Digital Audio Effects
Assignment 2: Source Separator

Shih-Ho Wang

March 22, 2019

1 Introduction

1.1 Motivation

When I left my country, came to London studying, the phone telecommunication has became more and more important since I'd like to talk to my family very often. However, I have noticed one thing: the noise or interference is really annoying. Therefore, when I first saw this paper (Barry, 2004), which used a low-complexity algorithm to perform a source separation, I was so excited that I want to implement it as a start to research the noise cancellation and source separation fields.

1.2 Background

The algorithm is called "ADRess", which is abbreviated by "Azimuth Discrimintation and Resynthesis".

The methodology is that when stereo recordings are generated by panning, the only difference between left and right channel is the magnitude. Therefore, try different scaling and comparing with left and right channel combinations, and finally there would be a combination which has the minimum difference of these two channel. Depending on this result, the spatial map can be reconstructed back. Finally, we can choose which position we want to play.

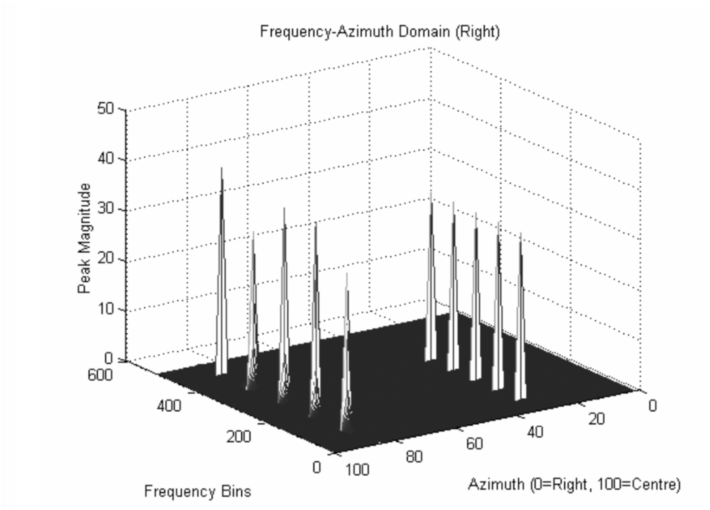


Figure 2: ***The Frequency-Azimuth plane for the right channel. The magnitude of the frequency dependent nulls are estimated. The harmonic structure of each source is now clearly visible as is their spatial distribution.***

Figure 1.2.1: Frequency Azimuth plane

2 Implementation

2.1 Matlab Prototype

We can realise the concept on Matlab first. The rough steps are as following:

1. Do overlap window
   Just as STFT, when analysis and resynthesis, we need to apply a window to the time domain signal, and should overlap add them to compensate the attenuating end of the window.

2. Find azimuth position
   In each FFT frame for each frequency bin, find the best proportion of the two stereo inputs, which means the difference between the scaled stereo inputs is minimum. Assign the best position (according to the proportion) to a dictionary table.

3. Apply magnitude
   For each frequency bin, apply the magnitude only on the best position, leave the other position's magnitude be zero. That is, there is only one position can exist in one frequency bin.

4. Check which position the user choose to play
   Actually, in Matlab prototype, this step in not necessary. Because we can plot all the position in the Matlab figure. But still we can do it. Once we have the overall sound track dictionary table, we can take only the user specified position to reconstruct back to time domain sound track.

5. Re-synthesis
   After the specific position has been chosen, just inverse-fft to reconstruct it back to time domain. Since the stereo structure is destructed by analysis, now we apply the proposed magnitude to both left and right channel. The important thing is, even we changed the magnitude, we still apply the magnitude on the original phases, which can make the time domain reconstruction still be continuous.

Finally, we can build our own frequency azimuth plane, which looks good, in separating positions.

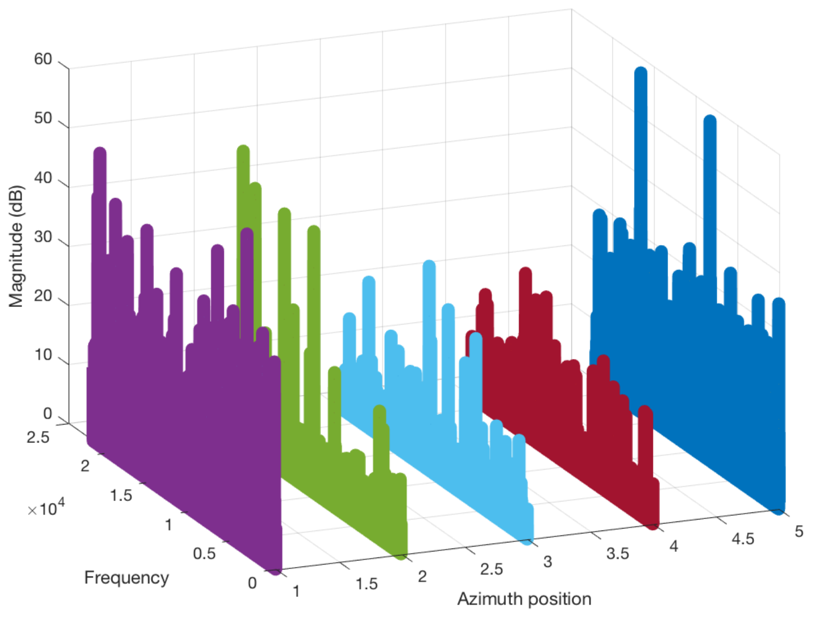


Figure 2.1.1: Frequency Azimuth plane in Matlab prototype

2.2 Implement on Juce

2.2.1 Causality on overlap-add

The biggest difference between implement on Matlab and Juce is, on Matlab, we can use the whole length of audio track without considering causality, but Juce can not. In addition, on Juce, there is a real-time processing requirement, even though it is not necessarily, but still better considering it at first.

To analysis-process-resynthesis, just like doing STFT, it is necessary to overlap and add the consecutive windows. When considering causality, it comes out that even when the previous window has been processed already, it cannot be overlap-add to output directly, until the next window is also ready.

However, when waiting for the next window being ready, the process of recent window still need to be working. That is to say, the processed data should be stored in the other buffer, waiting for the appropriate time to deliver, which is when the next window is ready.

2.2.2 Overlap Window

The type of window we use for FFT and overlap should be carefully considered. The overall summation of each window should perfrom a flat gain. In this project, the triangular window is chosen. As a triangular window, the hop size can be set at half of the window size. The corresponding overlap-add triangular windows are examined no issue as following figure:

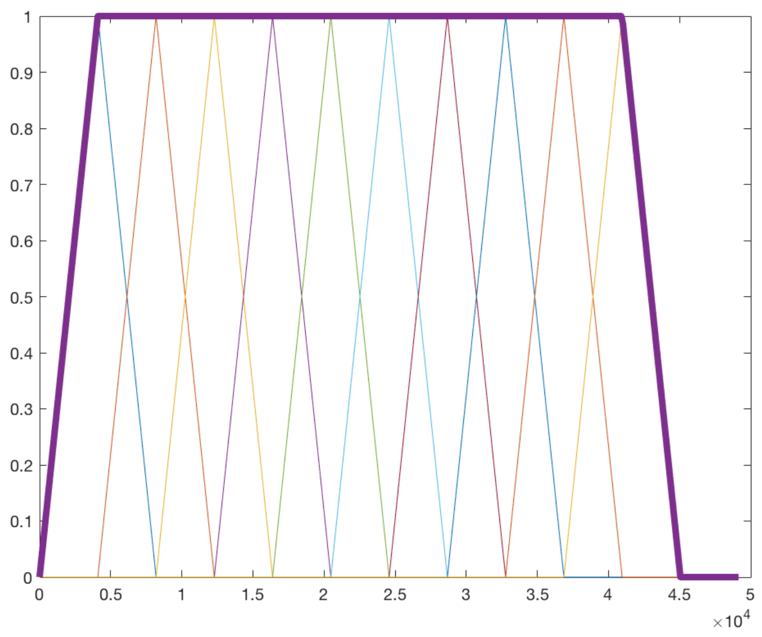


Figure 2.2.1: Triangular window overlap

2.2.3 Timing flow diagram

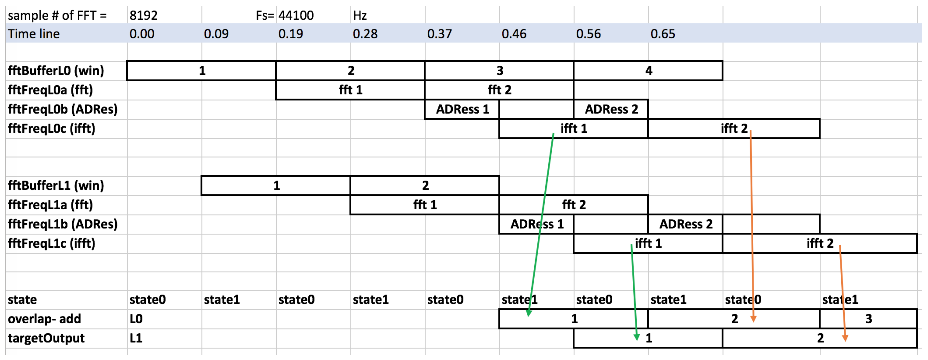

sample # of FFT = 8192
Fs= 44100 Hz
Time line 0.00 0.09 0.19 0.28 0.37 0.46 0.56 0.65
fftBufferL0 (win) 1 2 3 4
fftFreqL0a (fft) fft 1 fft 2
fftFreqL0b (ADRes) ADRess 1 ADRess 2
fftFreqL0c (ifft) ifft 1 ifft 2
fftBufferL1 (win) 1 2
fftFreqL1a (fft) fft 1 fft 2
fftFreqL1b (ADRes) ADRess 1 ADRess 2
fftFreqL1c (ifft) ifft 1 ifft 2
state state0 state1 state0 state1 state0 state1 state0 state1 state0 state1
overlap- add L0 1 2 3
targetOutput L1 1 2

Figure 2.2.2: Proposed timing flow diagram

Above is the proposed timing diagram. Since the hop size is designed in half of the window size, there should be two buffers processing overlap window in the same time. After that, the two independent processing procedures can successfully produce processed data, and add up together as a well-reconstructed audio signal.

The buffers are described in detail as followed:

1. fftBufferL0
   It is the sample buffer, iterating each input sample element and store into this buffer until the amount reaches the fft window size, which by default is 8192.

2. fftFreqL0a
   Do fft, and store the fft data in this buffer.

3. fftFreqL0b
   Do ADRess algorithm. In this function, the inputs should be the fft data from both left and right channels.

4. fftFreqL0c
   Do inverse-fft, to trasnform back to time domain signal.

5. fftBufferL1, fftFreqL1a, fftFreqL1b, fftFreqL1c
   The function is totally the same as above named L0*. Just the other part for storing overlap data.

6. targetOutput
   overlap-add with fftFreqL0c and fftFreqL1c, then the reconstruction process is done.

Finally, move the targetOutput data one by one into the original audio sample buffer, then it can be played.

Be aware, since every buffer has its own process time, and the pipeline processing method will accumulate the processing time. The 2nd row of F fig.**??** shows the corresponding time. Obviously, there is 0.46 millisecond latency for this process.

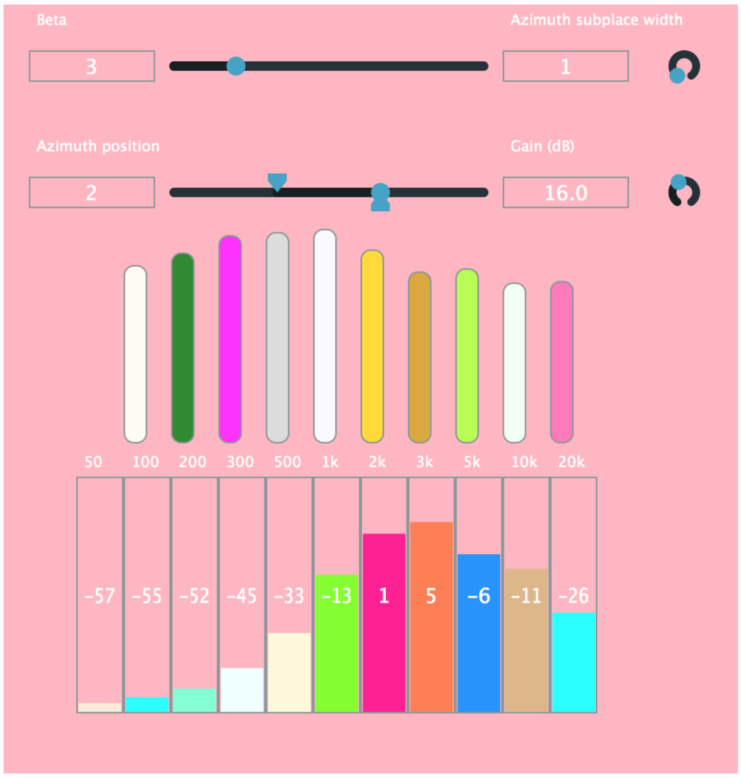
Note that, the FFT and IFFT function, this project uses the inherited FFT function Juce dsp module.

2.2.4 Equaliser

Since ADRess algorithm is a relatively simple algorithm for source separation, sometimes it is not enough effective, especially when the sounds have too much reverb. Therefore, I add an FFT based equaliser, in order to let users be able to manually manipulate the sound track. To elaborate, if there was some noise or interference cannot be separated by ADRess but users know the frequency, then users can directly mute that specified band to eliminate the interference. Or even users don't know which band the interference belong to, there are still lots of chances to try and manipulate.

The equaliser has 11 bands, and all of them can be adjusted separately. Thanks to the ADRess processing which once has to transfer to frequency domain, the equaliser can simply be applied on each corresponding frequency bins. Since the equaliser just changes the magnitude of the spectrum, there is no any phase concern.

3 Parameters


Beta: 3

Azimuth subplace width: 1

Azimuth position: 2

Gain (dB): 16.0

50	100	200	300	500	1k	2k	3k	5k	10k	20k
-57	-55	-52	-45	-33	-13	1	5	-6	-11	-26

Figure 3.0.1: User Interface

The user interface can be briefly separated into two parts: the upper part is ADRess algorithm, and the lower part is equaliser.

1. Beta
   Beta in ADRess algorithm, means how many azimuth position should we divide into. Since there are left and right part, the number of position will be $2 * Beta - 1$

2. Azimuth position
   Choose which azimuth position should be played. Note that the maximum and minimum value of this parameter will be auto adjusted when Beta is changed. As described in Beta, the azimuth position will be limited between $-(Beta - 1)$ to $+(Beta - 1)$.

3. Azimuth subplace width
   Select how many azimuth subplaces should be included, by the centre of the chosen azimuth position. This parameter is often used when Beta is large, greater than 5 for example, which means the overall sound track has been divided into many azimuth position. Thus, in each position, there might be too few data since the power has been diffused into nearby positions. Choose a relatively bigger width can include more nearby position into magnitude calculation.

4. Gain
   Adjust overall gain. When separated sources and played, the magnitude might be too small. So try to add 10-15dB when the sound volume is too small. The gain in dB will first transform to linear scale, and then multiplied to the amplitude in time domain.
   $gainLinear = 10^{gainDB/20}$

5. Real-time FFT display
   In the middle part of UI. There are 11 bars which represent every band's instant magnitude.
   The magnitude value is updated by every fft calculation. In each separated band, the maximum magnitude will be stored into a buffer. Thus, when timercallback has been called, the displayed magnitude will be updated from the buffer. The update rate follows the system callback rate, which is 50ms by default.

6. Equaliser
   In the bottom part of UI. There are 11 bars to control the volume of the bands. The real-time FFT display detects the dry signal, which means before applying on equaliser. So when changing equaliser, the real-time FFT display won't change its amplitude but the sound has changed. The gain in dB will first transform to linear scale, and then multiplied to the amplitude on the corresponding frequency bins.
   $gainLinear = 10^{gainDB/20}$

4 Evaluation

4.1 Test manipulated sine input

At first, we should examine the basic algorithm based on mathematics. As a start, we create an multi-sine-wave input and panned to different azimuth for testing.

First, a 3537 Hz sine-wave with harmonics, the amplitude is 0.69. Second, a 19101 Hz sine-wave with amplitude 0.93. The last one, a 8317 Hz sine-wave harmonic series with amplitude 0.85.

The pan configuration is as followed: signal 1 is panned 1/4 to the left channel, signal 2 is panned 2/5 to the left channel, and signal 3 is panned 3/4 to the left channel.

The signal lasts for 5 seconds identically. The original FFT figure is:

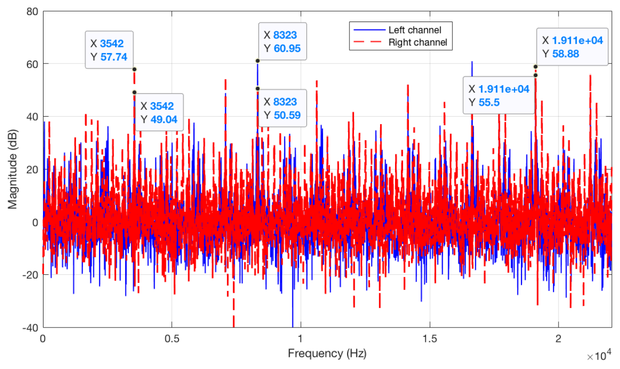


Figure 4.1.1: Panned stereo test input

By theoretic calculation with its panned proportion, beta should be chosen to 3, and the corresponding position will be: signal 1 is at position -2, signal 2 is at position -1, signal 3 is at position 2.

After doing ADRess effect and choosing the position to be -2, the frequency response of the output signal is:

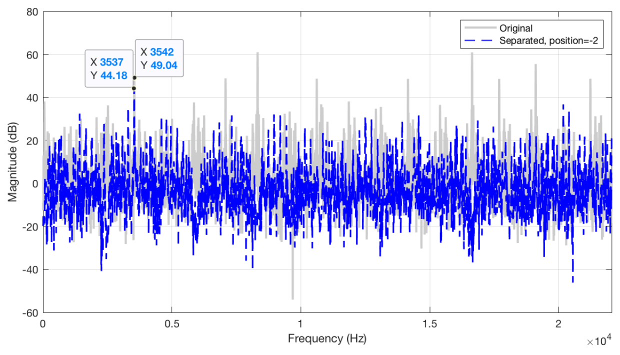


Figure 4.1.2: ADRess effect output. Choose position -2, there should be only signal 1 remained.

There is only signal 1 stayed as expected.

Choose another position, which is position -1. There might only signal 2 stayed, and it has been proved as the following figure:

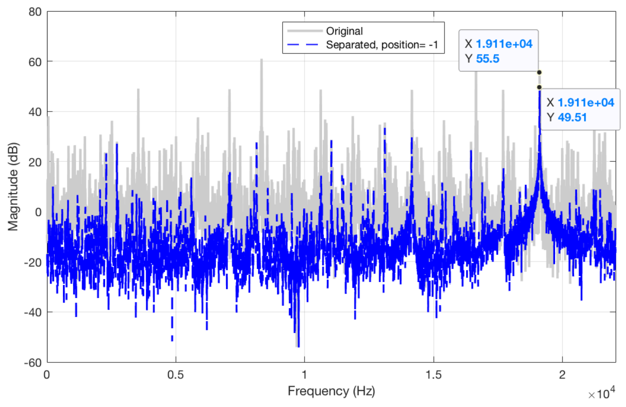


Figure 4.1.3: ADRess effect output. Choose position -1, there should be only signal 1 remained.

Overall, the ADRess algorithm is as expected, successfully separating sources at different positions. Then we can start to analyse the real music recording.

4.2 Test music recording

The input music recording is a chrous contains two vocal part and some instrument. We should take a look at the azimuth plane from Matlab first:

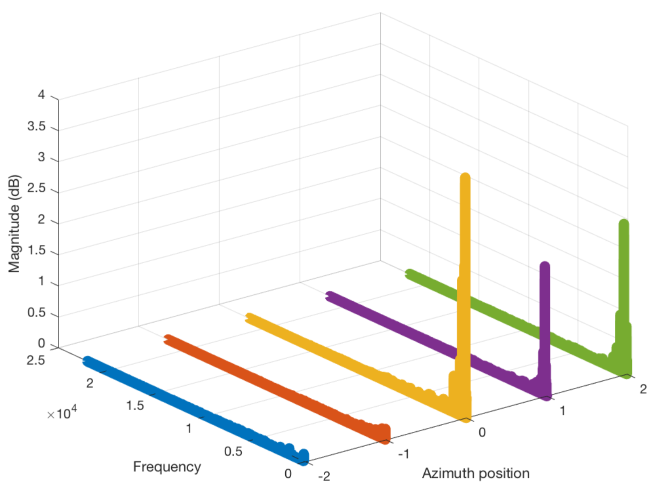


Figure 4.2.1: Azimuth plane of music recording "Fragile"

From the azimuth plane figure, there are three positions contain signals. As a result, we can apply ADRess on these positions and see how they perform. However, since this is a complicated music recording, which is not manipulated by special simple sine tones only, the FFT plot might not be easy to tell, neither the music signals in time domain. The best way to check the separation performance is to listen to the output sound track directly.

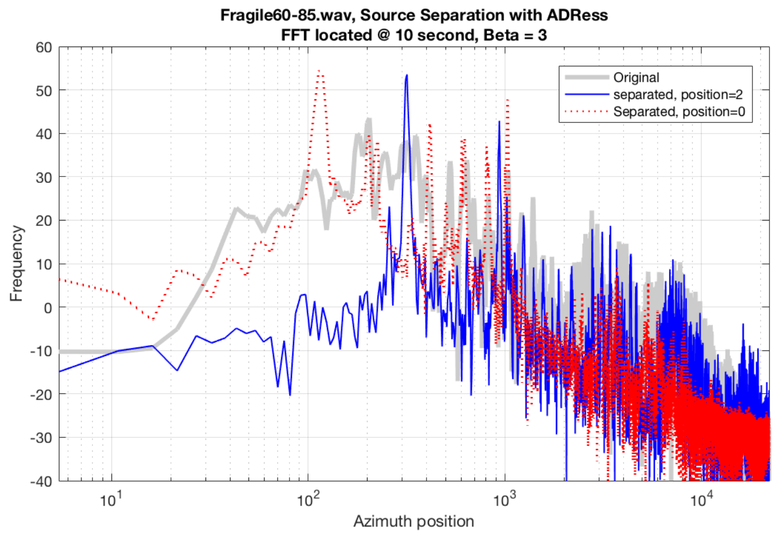


Figure 4.2.2: Fragile60-85.wav audio file FFT in original, ADRess position 0, and ADRess position 2

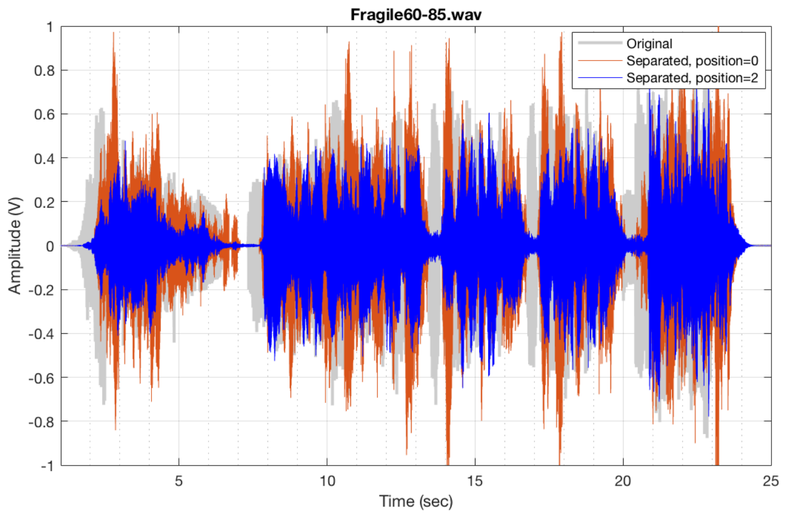


Figure 4.2.3: Fragile60-85.wav audio file time signal in original, ADRess position 0, and ADRess position 2

In time domain signal, we can also observe that there is about 0.5 letancy between original and separated signals, which is expected as discussed previously on fig.2.2.2.

5 Note

1. The juce code should import the juce_dsp module.
2. If when building, it showed "complex is not found in std (or something like that)", my solution is:
   i. In Xcode, go to "project"
   ii. "Building settings"
   iii. "Search Paths"
   iv. "System Header Search Paths"
   v. add: /usr/local/include

   In my /usr/local/include, there are some .h file which were prdouced when I first installed "fftw3"