Columbia University, Fall 2007

DIGITAL SIGNAL PROCESSING

REMOVING VOCALS FROM COMMERCIAL TRACKS

An ELEN E4810 DSP Project by Jaime Peretzman and Shrivathsa Bhargav

Table of Contents

Introduction 2
Bandstop filtering 2
Stereo Cancellation 2
Audio Blind Source Separation 3
Selection of song 3
Short Time Fourier Transform (STFT) and spectrogram 3
Track Identification 4
Time Frequency Mask – Binary Method 4
Time-signal recovery – Inverse STFT (ISTFT) 5
Conclusion 6
References 6
Acknowledgements 6
Appendix A – Matlab Code 7

Removing Vocals from Commercial Tracks

Introduction

Since the advent of commercial music, songs have commonly been identified by the singer's voice. Popular tracks are distinguishable by an extraordinary lead voice. Meanwhile, songs with great instrumental backgrounds can be marred by an unfortunate selection of vocalists and lyrics. The goal of this project is to remove vocals from commercial tracks in order to appreciate the underlying instrumental background.

Moreover, the removal of vocals has several applications. Karaoke is a common pastime in which the removal of vocals permits the participants to effectively interact with the song. Additionally, removing vocals makes the production of ringtones and remixes easier. Finally, people may just wish to hear the track without the lead singer's voice.

Consequently, in order to remove vocals from commercial tracks, throughout this report, we are going to cover several techniques. These techniques include filtering the known (average) frequency range of the human voice (bandstop filtering), canceling common frequencies between stereo channels (stereo cancellation), and masking a time frequency spectrogram (audio blind source separation). These techniques are described in detail below.

Bandstop filtering

The simplest technique for removing vocals is stopband filtering. The human voice has a distinct frequency range between 300 Hz and 3 kHz. By applying a bandstop filter (figure 1) at these frequencies, most vocals can be removed. Software filtering allows us the luxury of implementing a very high order filter. Unfortunately, however, this technique has the side-effect of also removing any instruments that occupy the same frequency range, such as strings and guitars. This is undesirable.

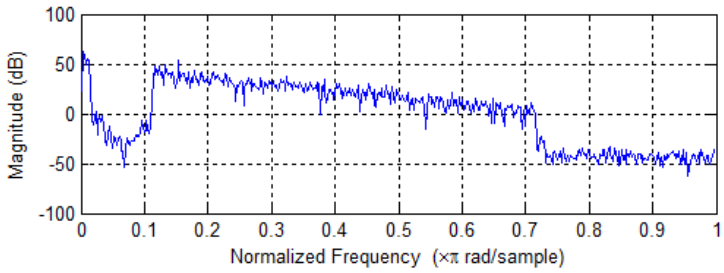

Figure 1 - The result of applying a bandstop filter

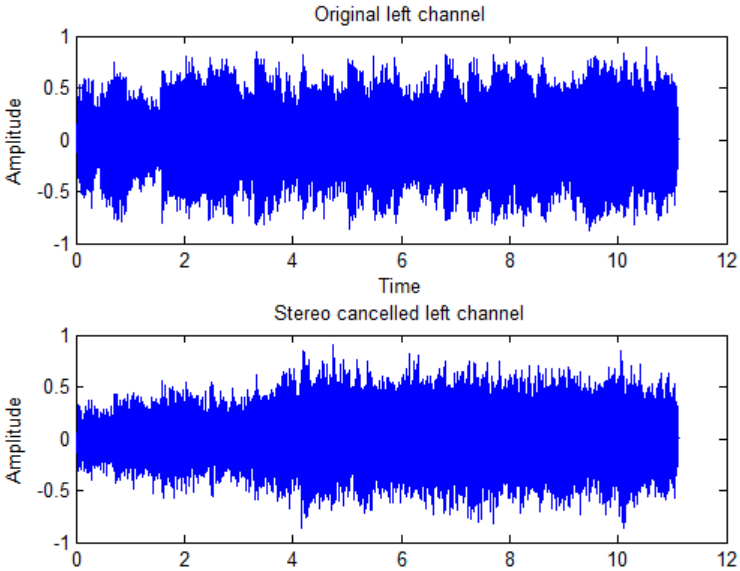

Figure 2 - The left channel, before and after stereo cancellation

Stereo Cancellation

Stereo cancellation, as the name implies, **requires** using a stereo sample, and involves subtracting common frequencies between the two channels (figure 2). This works most of the time, because the lead singer's voice is mostly center-panned, and therefore has common frequencies in both channels.

However, since one channel is subtracted from the other, the result of stereo cancellation is a mono track. Another unwelcome result is the lowering in volume of other sounds that are common between both channels, such as drums or bass. Still, the results of this technique are better than that of simple bandstop cancellation.

Audio Blind Source Separation

This technique consists of *"extracting from an input audio signal a set of audio signals whose mix is perceived similarly to the original audio signal"*. In our case, we focused on extracting the vocals track from the mix consisting of the rest of the instruments.

In order to extract the vocals from the mix, the following steps were followed:

1) Selection of song

A stereo mix has to be chosen, preferably without any reverberation. If the selected mix has reverberation, the procedure is complicated considerably because the mono tracks overlap with other tracks. Good candidate mixes for this technique are old stereo songs, where complex post-recording audio effects are not as frequent.

The song that we chose is "Let it be" by the Beatles. In figure 3, we observe both channels of our song. This clearly depicts that the left channel (l) does not contain the same information as the right channel (r). Hence, the song was recorded in stereo mode.

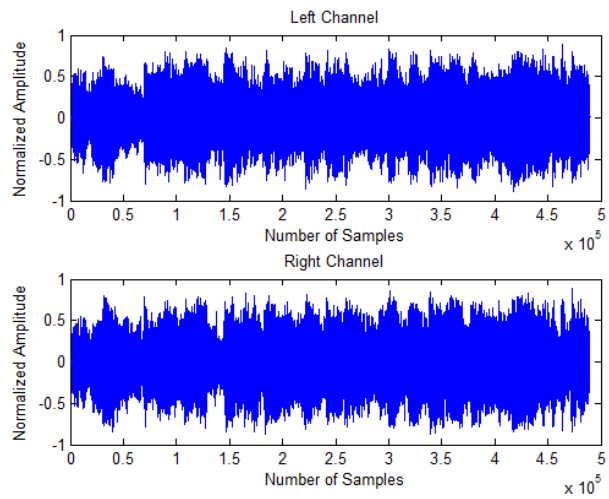

Figure 3 - The chosen stereo clip

2) Short Time Fourier Transform (STFT) and spectrogram

After the song was chosen and the channels separated, a spectrogram was generated for each one of the channels. In order to get the spectrogram, we used the STFT.

$$L = \text{STFT}(l) \quad (1)$$

$$R = \text{STFT}(r) \quad (2)$$

Where,

$L = left\ channel\ \ STFT$

$R = right\ channel\ \ STFT$

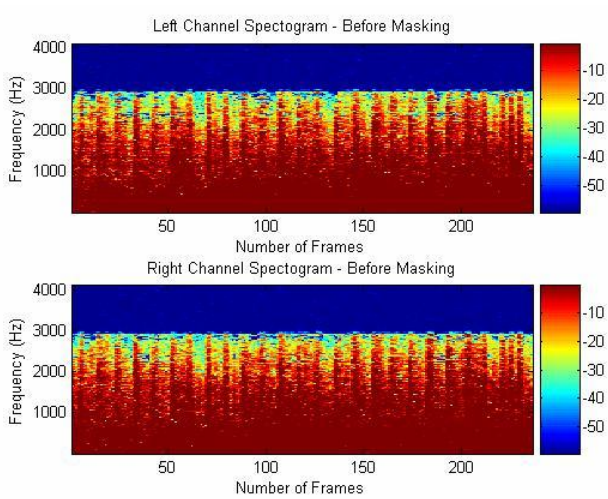

Figure 4 - Spectrogram of left and right channel

The STFT has three important factors to consider: The first factor is the number of DFT points (N) that will be generated per frame; the second factor is the offset among frames; and third factor is the type of window that each frame will use.

The first two factors will create a large number of samples in our spectrogram. This large number of samples will let us extract areas were the voice is concentrated with more accuracy. Also, the window in each frame is of extreme importance because the offset among frames is really low. So, if the data is not well confined in each frame, we will have noise coming from other frames due to overlap.

In our project, the number of DFT points for our STFT (N) was set to 8192 while the offset among frames was set to 2048 (N/4). This yields, in the spectrogram, an array of $4097 * \left(\frac{N}{2} + 1\right) * 236$ DFT points (for our specific clip). Moreover, the window used for the frames is the Hamming window, although a Blackman window will also yield the same or better results due to a higher attenuation at the stopband. Figure 4 shows the spectrograms for both the left and the right channels.

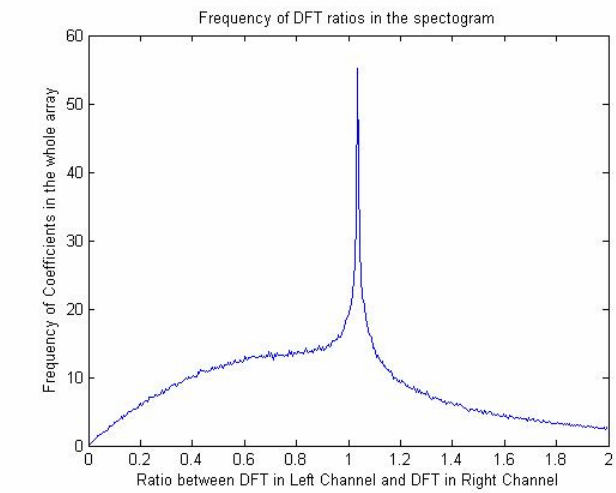

Figure 5 - Frequency of channel ratio

3) Track Identification

Once we have both spectrograms, we proceed to recognize similarities between both channels. We do this with the purpose of getting one or more of the tracks that the channels share. In our case, the technique that we used is to divide the magnitude of each DFT coefficient in the left channel by its counterpart in the right channel spectrogram. This division gives us a channel ratio (CR) between both channels. This is expressed as follows:

$$\mathrm{CR} = \frac{|L|}{|R|} \tag{3}$$

When you divide a coefficient from the left channel (that represents a single track) by a coefficient from the right channel (that also represents the same single track), the result will be a constant value no matter where we are located in the spectrogram [1]. However, if you divide coefficients that represent two or more tracks, your result will not be constant throughout the spectrogram anymore. In figure 5, we can see the frequency for each ratio in the spectrogram. At a ratio of 1, we found a peak. This peak represents a mono track that was inserted evenly on both channels (if the track is different to one, this means that the mono track in one of the channels was attenuated or amplified).

4) Time Frequency Mask – Binary Method

Once the coefficients that represent mono tracks in the channels are identified, we proceed to substitute them by zeros. This method is usually known as binary masking because the coefficients are multiplied by either one or by zero (in other, more advanced, techniques, the coefficients can be weighted). This can be seen mathematically as follows:

$$M = \begin{cases} 0 & if \quad a < CR < b \\ 1 & otherwise \end{cases} \tag{4}$$

where

$$M = binary\ mask$$

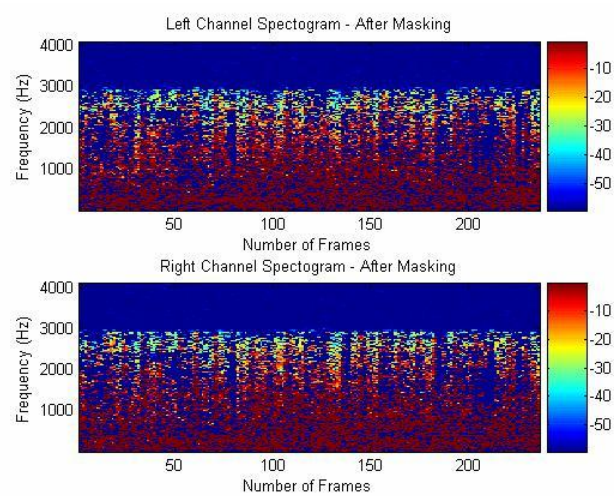

Figure 6 - Spectrogram with binary mask

For the Beatles clip that we chose, the values for expression 4 with a = 0.65 and b = 1.35.

In figure 6, it can be observed that some parts of the spectrograms have been reduced to zero (blue). Now, it becomes clear why a greater array of coefficients would yield better results. This is mainly because we can be more selective in the areas we want to reduce.

5) Time-signal recovery – Inverse STFT (ISTFT)

After the binary mask has been applied, the signal has to be transformed back to the time domain. In order to do this, the ISTFT is used on each one of the channels. The following expressions show this:

$$\bar{l} = istft(\bar{L}) \quad (5)$$

$$r = istft(\bar{R}) \quad (6)$$

where

$\bar{L} = Left\ channel\ masked\ signal\ before\ istft$

$\bar{l} = RECOVERED\ left\ channel\ signal$

$\bar{R} = Right\ channel\ masked\ signal\ before\ istft$

$\bar{r} = RECOVERED\ right\ channel\ signal$

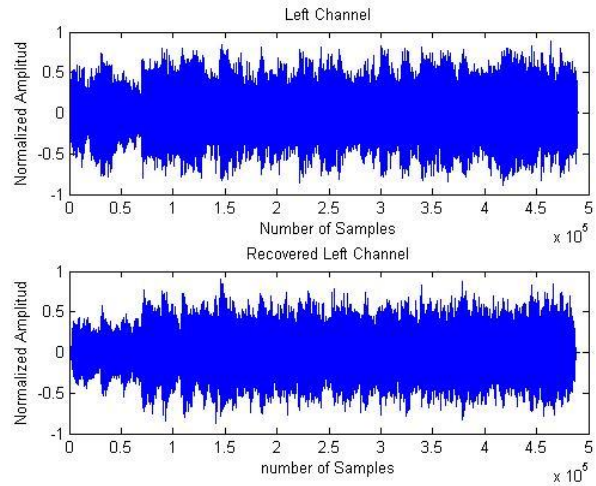

Figure 7 - Left channel with its recovered counterpart

In figures 7 and 8, the original signal and the recovered signal are compared. It can be seen that each one of the signals has been altered from the original one. When the two recovered signals are mixed again, we get a stereo clip, but this time without vocals. There is a noticeable difference between this technique and the stereo cancellation technique. In this technique we end up with a stereo output, while in stereo cancellation we can only produce mono outputs.

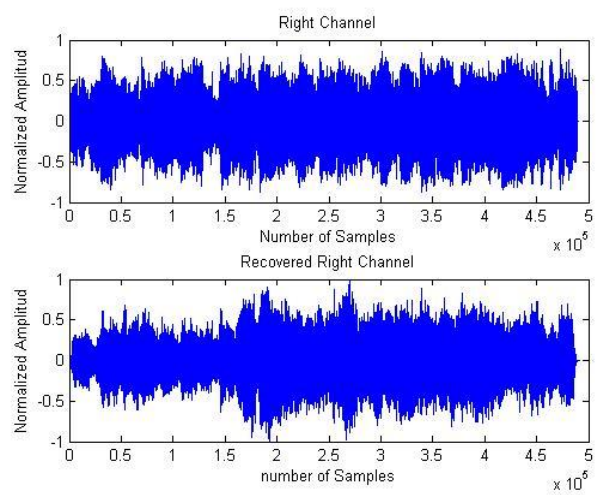

Figure 8 - Right channel with its recovered counterpart

The five steps explained above are one way to remove the vocals from the song. However, this technique is not limited to extracting the vocals from a song; for instance, we could extract the instruments and leave the vocals in the song. Therefore, this technique permits a greater flexibility compared to the other techniques explained in this paper.

Nonetheless, the audio source separation technique used in this project is just one of many audio source separation approaches. This is mainly because different mixes of instruments and new sound effects intermingles frequencies in more complex ways. Due to this added complexity, a binary mask approach will not be enough to separate the sources from the song.

Conclusion

In this report, we learned several approaches to separate vocals from a commercial song. Firstly, bandstop filtering was discussed - this approach consists in applying a filter at the vocals frequency; however, this technique not only filtered out the vocals but also the other instruments in the same frequency range. Second, a slightly more advanced technique, stereo cancellation - this technique gives good results for stereo songs but the results are limited to the fact that the lead singer's voice has to be recorded with the same pan in the left and right channel (besides, the output from this technique is a mono output). Finally, audio blind source separation was discussed - this technique permits the extraction of sounds at certain frequencies and time intervals. This gives us a greater selectivity in discerning vocals.

Due to the huge diversity of sounds and mixes in modern commercial audio mastering, there exists no universal approach to vocals removal (it is still an ongoing research field). However, for some commercial tracks, the approaches presented here will yield a satisfying result.

References

[1] MarC Vinyes, Jordi Bonada, Alex Loscos. "*Demixing Commercial Music Productions via Human-Assisted Time-Frequency Masking*" Presented at the Audio Engineering Society, Paris, France, 2006.

Acknowledgements

The authors would like to thank Prof. Dan Ellis for his unending support and encouragement, and also for providing us with an opportunity to learn by design. Digital signal processing is a fascinating field, and is made all the better by the guidance of a good mentor and friend. The authors would also like to thank The Beatles for creating a song that is now so deeply imprinted on our minds that we unconsciously hum it from time to time, with mostly embarrassing results.

Appendix A – Matlab Code

```
% % ELEN E4810 - Digital Signal Processing
% % Course Project - Removing vocals from commercial tracks
% % Instructor: Prof. Dan Ellis (dpwe@columbia.edu)
% % Authors: Jaime Peretzman (jp2642), Shrivathsa Bhargav (sb2784)
% % Columbia University, Fall 2007

display(' ');
display(' ');
display('-----------------------------------------------------------');
display('          Removing vocals from commercial tracks');
display('-----------------------------------------------------------');
display(' ');
display('Welcome to the ELEN E4810 - DSP Project');
display('By Jaime Peretzman and Shrivathsa Bhargav');
display('Columbia University, Fall 2007');
display(' ');

display('Reading in wave file');
%Clip from The Beatles - Let It Be
[sam,sr]=wavread('let_it_be.wav');
display('>>PRESS ENTER to hear original clip');
pause;
soundsc(sam,sr); %Original Stereo Clip
pause(10);
info_channels = size(sam);

display('Splitting stereo channels');
%Split the channels
n = 1:info_channels(1,1);
sam_l(n) = sam(n,1); %Left Channel
sam_r(n) = sam(n,2); %Right Channel

display('>>PRESS ENTER to plot the two channels');
pause;
figure(1);
subplot(2,1,1);
plot(sam_l);
title('Left Channel');
xlabel ('Number of Samples');
ylabel ('Normalized Amplitude');
axis ([0 500000 -1 1])
subplot(2,1,2);
plot(sam_r);
title('Right Channel');
xlabel ('Number of Samples');
ylabel ('Normalized Amplitude');
axis ([0 500000 -1 1])

display(' ');
display('Bandstop filtering');
%BANDSTOP FILTERING
%Find a good filter window
wh = transpose(hanning(length(sam_l)));

%Cutoff frequency, bandstop left
f_cl = 300;
w_cl = f_cl/(sr*0.5);
```

```
%Cutoff frequency, bandstop right
f_cr = 2500;
w_cr = f_cr/(sr*0.5);

display('>>PRESS ENTER to see the bandstop filter');
pause;
display('Plotting bandstop filter characteristics');
%Create the bandstop filter of order 1000 (!)
b_stop = fir1(1000, [w_cl, w_cr], 'stop');
freqz(b_stop);
title('Filter characteristics');

display('>>PRESS ENTER to see and hear the bandstop-filtered sample');
pause;
display('Plotting filtered sample');
filtered1 = conv(sam_l,b_stop);
soundsc(filtered1,sr);
freqz(filtered1);
pause(10);

display(' ');
display('Stereo cancellation');
sam_no_vox = sam_l - sam_r;
sam_l_fft = fft(sam_l);
sam_r_fft = fft(sam_r);
sam_no_vox_fft = sam_l_fft - sam_r_fft;
sam_no_vox = ifft(sam_no_vox_fft);

display('>>PRESS ENTER to see and hear the stereo-cancelled sample');
pause;
display('Plotting stereo cancelled sample');
soundsc(sam_no_vox,sr);
subplot(2,1,1);
plot(n./sr,sam_l);
title('Original left channel');
ylabel('Amplitude')
xlabel('Time')
subplot(2,1,2);
plot(n./sr,sam_no_vox);
title('Stereo cancelled left channel');
ylabel('Amplitude')
xlabel('Time')
pause(10);


display(' ');
display('Audio Blind Source Separation');
% fft_points is the number of samples per frame. This has to be large in
% order to create enough points to mask in our spectogram
fft_points = 8192;
%This is the size of the window, in our case we use a hamming window, but
%it would yield better results a blackman window
window_size = 8192;
%This is the offset between frames. This has to be small enough to get
%more points in our spectogram but not cause noise in our final result.
frame_size = fft_points./4;

display('Performing Short-Time Fourier Transform (STFT)');
%%%STFT OF THE LEFT CHANNEL%%%
```

```
sam_l_stft = stft(sam_l, fft_points, window_size, frame_size);
%%%STFT OF THE RIGHT CHANNEL%%%
sam_r_stft = stft(sam_r, fft_points, window_size, frame_size);
%%%size(sam_l_stft)=(rows=4097 columns=236) In our diagram

%Comparison between left and right channel through
%DFT(left channel)/DFT(right channel)
div_dft = sam_l_stft./sam_r_stft;
mod_div_dft = (abs(div_dft)); %The modulus is gotten in order to get rid of
%complex numbers and just have magnitudes to compare.


display('>>PRESS ENTER to see spectrogram before masking vocals');
pause;
display('Plotting spectrogram before masking vocals, and')
%Spectogram before masking the vocals
% figure(2)
figure(1);
subplot(2,1,1);
imagesc(20*log10(abs(sam_l_stft))); %plots the spectogram
title('Left Channel Spectogram - Before Masking');
ylabel('Frequency (Hz)')
xlabel('Number of Frames')
axis xy;
caxis([-60 0]);
colorbar; % provides decode of colors

subplot(2,1,2);
imagesc(20*log10(abs(sam_r_stft)));
title('Right Channel Spectogram - Before Masking');
ylabel('Frequency (Hz)')
xlabel('Number of Frames')
axis xy;
caxis([-60 0]);
colorbar;

display('Identifying masking coefficients, this could take a minute...');
A=0;
a=0;
b=a+0.005;
i=1;
%This while loop finds the number of DFT(left)/DFT(right) coefficients
%inbetween fixed ranges. (e.g. between 0.995 and 1.000)
while (b<2)
    A(i) = mean(sum(mod_div_dft>a & mod_div_dft<b)) ;
    i = i + 1;
    a=a+0.005;
    b=a+0.005;
end

display('>>PRESS ENTER to see frequency of DFT ratios');
pause;
%In this graph, it can be seen that the peak happens around a ratio of 1.
%Extracting the coefficients around 1 will let us extract any mono track
%this peak represents.
% figure(3)
a=0:0.005:1.995;
figure(2);
plot(a,A)
```

```
title('Frequency of DFT ratios in the spectogram');
xlabel('Ratio between DFT in Left Channel and DFT in Right Channel')
ylabel('Frequency of Coefficients in the whole array')

display('Performing Binary Time Frequency Mask');
%This is a binary Time Frequency Mask (BTFM). This loop will search
%throughout the whole spectogram for DFT(left)/DFT(right) inbetween 0.65
%and 1.35. In case it finds a value, it will be set the DFT coefficients of
% both left and right channel to zero, otherwise the coefficients are not
% changed at all
for k=1:236;
    for m=1:4097;
        if (mod_div_dft(m,k)>0.65 & mod_div_dft(m,k)<1.35);
            sam_l_stft (m,k)=0;
            sam_r_stft (m,k)=0;
            m=m+1;
        else
            m=m+1;
        end
    end
    k=k+1;
end

display('>>PRESS ENTER to see spectrogram after masking vocals');
pause;
%Spectogram with mask applied
% figure(4)
figure(2);
subplot(2,1,1);
imagesc(20*log10(abs(sam_l_stft))); %plots the spectogram
title('Left Channel Spectogram - After Masking');
ylabel('Frequency (Hz)')
xlabel('Number of Frames')
axis xy;
caxis([-60 0]);
colorbar; % provides decode of colors

subplot(2,1,2);
imagesc(20*log10(abs(sam_r_stft)));
title('Right Channel Spectogram - After Masking');
ylabel('Frequency (Hz)')
xlabel('Number of Frames')
axis xy;
caxis([-60 0]);
colorbar;

display('Performing inverse-STFT');
%Inverse STFT
sam_l_istft = istft(sam_l_stft, fft_points, window_size, frame_size);
sam_r_istft = istft(sam_r_stft, fft_points, window_size, frame_size);

display('>>PRESS ENTER to see original channels vs recovered channels');
pause;
%Original Channels vs Recovered Channels
% figure(5);
figure(1);
subplot(2,1,1);
plot(sam_l);
```

```
title('Left Channel');
xlabel ('Number of Samples');
ylabel ('Normalized Amplitude');
axis ([0 500000 -1 1])
subplot(2,1,2);
plot(sam_l_istft);
title('Recovered Left Channel');
xlabel ('number of Samples');
ylabel ('Normalized Amplitude');
axis ([0 500000 -1 1]);

figure(2);
subplot(2,1,1);
plot(sam_r);
title('Right Channel');
xlabel ('Number of Samples');
ylabel ('Normalized Amplitude');
axis ([0 500000 -1 1])
subplot(2,1,2);
plot(sam_r_istft);
title('Recovered Right Channel');
xlabel ('number of Samples');
ylabel ('Normalized Amplitude');
axis ([0 500000 -1 1])

display('>>PRESS ENTER to hear recovered sample');
pause;
%display('Playing original clip');
%soundsc(sam,sr);
display('Playing recovered sample');
sam_recovered=[sam_l_istft;sam_r_istft];
soundsc(sam_recovered.',sr);

display(' ');
display('Done! Thanks for listening. Goodbye!');
display(' ');
```