear list, from getEarScans
*getMatchSubjects**	Get the .sofa files of the match indices from the similar ear index from getSimilarEars
*getNewHRTF**	Gets personalized hrtf from hrtf_list from getMatchSubjects
*listenHRTF**	Used to return soundOutput based on input sound and hrtf values from getNewHRTF
*test_customHRTF**	This function is the test function used to generate the figures for the results

** N.B Custom functions. Refer to Section* IV-C

III. Theoretical Consideration

A. 3D3A HRTFs and Morphological Scans Measurement

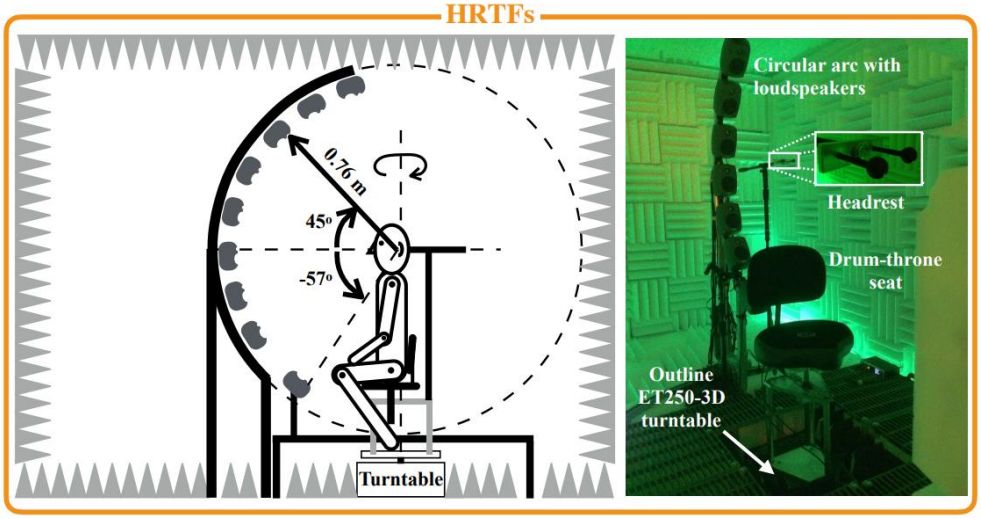


Figure 1. 3D3A HRTF measurement setup [11]

According to 3D3A database documentation [1], the HRTF measurement setup was done in an anechoic chamber room, subject being placed in a turntable, with the HRTFs collected within 648 directions, 72 values for azimuth, and 9 for elevations. Azimuth refers to the left and right direction of the sound source position with respect to the subject's position, while elevation refers to the upward and downwards direction. The azimuth values range from [0, 5, 10, 15, …, 355] and the elevation values are [–57°, –30°, –15°, 0°, 15°, 30°, 45°, 60°, 75°] [1]. The sound source comes from nine (9) Genelec 8010A loudspeakers (hence the 9 elevation values), and sound was recorded using Theoretica Applied Physics and BACCH-BM Pro microphones [11]. The sound being listened by subjects are various exponential sine sweeps, played within 500 milliseconds, with frequency values between 20 Hz to 48kHz, at 96kHz sampling rate [1]. The measured values are in the form of binaural impulse responses (BIRs), and reference impulse responses (RIRs), with the former comes from the binaural microphone measurement placed within the subject, and the latter is derived from the former, which are the measurement for each of the loudspeakers that act as the sound source. These two values are used to calculate the HRTFs using Mesh2HRTF [12] and are exported in *.sofa* file and can be downloaded for free at [2].

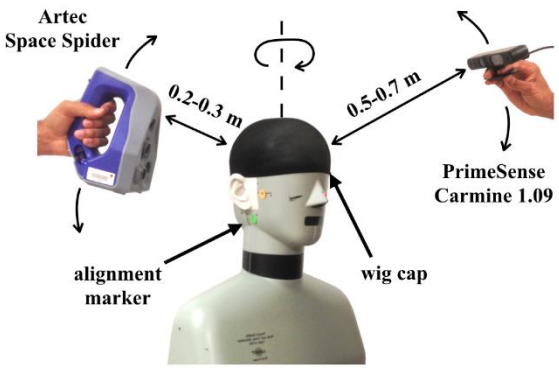


Figure 2. 3D3A 3D head, torso, and ear scanning setup [1]

Aside from the HRTFs measurements, the head, torso, and ear of the subjects were also scanned to keep the morphological models of the subjects that correspond to the HRTF values. The head and torso were scanned using PrimeSense Carmine 1.09, and the subject's pinna was scanned using Artec Space Spider, which used Skanect Pro and Artec Studio 12 Pro, respectively [1]. The scans are exported in *.ply* format and can be downloaded for free at [2].

B. 3D3A Data Processing of HRTFs and Morphological Scans

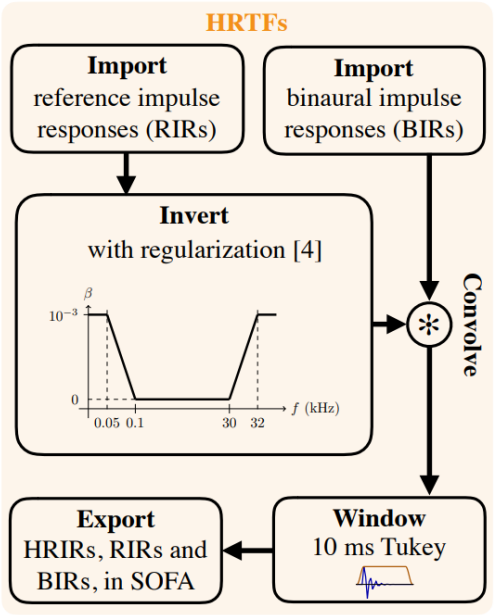


Figure 3. 3D3A HRTFs Measurement Data Processing [11]

By using the measured RIRs and BIRs, the HRIRs is being yield by convolving the inverted RIRs and BIRs, and then applied into a 10-millisecond Tukey Window being exported into *.sofa* files as reflected in [11]. The transfer function for the inversion, as well as related signal processing techniques applied in generating HRTFs are detailed in [1].

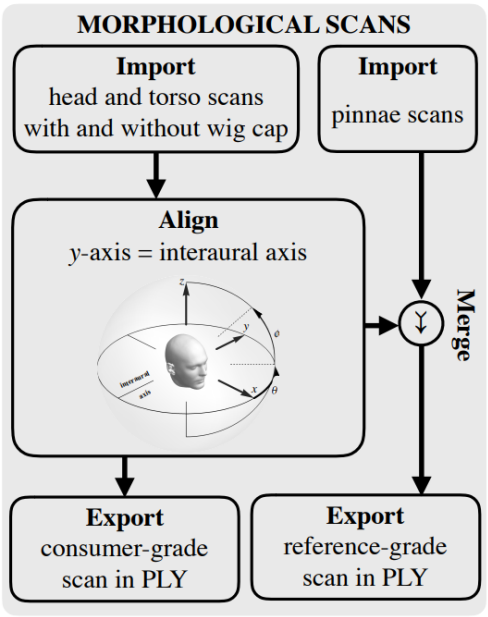


Figure 4. 3D3A Morphological Scanning Process [11]

For the morphological scans, two scans are being combined to produce the final scans: the head/torso scans and the pinnae scans. By using the program partnered with the state-of-the-art scanners, they align the head and torso scans with respect to the interaural axis, and then the resulting scan is being merged with the pinnae scans by replacing the pinnae from the head/torso scans with the scan from the Artec scanner. The process is done by manually removing the pinnae from the head/torso scan and place the higher quality pinnae scan to the head/torso scans via algorithm. The merged scan is now exported into *.ply* files as reflected in [2].

C. Measuring Interaural Time Difference (ITD)

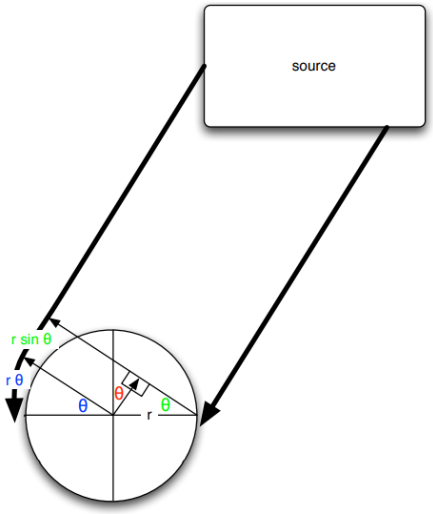


Figure 5. Calculating ITD using Woodworth's Formula [13]

The ITD is the interval between the time when sound enters the left and right ear, respectively. As referenced in [13], the formula to calculate the ITD can be simplified by using Equation (1), although simplified, as also illustrated in Figure X.

$$ITD = \frac{HR(\theta + \sin(\theta))}{c} \qquad (1)$$

wherein *HR* = head radius, wherein a value of 0.0875 meters is used in this project in reference to [13], wherein the value is the average radius of an adult's head; θ = azimuth value or the angle of the sound source with respect to the subject, and *c* = speed of sound (343 meters/second).

D. Getting the Interpolation of HRTF

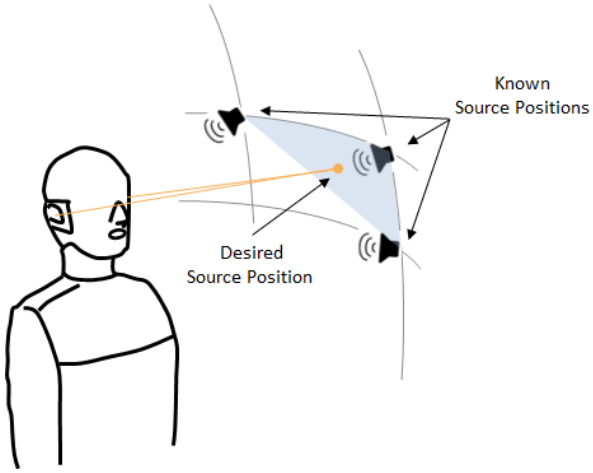


Figure 6. Interpolating HRTF Illustration [14]

The function *interpolateHRTF* from MATLAB Audio Toolbox [14], is used to adjust the input HRTF with respect to the desired source position values, in relation to known source position. In the 3D3A HRTF database, there are 648 known source positions (72 azimuths * 9 elevations), and the said function is utilized to only focus the output with respect to the input desired source position. The function has three (3) input arguments: (1) HRTF, (2) source position, (3) desired source position. For this project, the HRTF is the customized HRTF, the source position is fixed on the database azimuth and elevation values, and the desired source position is dependent on the user, which comprises of a *Nx2* matrix, wherein N is the number of desired positions (azimuth and elevation).

IV. Methodology

A. Project Specifications

In this project, the main objective is to generate an adjusted spatial audio file using a custom-made HRTF, with inputs of images of ear pairs (left, right) and an audio file. The custom HRTF shall be based on the 3D3A Lab Head-Related Transfer Function Database [2].

Input: images of ears (left/right) of a subject
azimuth, elevation value
input audio file to be adjusted
sampling rate of audio file
Output: audio file adjusted using the custom HRTF

B. Experimental Approach

1. Workspace Directory

In order to replicate the output of this project, the project workspace in MATLAB must be the same as shown in the figure below:

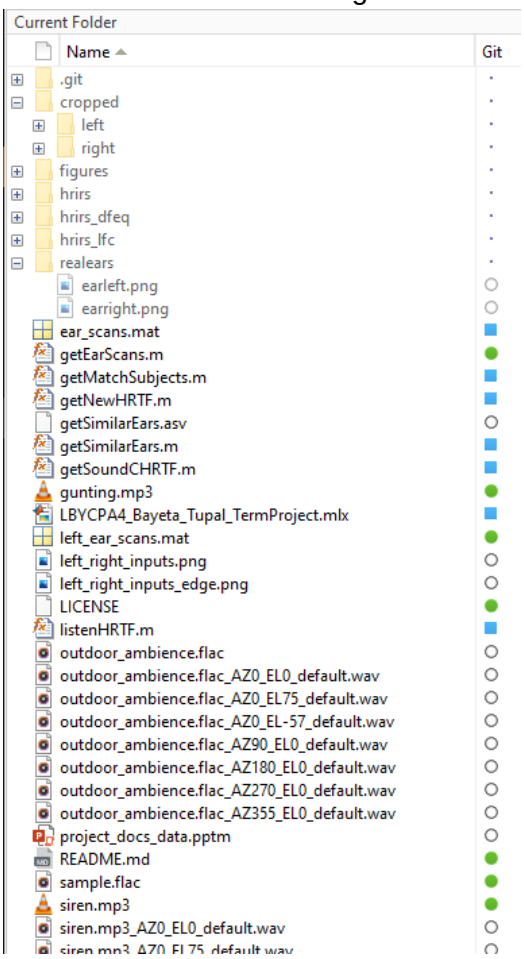


Figure 7. Workspace Directory

Folder description

1. *cropped* – folder that contains the cropped images as discussed in Section IV-B2 of this report
2. *hrirs* – folder that contains the default HRIRs SOFA files from [1].
3. *hrirs_dfeq* – folder that contains the low-frequency extension HRIRs version from [1].
4. *hrirs_lfc* – folder that contains the diffuse-field equalization HRIRs version from [1].
5. *realears* – folder that contains the input ear image pair (left/right)

To save time, this project can be cloned from **https://github.com/rglbiv/custom-hrtf**. To download the *.sofa* file from [2], a Python script, as shown in Figure 8, is provided in each *hrirs*, *hrirs_dfeq*, and *hrirs_lfc* folders that automatically download the files in their respective file locations.

```
# Download HRTFS from https://sofacoustics.org/data/database/3d3a/
import requests
i = 0 # Index
# Default
hrir_def = '_HRIRs.sofa'
# Diffuse-field equalization
hrir_dfeq = '_HRIRs_dfeq.sofa'
# Low-frequency extension
hrir_lfc = '_HRIRs_lfc.sofa'

for x in range(42):
    i += 1
    name = 'Subject'
    name = name + str(i) + hrir_def # Change the hrir_def with what you want

    url = 'https://sofacoustics.org/data/database/3d3a/'+ name
    print('Downloading:    ' + url)

    r = requests.get(url, allow_redirects=True)
    open(name, 'wb').write(r.content)
```

Figure 8. Python script to download *.sofa* files from 3D3A Database

2. Converting 3D3A 3D-Scans to 2D images

The 3D3A database provides 3D scans of the subjects' head, ears, and torso and is saved in *.ply* files. For this project, the *.ply* files are converted to *.png* files by opening it in Blender and taking a screenshot of the workspace. Details on how *.png* files are generated are detailed at [10]. A sample of conversion is shown in Figure 9. The 2D images is saved in the folder *cropped/left* or *cropped/right* depending on the ear type.

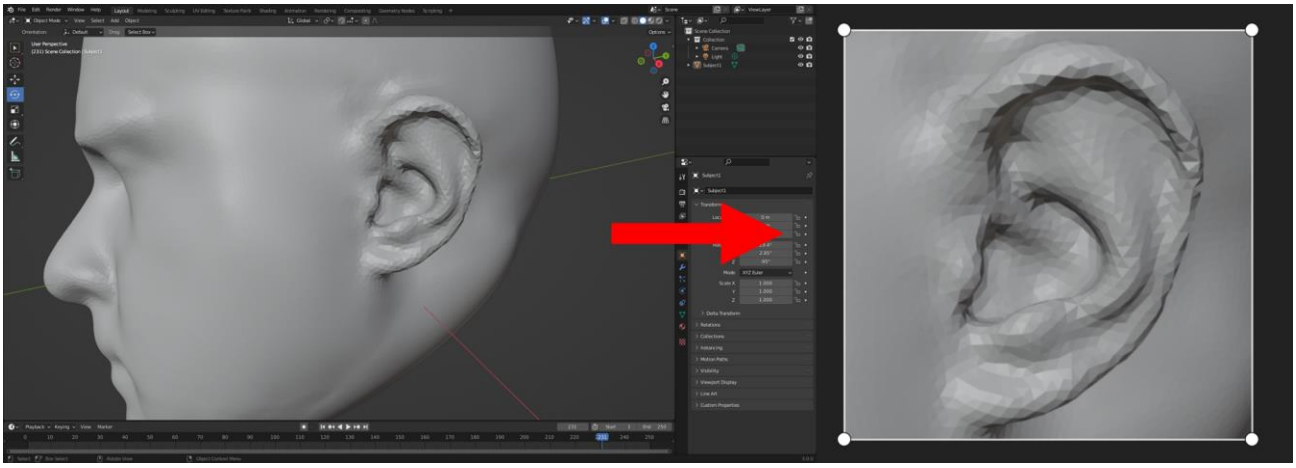


Figure 9. Subject1's head-ear 3D scan to 2D conversion sample

3. Getting or reading the ear scans images

   The next step is to read the converted 2D ear scans in MATLAB. This is defined in *getEarScans.m* function which scans all files and save it into a structure. Each image goes through some image processing wherein it gets converted into grayscale, resized into 500x500 pixels, and applied filters such as *imadjust, imgaussfilt, medfilt2* for adjust, blur, and denoise, and finally edge filtering using Roberts' method to be the same structure with the input. The structured contains the left and right ear scans with data from *cropped/left* and *cropped/right* folders. The processed images are saved in a .mat file named *ear_scans.mat*.

4. Define audio input and reference left/right ears

   As reflected in the GitHub repository [15], there are three (3) audio inputs, siren.mp3 [16], gunting.mp3 [17], and outdoor_ambience.flac (given in the laboratory session). For the input ear images (left/right), the same image processing with *getEarScans* were done to them as reflected in Figure 11.

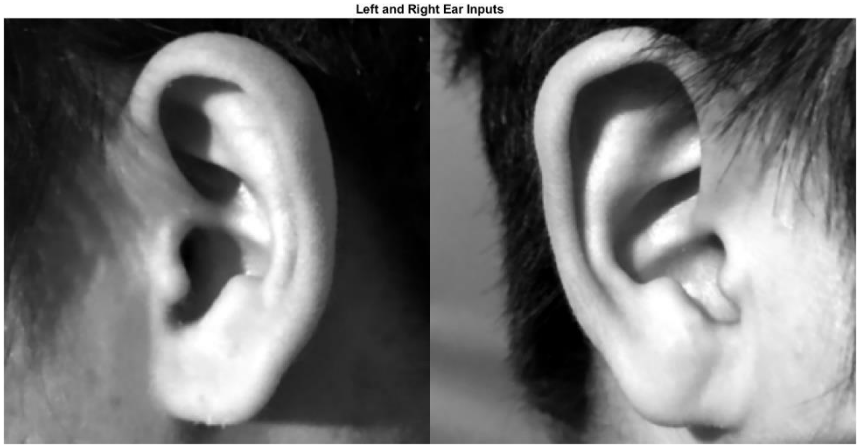


Figure 10. Input reference images

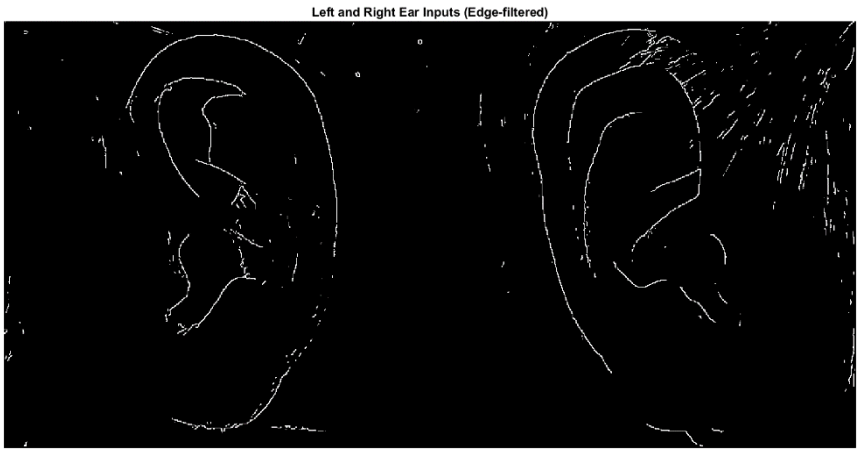


Figure 11. Input reference images after processing

5. Getting the top similar ears from scanned ear list

   Defined in *getSimilarEars.m* function that takes input ear images as reference and compare its similarity with all the ear scans for both left and right ears. Afterwards, it gets the Top 5 results for both ears, return the intersection of the two lists, otherwise if the intersection does not exist, get the Top 2 from both lists. Figure 12 shows the flow or process of *getSimilarEars* function.

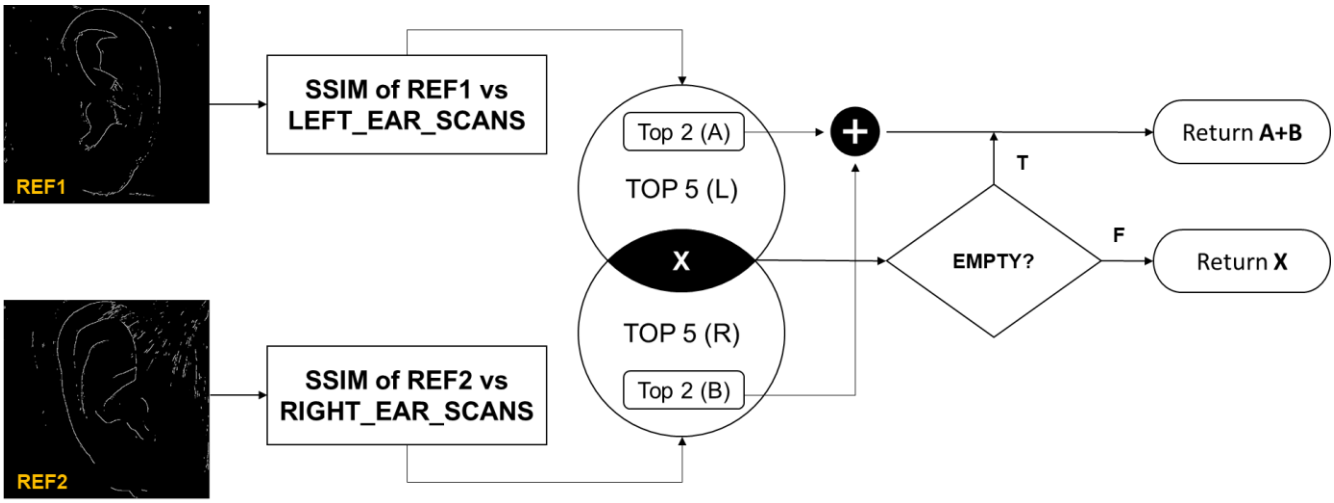


Figure 12. Flow of getting the top similar ears index in *getSimilarEars*

6. Getting the match subjects' HRTF data

   With the returned index from *getSimilarEars*, *getMatchSubjects.m* is the function responsible in loading and getting the HRTF data in the form of list, of the top similar ears with the reference images.

7. Generate the custom or personalized HRTF from the gathered HRTF list

   By using the HRTF list from *getMatchSubjects* as well as the user-defined azimuth and elevation values for the desired position, a custom HRTF is generated by *getNewHRTF.m.* It assigns a weight of 1/N for each match

subject data and utilizes MATLAB *interpolateHRTF* from [14] to finalize the new HRTF.

8. Listen to the spatial sound generated from the custom HRTF

   Afterwards, the function *listenHRTF.m* generates and plays the adjusted spatial audio file as well as returning the filters for left and right ears for frequency response plot.

9. Generate graphs to validate results

   Finally, graph is being generated to visually display the output of this project. Six (6) figures are to be shown once the main function, *getSoundCHRTF.m* is called. one (1) figure for the preprocessed left and right ears, another figure (1) for the ITD plot as discussed in Section III-C, two (2) figures for the frequency response of the left and right channels of the new HRTF, one (1) figure for the impulse response of the new HRTF, and one (1) figure for plotting the generated adjusted audio file.

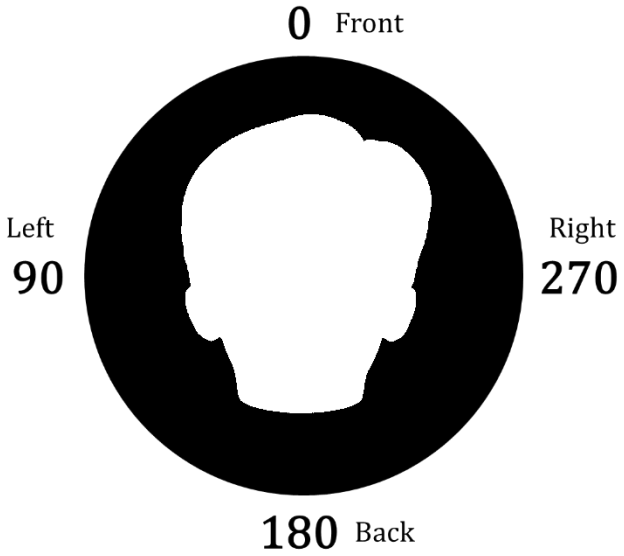


Figure 13. Azimuth Values for Testing

Six (6) tests for the results will be conducted, with desired position at the front, back, right, left, top and bottom Figure 13 shows the azimuth values used for testing the first four tests, while top and bottom tests are done by changing the elevation values (75 degrees for top, -57 degrees for bottom).

C. Source Code

This section will discuss the source code of the project. The main function of this project is called *getSoundCHRTF* which calls the other functions inside its operation.

The following are the summary of the whole code structure, and the detailed explanation comes in the latter part. This is all reflected in the live script file (*LBYCPA4_Bayeta_Tupal_TermProject.mlx*) at the GitHub repository of this project [15].

1. Code Summary

   getSoundCHRTF();

   Generate audio file adjusted using the custom HRTF

   Example:

```
s = getSoundCHRTF('realears/earright.png',
'realears/earleft.png', 'siren.mp3', 90, 30, 'dfeq', 96000, 1);

Saves the file in the format:
Format:  soundfilename_AZXX_ELYY_TYPE.type'
Example: siren.mp3_AZ90_EL0_DFEQ.wav
```

   - **getEarScans();**

     Read all scanned left/right ear images. Generating this is done by opening all .ply files from [2] and load it in Blender.

     Next is to crop the left/right ear images with methods listed in [10].
   - **getSimilarEars();**

     Get similar ears from scanned ear list.

     1. Get top N ssim() results between leftRef and leftEarScans (N=5)
     2. Get top N ssim() results between rightRef and rightEarScans
     3. Get the intersection of 1 and 2
     4. If no intersection, get Top 2 from both
   - **getMatchSubjects();**

     Get the .sofa files of the match indices from the similar ear index.
   - **getNewHRTF();**

     Gets personalized hrtf from hrtf_list
   - **listenHRTF();**

     Returns soundOutput based from input sound and hrtf values

2. MAIN: *getSoundCHRTF.m*

   Inputs:

earsR	– path of the right ear image, e.g., ‘earright.png’
earsL	– path of the left ear image, e.g., ‘earleft.png’
soundInputName	– path of the sound audio file, e.g., ‘siren.mp3’
azimuth	– desired azimuth position
elevation	– desired elevation position
type	– type of HRTF to load, e.g., ‘default’, ‘dfeq’, ‘lfc’
sr	– sampling rate

isListen – plays the audio when function called or not

Outputs:

soundOutput – adjusted spatial audio file based on new HRTF

This function serves as the main function of this whole project. It calls on the functions as stated in the summary (Section IV-C1) respectively. Aside from generating the adjusted spatial audio file as stated in Section IV-A, it also outputs six figures as stated in Section IV-B9 of this report.

The output figure is done using a 3x2 subplot. The left and right ear inputs are plotted using *imshow* function. The frequency response of the new HRTF is generated using the *freqz* function. The impulse response is generated by swapping first the first and third column of the new HRTF as it is of size 1x2x2048 → 2048x2x1 and plotting the two channels on top of each other using *plot*. Finally, the adjusted spatial audio was plotted according to its amplitude, wherein the higher amplitude is plotted first before the other to ensure that both plots are visible. This is done by setting up a flag that check which channel has a louder output and plot the two channels on top of each other using *plot*.

As seen in source code below, the ITD plot is generated by first converting the azimuth values from the 0→355 convention to the range of -90 ≤ 0 ≤ 90. Afterwards, the ITD is calculated using the formula discussed in Section III-C, and is plotted with azimuth values as the X, and ITD in *ms* as the Y.

```
function soundOutput =
getSoundCHRTF(earsR,earsL,soundInputName,azimuth,elevation,typ
e,sr,isListen)
%GETSOUNDCHRTF Generate audio file adjusted using the custom
HRTF
%   Get ear scans
%   Get Match Indices of Match Subject Ears wrt to input
images
%   Get the Match Subject HRTF data
%   Get new HRTF values
%   Listen to the sound input based from the generated HRTF
%   soundOutput =
getSoundCHRTF(earsR,earsL,soundInputName,azimuth,elevation,typ
e,sr,isListen)
%   earsR -> right ear input
%   earsL -> left ear input
%   soundInputName -> audio input filename
%   azimuth   -> azimuth (72 azimuths    : [0°, 5°, 10°, …,
355°])
%   Left << 0 << 180 >> 355 >> Right
%   elevation -> elevation (9 elevations : [-57°, -30°, -15°,
0°, 15°, 30°, 45°, 60°, 75°])
%   type      -> 3 types                 : ['default', 'dfeq',
'lfc']
%   sr        -> sample rate
```

```
%   isListen  -> play audio? 1 if yes, 0 if you use this just
to generate

%% Input Images
% Applied some filters to kind of
% make it similar to the scanned pictures
% Right
r = im2double(rgb2gray(imread(earsR)));
r = imresize(r,[500 500]);           % Resizing
r = imadjust(r);                     % Adjusting the color
scaling
r = imgaussfilt(r,1);                % Blurring a bit
r = medfilt2(r,'symmetric'); % Cleaning some noise
% Left, with same effects applied
l = im2double(rgb2gray(imread(earsL)));
l = imresize(l,[500 500]);
l = imadjust(l);
l = imgaussfilt(l,1);
l = medfilt2(l,'symmetric');

% Edge filtering
r = im2double(edge(r,'Roberts'));
l = im2double(edge(l,'Roberts'));

% Show figure of ears
figure('Name','LBYCPA4 Bayeta Tupal Project')
subplot(3,2,1)
imshowpair(l,r,'montage')
title('Left/Right Ear Inputs')

%% Calculate the Interaural Time Difference (ITD)
% Convert azimuth from 360->180 convention
AZrange360 = 0:5:355;
AZrange180 = linspace(-90,90,72);
AZindex    = AZrange360==azimuth;
AZnew      = (AZrange180(AZindex));

% hr, c values came from
% https://www.researchgate.net/publication/247930825_Under-
explored_dimensions_in_spatial_sound
% formula for itd is also from the same source
hr = .0875;  % head radius assumption, in m
c =  340;  % Approx speed of sound in m/s
itd = hr/c * (AZnew + sin(AZnew)) * 1000; % ms

% Plot the ITD
subplot(3,2,2)
yline(abs(itd)) % Line graph
xlim([0 355]);  % X boundary (Azimuth values)
ylim([0 30]);   % Time in milliseconds
title("Interaural Time Difference is " + abs(itd) + " ms")
xlabel('Azimuth'); ylabel('ITD (ms)');

%% Get Ear Scans
getEarScans();

%% Get Match Indices of Match Subject Ears wrt to input images
```

```
matchIndex = getSimilarEars(l,r);

%% Get the Match Subject HRTF data
hrtf_list = getMatchSubjects(matchIndex,type);

%% Get new HRTF values
new_hrtf = getNewHRTF(hrtf_list,azimuth,elevation);

%% Listen to the sound input based from the generated HRTF
[soundOutput, leftFilter, rightFilter] =
listenHRTF(soundInputName, new_hrtf, sr, isListen);

% Frequency Response
[h1, w1] = freqz(leftFilter);
[h2, w2] = freqz(rightFilter);
% Plot the figures
subplot(3,2,3)
plot(w1/pi,20*log10(abs(h1)))
title('Frequency Response of Left Ear IR')
xlabel('Normalized Frequency (\times\pi rad/sample)')
ylabel('Magnitude (dB)')
subplot(3,2,4)
plot(w2/pi,20*log10(abs(h2)))
title('Frequency Response of Right Ear IR')
xlabel('Normalized Frequency (\times\pi rad/sample)')
ylabel('Magnitude (dB)')


%% Plot HRIR
c = permute(new_hrtf, [3 2 1]); % Swap first and third column
(for plotting)
d = c(:,1)';
e = c(:,2)';

y1 = linspace(0, 5, length(d));
subplot(3,2,5)
plot(y1, d)
hold on
plot(y1, e)
title('HRIR')
ylim tight
xlabel('t (ms)')
ylabel('h')
legend('left', 'right')

% Plot audio (left/right)
t0 = 0: 1/sr : (length(soundOutput)-1)/sr;

% Flag on which channel has louder output
left_flag  = max(soundOutput(:,1));
right_flag = max(soundOutput(:,2));

% Plot the louder signal first
% to ensure both plots are visible
if left_flag > right_flag
    disp('Left is louder')
    % Plot left first before right
```

```
    subplot(3,2,6)
    plot(t0,soundOutput(:,1))
    hold on
    plot(t0,soundOutput(:,2))
    title('Output audio, Left is Lounder')
    legend('LEFT','RIGHT')
    xlabel('t'); ylabel('Amplitude');
else
    disp('Right is louder')
    % Plot right first before left
    subplot(3,2,6)
    plot(t0,soundOutput(:,2))
    hold on
    plot(t0,soundOutput(:,1))
    title('Output audio, Right is louder')
    legend('RIGHT','LEFT')
    xlabel('t'); ylabel('Amplitude');
end
    % Normalize sound output [-1, 1]
    % Otherwise will throw an error of 'Data clipped when
writing file'
    % when called with audiowrite
    soundOutput = soundOutput ./ max(abs(soundOutput));

    %% Save
    outFileType  = '.wav'; % Change this if you want
    % Format soundfilename_AZXX_ELYY_TYPE.type'
    % where XX, YY are values
    % Example siren.mp3_AZ90_EL0_DFEQ.wav
    outFileName  =
append(soundInputName,'_','AZ',int2str(azimuth),'_EL',int2str(
elevation),'_',type,outFileType);
    audiowrite(outFileName, soundOutput, sr);
    fprintf('getSoundCHRTF | Output %s is saved at
directory.\n',outFileName);
end
```

3. g*etEarScans*

This functions traverse through the *cropped/left* and *cropped/right* folders and save all 2D scans image to a structure. The images read are then processed as stated in Section IV-B3 of this report. It will output a display message to the terminal after it is done with all the process. These images will be used for getting the similar ear scans relative to the reference images.

```
function getEarScans()
% GETEARSCANS Read all scanned left/right ear images
% Scan all files and save it into a structure

% Steps
% Define directory paths for left and right
    lpath = dir('cropped/left/*.png');
    rpath = dir('cropped/right/*.png');

% Get all left images
    for j = 1:length(lpath)
        % Define file name
```

```
        fname = append('cropped/left/',lpath(j).name);
        % Get index of file for assignment (basically the
number beside L/R
        cellInd = (str2double(erase(lpath(j).name,'L.png')));
        % Double, grayscale
        img = im2double(rgb2gray(imread(fname)));
        % Resize for uniformity
        img = imresize(img,[500 500]);
        % Adjust,blur,denoise
        img = imadjust(img);
        img = imgaussfilt(img,1);
        img = medfilt2(img,'symmetric');
        % Edge filter using Roberts
        img = im2double(edge(img,'Roberts'));
        % Assign it to left ear container
        left_ear_scans{cellInd} = img;
    end

% Get all right images
    for j = 1:length(rpath)
        % Define file name
        fname = append('cropped/right/',rpath(j).name);
        % Get index of file for assignment (basically the
number beside L/R
        cellInd = (str2double(erase(rpath(j).name,'R.png')));
        % Double, grayscale
        img = im2double(rgb2gray(imread(fname)));
        % Resize for uniformity
        img = imresize(img,[500 500]);
        % Adjust,blur,denoise
        img = imadjust(img);
        img = imgaussfilt(img,1);
        img = medfilt2(img,'symmetric');
        % Edge filter using Roberts
        img = im2double(edge(img,'Roberts'));
        % Assign it to right ear container
        right_ear_scans{cellInd} = img;
    end

% Create struct and save
    ear_scans.left = left_ear_scans;
    ear_scans.right = right_ear_scans;
    save ear_scans.mat ear_scans
    disp('getEarScans | Completed! ear_scans saved at current
directory.')
end
```

4. *getSimilarEars*

This function takes the left and right ear reference images as inputs, as reflected in Section IV-B4, and the similarity index/indices as the output. It loads the ear scans structure from the *getEarScans* function and perform structural similarity index (SSIM) between the reference image and each image (left/right). Afterwards, the Top 5 results between the references and ear scans are collected, and the intersection between the two lists will ideally serve as

the return output for similarity index. If there are no elements within the intersection of the Top 5 lists, the Top 2 indices from the two (totaling to four (4)) will be returned. Finally, it will output a display message to the terminal after it is done with all the process.

```
function simIndex = getSimilarEars(leftRef, rightRef)
% GETSIMILAREARS Get similar ears from scanned ear list
% simIndex = getSimilarEars(leftRef, rightRef)
% leftRef = left reference image
% rightRef = right reference image
% 1. Get top N ssim() results between leftRef and leftEarScans
% 2. Get top N ssim() results between rightRef and
rightEarScans
% 3. Get the intersection of 1 and 2
% 4. If no intersection, get Top 2 from both
% N here is set at 5 (Top 5)

N = 5;

% Load ear_scans.mat
load ear_scans.mat ear_scans

    % Get similarity index of scanned image
    % with respect to reference image LEFT
    for j=1:length(ear_scans.left)
        if ~isempty(cell2mat(ear_scans.left(j)))
            f = cell2mat(ear_scans.left(j));
            g = ssim(f,leftRef);
            lrank(j) = g;
        end
    end

    % Get Top 5 LEFT
    [~,topLeft] = maxk(lrank,N)

    % Get similarity index of scanned image
    % with respect to reference image RIGHT
    for j=1:length(ear_scans.right)
        if ~isempty(cell2mat(ear_scans.right(j)))
            f = cell2mat(ear_scans.right(j));
            g = ssim(f,rightRef);
            rrank(j) = g;
        end
    end

    % Get Top 5 RIGHT
    [~,topRight] = maxk(rrank,N)

    % Get the Intersection
    simIndex = intersect(topLeft, topRight);

    % If no intersection then choose 2 each from Left and
Right
    if isempty(simIndex)
        disp('No intersection found, getting nearest scans.');
        simIndex = [topLeft(1:2) topRight(1:2)];
    end
```

```
    % Display number of similar ears found
    fprintf('getSimilarEars | Found: %d
match/es\n',length(simIndex));
end
```

5. *getMatchSubjects*

This function takes the match index/indices from the *getSimilarIndex* function and type (default, dfeq, lfc) as reflected in Section IV-B1, and outputs a matrix containing the data from the *.sofa* files of the match indices, *hrtf_lists*. For example, if match index = [1, 2, 3, …, N], and type = dfeq, the function will load 'Subject1_HRIRs_dfeq.sofa' … 'SubjectN_HRIRs_dfeq.sofa'. Finally, it will output a display message to the terminal after it is done with all the process.

```
function hrtf_lists = getMatchSubjects(matchIndex, type)
% GETMATCHSUBJECTS Get the .sofa files of the match indices
% hrtf_lists = getMatchSubjects(matchIndex, type)
% Returns the HRTF lists with index from
% matchIndex. For example if matchIndex = [1 2 3], it will
% return Subject1,2,3 files contained in a list
    % Path of HRTFS according to type
    % You can change this if your directory has all the files
    % are not sorted into different folders
    switch type
        case 'dfeq'
            path = dir('hrirs_dfeq/*HRIRs_dfeq.sofa');
            folder = 'hrirs_dfeq/';
        case 'lfc'
            path = dir('hrirs_lfc/*HRIRs_lfc.sofa');
            folder = 'hrirs_lfc/';
        otherwise
            path = dir('hrirs/*HRIRs.sofa');
            folder = 'hrirs/';
    end

    % Get all matched HRIRs
    for j = 1:length(path)
        for k = 1:length(matchIndex)
            % If current file matches subject description
            if
contains(path(j).name,append(append('Subject',int2str(matchInd
ex(k))),'_'))
                fname = append(folder,path(j).name);
                % Display file name
                disp(fname);
                % Load SOFA file
                hrtf = SOFAload(fname);
                % Append loaded file to list
                hrtf_lists{k} = hrtf;
            end
        end
    end

    % Display message
    disp('getMatchSubjects | Loaded matched HRTFs.');
end
```

6. *getNewHRTF*

This function is used to generate the custom HRTF from the data in hrtf_list from *getMatchSubjects* function. Essentially, it put equal weight to each HRTF data in the hrtf_list, where weight = 1/length(hrtf_list). Afterwards, it will get the weighted sum of the HRTF data and source position of all HRTF data from hrtf_list. To generate the custom HRTF, *new_hrtf*, the MATLAB function *interpolateHRTF* from [14] is used. It takes three arguments, HRTF data, source position, and desired position. Desired position comes from the input azimuth and elevation values. Finally, it will output a display message to the terminal after it is done with all the process.

```
function new_hrtf = getNewHRTF(hrtf_list, azimuth, elevation)
% GETNEWHRTF Gets personalized hrtf from hrtf_list
% new_hrtf = getNewHRTF(hrtf_list, azimuth, elevation)
% hrtf_list comes from getMatchSubjects
% that gets HRTFs of matched subjects
% azimuth and elevation are user inputs

    % Uses interpolateHRTF
    % that needs
    % hrtfData (new_hrtf)
    % sourcePosition
    % desiredPosition
    % From
https://www.mathworks.com/help/audio/ref/interpolatehrtf.html

    % Define desiredPosition
    desiredPosition = [azimuth, elevation];

    % Weight
    weight = 1/length(hrtf_list);

    if ~isempty(hrtf_list)
        % Get first Data.IR as init value
        hrtfData = hrtf_list{1}.Data.IR .* weight;
        sourcePosition = hrtf_list{1}.SourcePosition(:,1:2) .*
weight;
        for j = 2: length(hrtf_list)
            % Get the weighted sum of hrtfData and
sourcePosition
            hrtfData = hrtfData + (hrtf_list{j}.Data.IR .*
weight);
            sourcePosition = sourcePosition +
hrtf_list{j}.SourcePosition(:,1:2) .* weight;
        end

        % Gets new HRTF
        new_hrtf = interpolateHRTF(hrtfData, sourcePosition,
desiredPosition);
        disp('getNewHRTF | new_hrtf generated!');
    else
        new_hrtf = [];
        disp('getNewHRTF | Error: hrtf_list is empty!');
    end
```

```
end
```

7. *listenHRTF*

This function returns the adjusted spatial audio file in compliance to the Section IV-A of this report. In addition, it also returns the left and right filters for frequency response. Its inputs are *soundInputName*, *hrtf* (from *getNewHRTF*) *sr* (sampling rate), and *isListen*. The soundOutput was generated in reference to the method demonstrated in [9]. Afterwards, it will play the audio file if isListen is TRUE, otherwise, it will just return the output parameters.

```
function [soundOutput, leftFilter, rightFilter] =
listenHRTF(soundInputName, hrtf, sr,isListen)
% LISTENHRTF Returns soundOutput based from input sound and
hrtf values
% and also left and right filters for frequency response
% [soundOutput, leftFilter, rightFilter] =
listenHRTF(soundInputName, hrtf, sr,isListen)
% sr is sample rate
% isListen is bool if you want to listen to the output sound

    % Display message
    disp('listenHRTF | Playing sound wrt new HRTF...');

    % Filters for frequency response
    leftFilter  =
dsp.FIRFilter('Numerator',squeeze(hrtf(:,1,:))');
    rightFilter =
dsp.FIRFilter('Numerator',squeeze(hrtf(:,2,:))');

    % Load soundInput
    soundInput = audioread(soundInputName);
    % Method to play sound with inputs HRTF, soundInput
    % from https://github.com/sofacoustics/API_MO
    soundOutput = [conv(squeeze(hrtf(:, 1, :)),
soundInput(:,1)) conv(squeeze(hrtf(:, 2, :)),
soundInput(:,2))];
    % Play HRTF-soundInput
    if isListen
        % Play
        sound(soundOutput, sr);
    end
end
```

8. *test_customHRTF*

This function is the test function used to generate the figures for the results. It tests the azimuth and elevation values pair of azimuth = [0, 180, 90, 270, 0, 0]; elevation = [0, 0, 0, 0, 75, -57] with reference to the Front, Back, Left, Right, Top, Bottom of the user; and tested ‘siren.mp3’ for more intuitive results, and ‘outdoorambience.flac’ for the file always used in laboratory activities.

```
% This serves as the test script for the project
```

```
% Generating Adjusted Spatial Audio from Customized HRTF using
3D3A Lab HRTF Database
% Created by BAYETA Ding | TUPAL Isaiah
% This project can be cloned at
https://github.com/rglbiv/custom-hrtf

%% You can edit these values
rightEarInput  = 'realears/earright.png';
leftEarInput   = 'realears/earleft.png';
audioFileInput = 'outdoor_ambience.flac'; % siren.mp3
% 72 azimuths   : [0°, 5°, 10°, …, 355°]
% 9  elevations : [-57°, -30°, -15°, 0°, 15°, 30°, 45°, 60°,
75°]
% 3  types      : ['default', 'dfeq', 'lfc']

% Azimuth-Elevation Pair
% Used for calling the function once
azimuth = 0;
elevation = 0;

% Azimuth-Elevation Pairs
% Used for handling batch processing of results
% Front, Back, Left, Right, Top, Bottom
% azimuth        = [0, 180, 90, 270, 0, 0];
% elevation      = [0, 0, 0, 0, 75, -57];
type           = 'default';
[~, sr]        = audioread(audioFileInput);
isListen       = 0; % 1 if you want to play the sound output
%% Calling the main function once
% type help getSoundCHRTF to see function definition
% or checkout the documentation

s =
getSoundCHRTF(rightEarInput,leftEarInput,audioFileInput,azimut
h,elevation,type,sr,isListen);
%% Batch Processing
% for i=1:6
%    disp('Processing . . .');
%    s =
getSoundCHRTF(rightEarInput,leftEarInput,audioFileInput,azimut
h(i),elevation(i),type,sr,isListen);
% end
```

V. Results

```
Command Window
>> test_customHRTF
getEarScans | Completed! ear_scans saved at current directory.

topLeft =

    26    25    42    27    24


topRight =

    27    23    32    24    38

getSimilarEars | Found: 2 match/es
hrirs/Subject24_HRIRs.sofa
hrirs/Subject27_HRIRs.sofa
getMatchSubjects | Loaded matched HRTFs.
getNewHRTF | new_hrtf generated!
listenHRTF | Playing sound wrt new HRTF...
Left is louder
getSoundCHRTF | Output outdoor_ambience.flac_AZ0_EL0_default.wav is saved at directory.
fx >>
```

Figure 14. Sample Program Execution, *test_customHRTF*

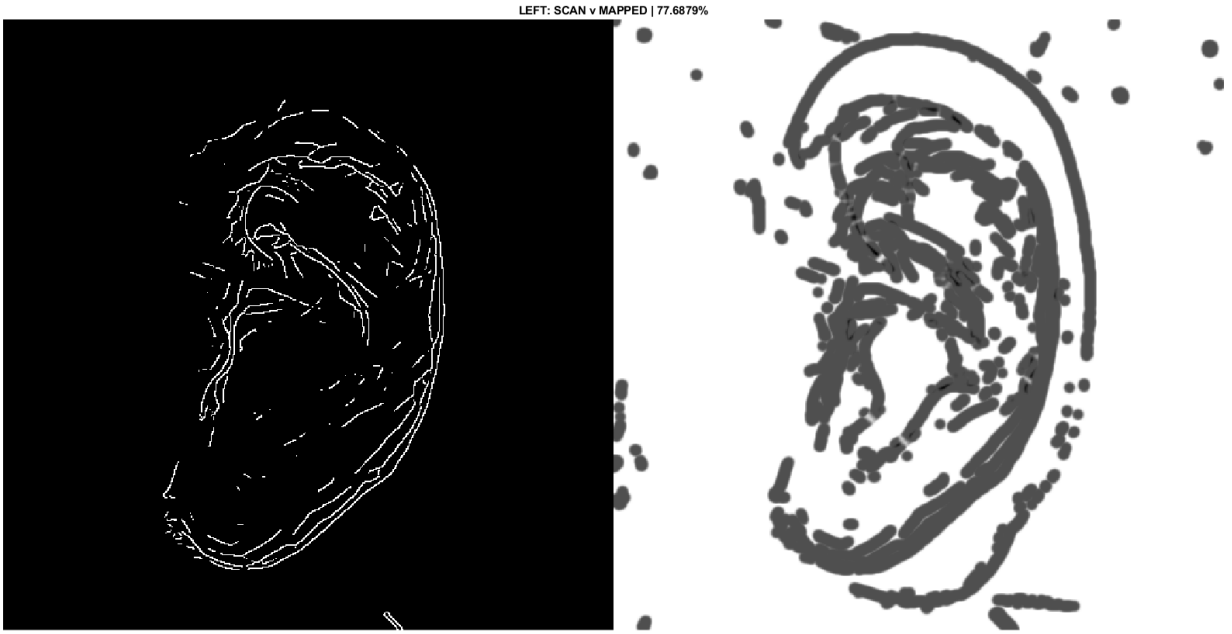


Figure 15. Sample of getting similarity score of input ears vs scanned ears

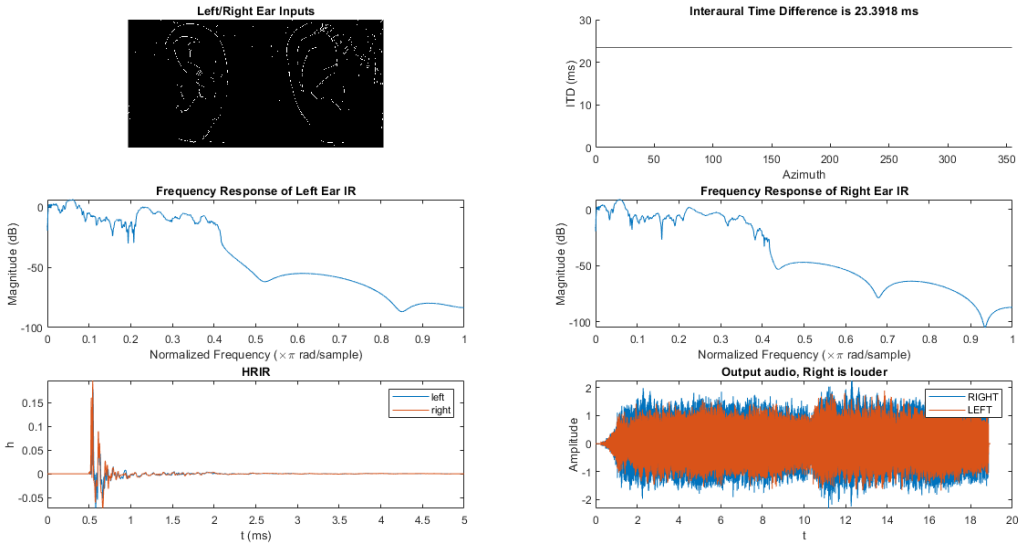


Figure 16. Program output for *siren.mp3*, azimuth = 0, elevation = 0

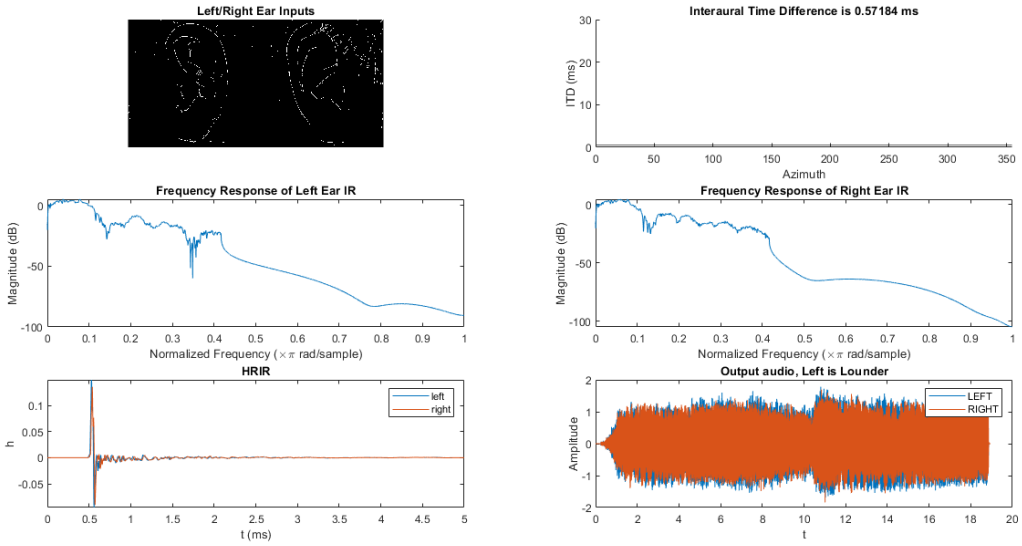


Figure 17. Program output for *siren.mp3*, azimuth = 180, elevation = 0

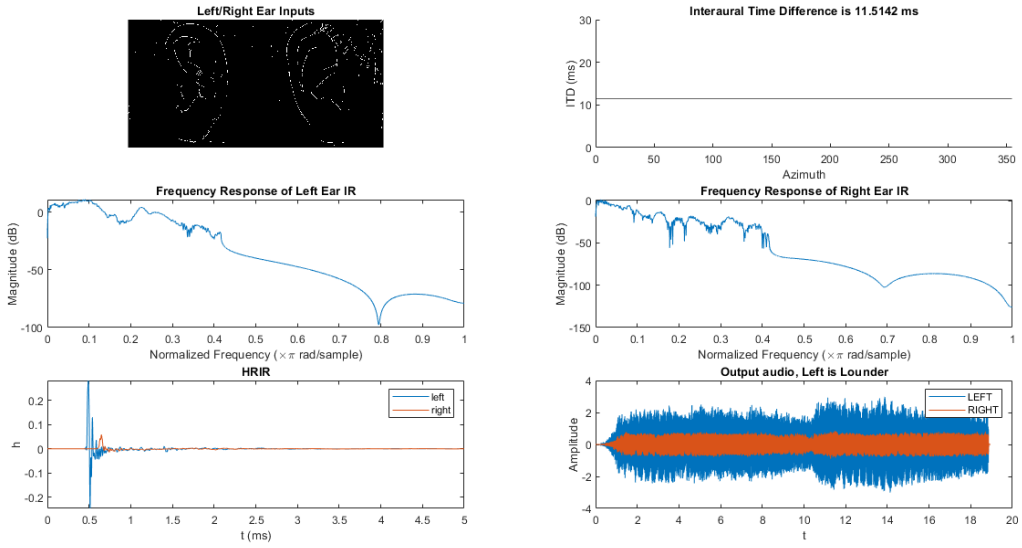


Figure 18. Program output for *siren.mp3*, azimuth = 90, elevation = 0

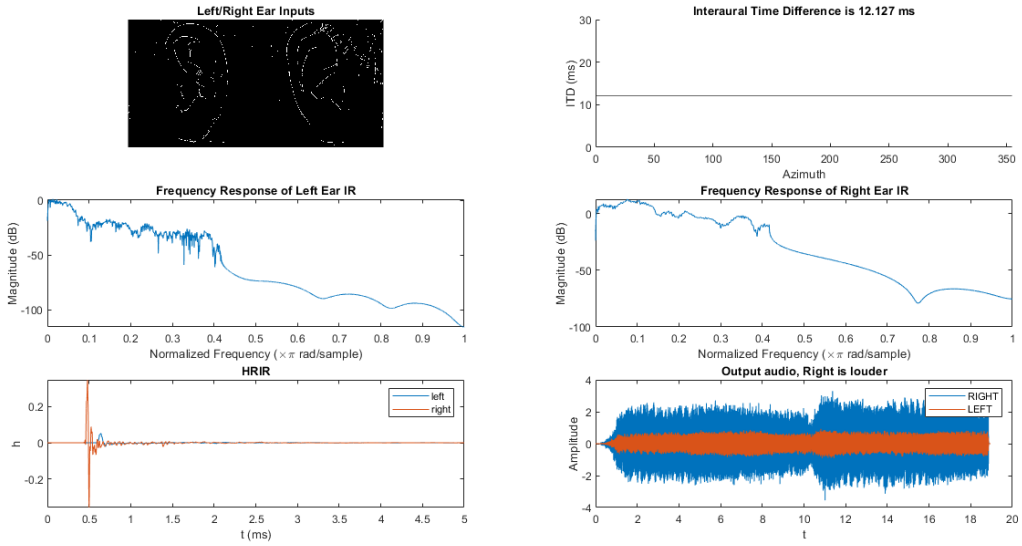


Figure 19. Program output for *siren.mp3*, azimuth = 270, elevation = 0

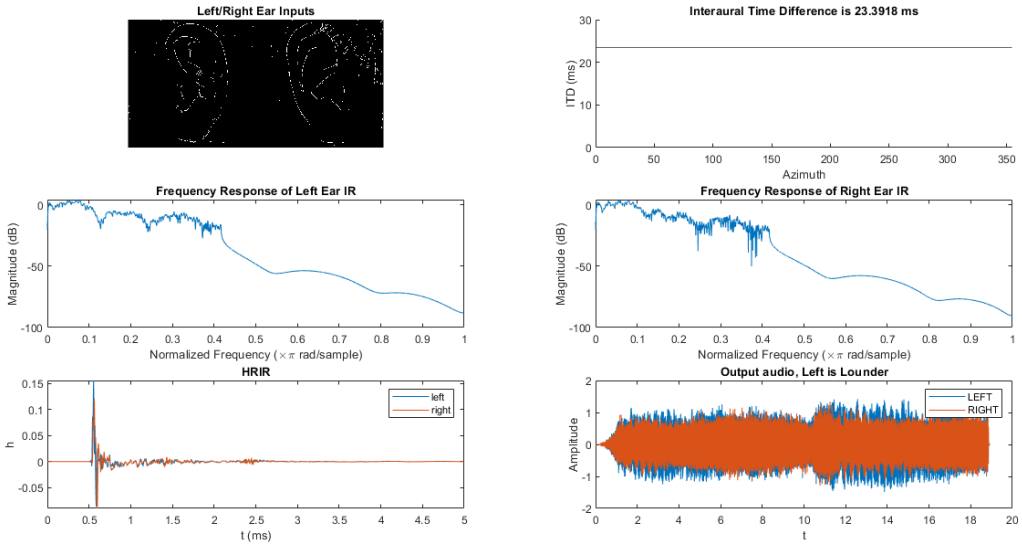


Figure 20. Program output for *siren.mp3*, azimuth = 0, elevation = 75

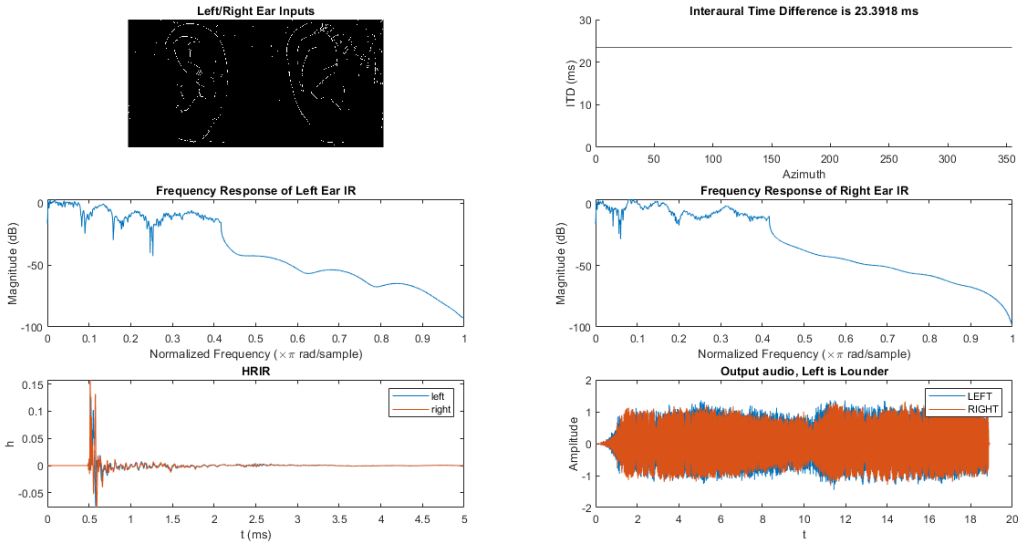


Figure 21. Program output for *siren.mp3*, azimuth = 0, elevation = -57

VI. Discussion

The first problem in this project is to convert the 3D scans of the ear photos from 3D3A database to their 2D version which was done as discussed in Section IV-B2. Afterwards, the steps of this project is highlighted in Figure 14 wherein the program goes through the following steps:

- Read/load ear scans in MATLAB and save it in a .mat file

```
getEarScans | Completed! ear_scans saved at current directory.
```

- Compare the reference input images (right/left) with the ear scans and get the intersection of the Top 5 lists. If there's no intersection, return the Top 2 from both lists.

```
topLeft =

    26    25    42    27    24


topRight =

    27    23    32    24    38

getSimilarEars | Found: 2 match/es
```

- Read/load the *.sofa* files for the corresponding indices from the previous step.

```
hrirs/Subject24_HRIRs.sofa
hrirs/Subject27_HRIRs.sofa
getMatchSubjects | Loaded matched HRTFs.
```

- Generate the new custom HRTF from the *.sofa* files from the previous step.

```
getNewHRTF | new_hrtf generated!
```

- Generate the adjusted spatial sound from the custom HRTF from the previous step.

```
listenHRTF | Playing sound wrt new HRTF...
Left is louder
```

- Lastly, save the generated sound file into the current directory, and show the plot as shown in Figure 16-21.

The top similar ears from the ear scans with respect to the reference input images are done via applying filters, most importantly edge filtering to list of images, as seen in a sample result in Figure 15. Afterwards, the structural similarity index scores are tallied, and the top 5 are selected. In this project, two ear indices was returned (24 and 27), and is used to generate the custom HRTF. The reason why the intersection between the two Top 5 list from both left and right ears are selected is because it can be signified that there's a higher level of confidence that such images are similar to the reference image as it appeared from both list of left and right ears.

The next step is to generate the new HRTF based on the list of *.sofa* files (hrtf_list) with the indices of the top similar ears from the previous step. Afterwards, the custom HRTF was generated by applying equal weight to the HRTF data wherein weight is equal to the reciprocal of the length of the hrtf_list and combining each weighted data into one new_hrtf. Afterwards, the new_hrtf would be then interpolated with respect to the desired

source position based on the input azimuth and elevation values. The resulting output will be used to play the input sound file, which will be the final adjusted spatial audio output.

Figures 16-21 details the testing results for the adjusted spatial audio based on six desired positions: front, back, right, left, top, and bottom positions with respect to the location of the user as illustrated in Figures 1 and 13. The audio file used for this result is the ‘siren.mp3’ which plays a sound of a siren for 13 seconds. This was more preferred than the usual file used in the laboratory, outdoorambience.flac which is 1 minute, 50 seconds long, and testing the former is faster and more intuitive. Nevertheless, the resulting figures for the latter is detailed in Appendix A-F of this report.

Figure 16 shows the graphs for the generated audio file with azimuth and elevation of both 0 degrees. This signifies that the audio source comes from the front of the subject. This is supported by the frequency response of the right and left ear which is similar in form, as well as the HRIR and output audio plot which shows values of left and right ear channels are identical to each other. The ITD results to 23.2918 ms for this demonstration.

Figure 17 highlights the result for the generated audio file with azimuth and elevation values of 180 and 0 degrees, respectively. This refers to the sound source is from the back of the subject. It outputs a similar frequency response with that of the front results, however with slightly lower values in terms of average magnitude and height. It yielded to a 0.57184 ms ITD.

Results for the adjusted spatial audio for left and right positions are shown in Figures 18 and 19. From these graphs, the difference can be easily seen within the frequency response, HRIR, and output audio plots wherein the values are more left-dominated for azimuth-elevation of 90-0, and right-dominated for azimuth-elevation values of 270-0. In addition, listening to the audio file also demonstrates the difference, more so than the front and back adjusted audio file outputs.

Finally, the generated spatial audio for top and bottom positions are highlighted in Figures 20 and 21. The output graphs are similar to that of Figure 16 as the azimuth value of 0 degrees was used, with different elevation values: 75 degrees for the top, and -57 degrees for the bottom positions. The graphs of Figure 16 are smoother and more refine with that of Figures 20 and 21 as the position is at the front. Comparing top and bottom results however, shows that the bottom plots are smoother than the top results, which might resulted from the difference of angle values relative to the front position of azimuth-elevation of 0 degrees. For the audio listening experience, however, it requires an above average earphones to be fully appreciated, which is normal for users who are into customized HRTF experience. Both top and bottom results yielded a 23.3918 ms ITD.

VII. Conclusion

In this project, the main objective is to generate an adjusted spatial audio file using a custom-made HRTF, with inputs of images of ear pairs (left, right) and an audio file. The custom HRTF is based on the 3D3A Lab Head-Related Transfer Function Database. Based on the results, the figures match the desired output relative to the desired input positions. Moreover, to ensure the validity of the results, the project is made publicly available in GitHub and can be cloned to be improved upon and scaled up. For future developments, the researcher will look into more advanced methods of getting the top similar ears with respect to the input ear images, as well as including the pinna details as inputs, and study the code's performance based on many subjects.

VIII. Authors Contribution

Mr. Bayeta (A) and Mr. Tupal (B) first divided the project into two: (1) Preprocessing of input data, and (2) figuring out how *.sofa* and HRTF works. A is assigned in (1), and B is assigned in (2). Task (1) includes the 3D to 2D conversion of the ear scans, as well as reading all through converted 2D images in MATLAB. Moreover, It also include getting the ranking of the top similar ears, as well as the processing of the input and scanned images. Task (2) includes reading *.sofa* files and generating custom HRTF based from the indices from the comparison results, as well as generating the adjusted spatial audio output. After accomplishing the assigned task, both A and B combined the project into one, and fixed bugs and made the code more efficient and readable. For testing the validity of the results, both A and B took the demo presented in class by the instructor as inspiration and plots of ear images, ITD, frequency responses (L/R), HRIR, and output audio plot are shown. To ensure that the project can be easily understood both A and B suggested that the project should be divided into well-documented functions, as reflected in this report. Moreover, they provided a live script file that serve as a code summary of the whole project. As the project is hard to validate as it involves audio file outputs, the whole project workspace is made publicly available in GitHub [15], hosted in A's account.

The authors would also like to thank Mr. Kyle Jomar C. Megino for his time and contribution in the conversion of 3D head-ear scans to 2D as reflected in [10].

IX. References

[1] R. Sridhar, J. G. Tylka, and E. Y. Choueiri, “Convention e-Brief 357 A database of head-related transfer function and morphological measurements,” Accessed: Feb. 15, 2022. [Online]. Available: http://www.princeton.edu/3D3A/HRTFMeasurements.html.

[2] E. Choueiri and R. Sridhar, “The 3D3A Lab Head-Related Transfer Function Database, 3D3A Lab Technical Report #3,” 2021. [Online]. Available: https://sofacoustics.org/data/database/3d3a/.

[3] E. A. Torres-Gallegos, F. Orduña-Bustamante, and F. Arámbula-Cosío, “Personalization of head-related transfer functions (HRTF) based on automatic photo-anthropometry and inference from a database,” *Appl. Acoust.*, vol. 97, pp. 84–95, Oct. 2015, doi: 10.1016/J.APACOUST.2015.04.009.

[4] H. Ziegelwanger, P. Majdak, and W. Kreuzer, “Numerical calculation of listener-specific head-related transfer functions and sound localization: Microphone model and mesh discretization,” *J. Acoust. Soc. Am.*, vol. 138, no. 1, pp. 208–222, Jul. 2015, doi: 10.1121/1.4922518.

[5] M. Zhang, J. H. Wang, and D. L. James, “Personalized hrtf modeling using dnn-augmented bem,” *ICASSP, IEEE Int. Conf. Acoust. Speech Signal Process. - Proc.*, vol. 2021-June, pp. 451–455, 2021, doi: 10.1109/ICASSP39728.2021.9414448.

[6] “HRTF Personalization | Hyperx | HyperX.” https://www.hyperxgaming.com/en/support/hrtf-setup (accessed Feb. 20, 2022).

[7] “Embody.” https://embody.co/ (accessed Feb. 20, 2022).

[8] S. Conventions, “General purpose free-filed databases,” 2022. https://www.sofaconventions.org/mediawiki/index.php/Files.

[9] “GitHub - sofacoustics/API_MO: SOFA Matlab/Octave API.” https://github.com/sofacoustics/API_MO (accessed Feb. 20, 2022).

[10] R. Bayeta, “Conversion of 3D3A Head and Ears Scans to 2D Images using Blender.” https://docs.google.com/document/d/1dPDnza4wmTFADhI1S8DtJcAZ8gN-kO7hDUVUrJbmXx4/edit?usp=sharing.

[11] R. Sridhar, J. Tylka, and E. Choueiri, “A database of head-related transfer function (HRTF) and morphological measurements (Poster),” *Princeton University*, 2017. http://www.princeton.edu/3D3A/Publications/Sridhar_AES143_HRTFMeasurements-poster.pdf (accessed Feb. 15, 2022).

[12] H. Ziegelwanger, W. Kreuzer, and P. Majdak, “Mesh2HRTF: Open-source software package for the numerical calculation of head-related transfer functions,” *Proc. 22nd Int. Congr. Sound Vib. Florence, IT*, 2015, [Online]. Available: https://mesh2hrtf.sourceforge.io/#cover.

[13] M. Cohen, “Under-explored dimensions in spatial sound,” *Proc. - VRCAI 2010, ACM SIGGRAPH Conf. Virtual-Reality Contin. Its Appl. to Ind.*, pp. 95–102, 2010, doi: 10.1145/1900179.1900199.

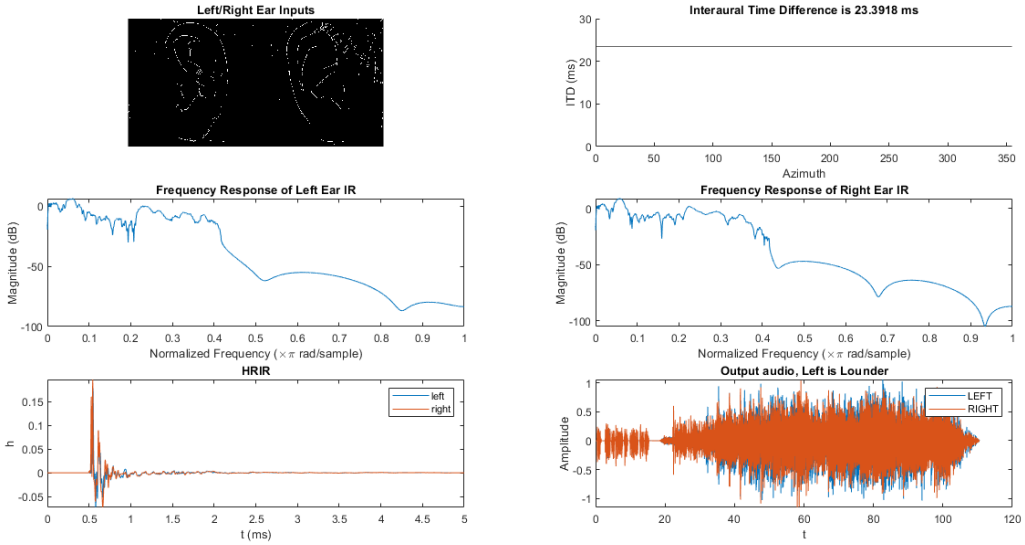
[14] MathWorks, “MATLAB interpolateHRTF.” https://www.mathworks.com/help/audio/ref/interpolatehrtf.html (accessed Feb. 20, 2022).

[15] R. Bayeta and I. Tupal, “GitHub - custom-hrtf: Generating Adjusted Spatial Audio from Customized HRTF using 3D3A Lab HRTF Database,” 2022. https://github.com/rglbiv/custom-hrtf (accessed Feb. 20, 2022).

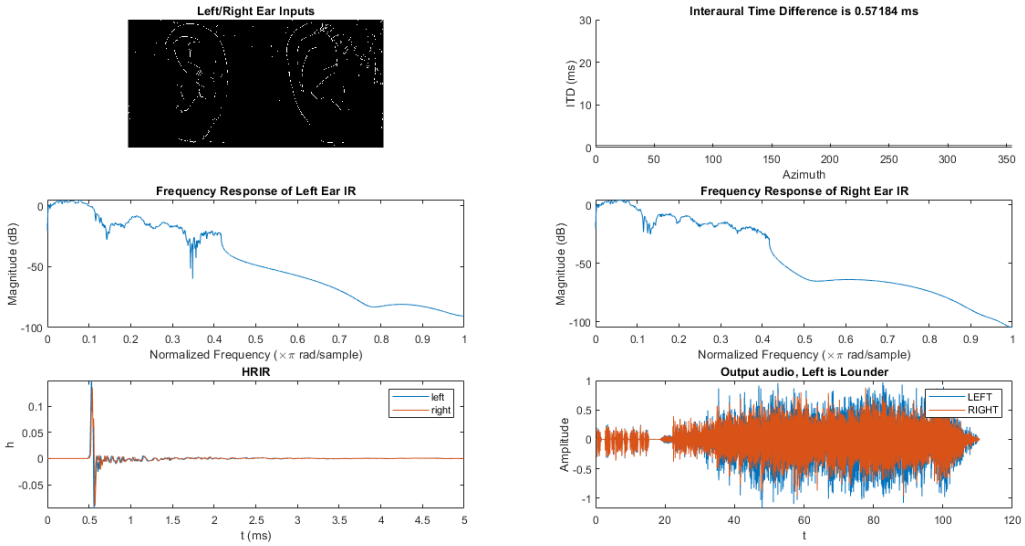
[16] “Siren (sound effect) - YouTube.” https://www.youtube.com/watch?v=heX2mar6R78 (accessed Feb. 20, 2022).

[17] “Scissors Sound Effect In High Quality - YouTube.” https://www.youtube.com/watch?v=2Zqhv2v3bdI (accessed Feb. 20, 2022).

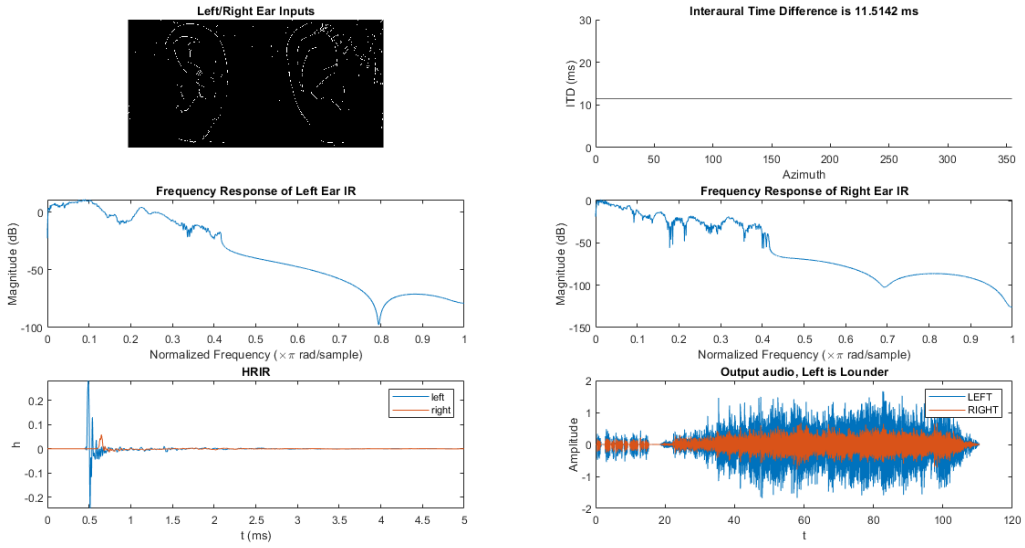
X. Appendix



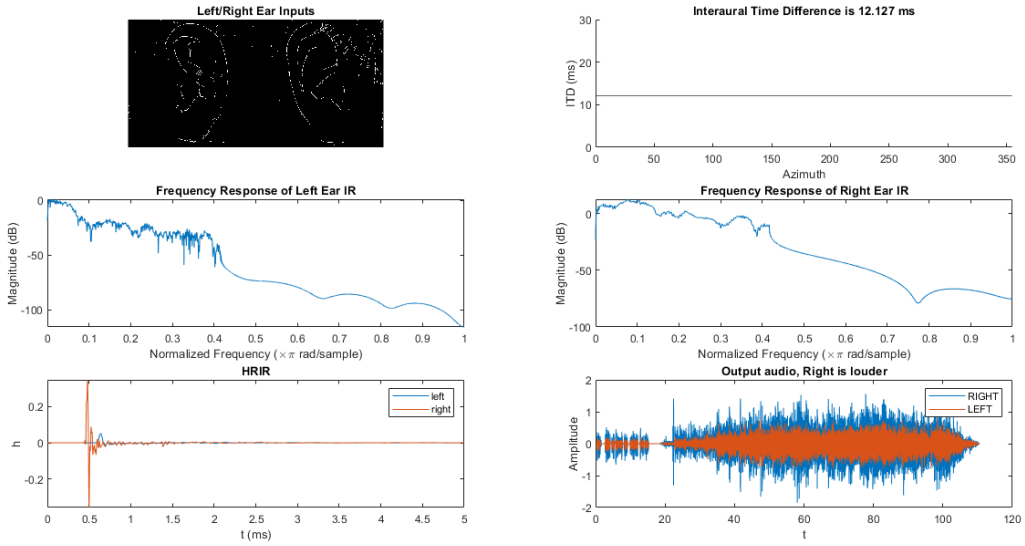
Appendix A. Program output for *outdoorambience.flac*, azimuth = 0, elevation = 0



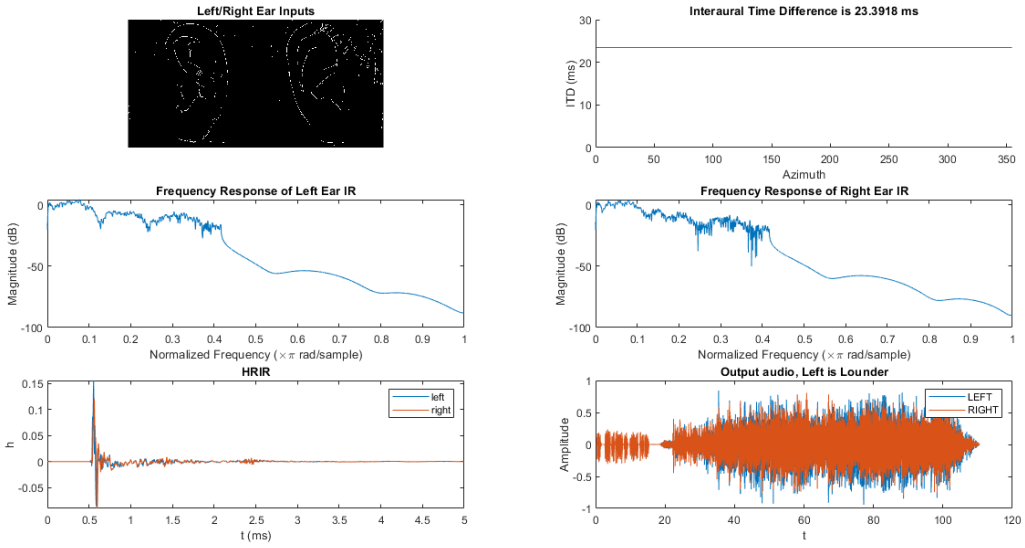
Appendix B. Program output for *outdoorambience.flac*, azimuth = 180, elevation = 0



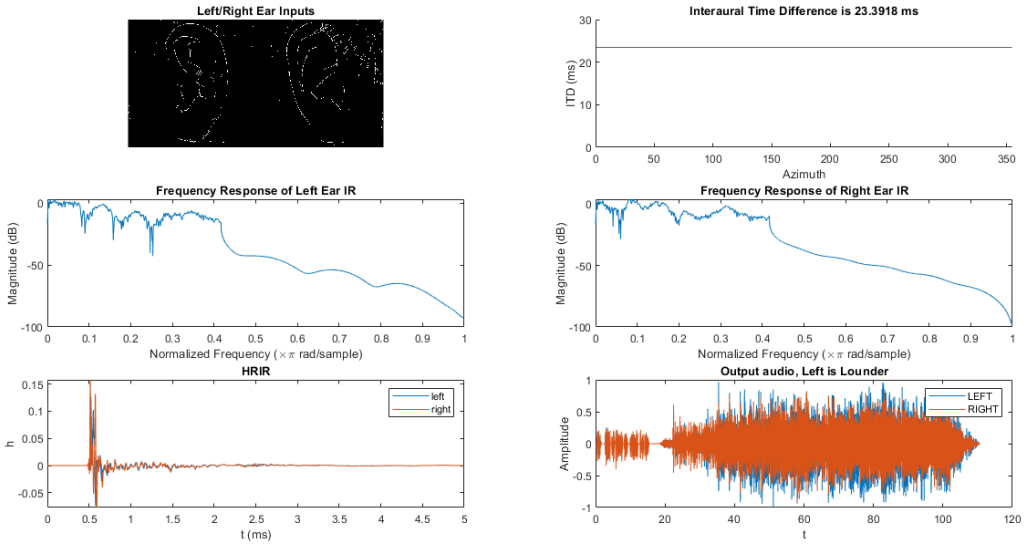
Appendix C. Program output for *outdoorambience.flac*, azimuth = 90, elevation = 0



Appendix D. Program output for *outdoorambience.flac*, azimuth = 270, elevation = 0



Appendix E. Program output for *outdoorambience.flac*, azimuth = 0, elevation = 75



Appendix F. Program output for *outdoorambience.flac*, azimuth = 0, elevation = -57