Stereo to 5.1 Surround Audio-Upmix using Time-Frequency domain Techniques

Ajin Tom
Music Technology Department
McGill University - Montreal, CA
ajin.tom@mail.mcgill.ca

ABSTRACT

This project is an implementation of a Time-Frequency Analysis-Synthesis model for generating multichannel audio from stereo recordings. The techniques I have used for up-mixing are based on comparing the Short-Time Fourier Transforms of left and right stereo signals to derive interesting statistical parameters for spectral modifications. Generating the surround channels involves extracting and feeding the ambience components from the stereo mix by computing instantaneous inter-channel coherence measure. A similarity measure is computed and tracked to identify and unmix various sources based on their panning coefficients, using which a faithful frontal sound image is generated. This report consists of the implementation and modifications made to the algorithm proposed by Avendano et al[1][2] for frequency domain based up-mixing techniques. This report also illustrates interesting graphs and test descriptions of various signals used to test this two-to-five channel up mix system thus giving a more intuitive and clearer picture of the algorithm's working.

Keywords

stereo-to-multichannel up-mixing, ambience extraction, un-mixing panned sources, spatial audio, time-frequency representation

1. MOTIVATION

Surround multi-channel home theater speaker systems are getting popular nowadays, but the **availability of multichannel audio recordings are very limited**. Most of us have only stereo tracks in our personal music libraries. Though recent movie soundtracks and very few music recordings make use of the 5.1 surround, faithful reproduction of the more common stereo sounds on these multi-channel systems pose a fundamental problem. A major problem is to deal with is the fact that stereo audio recordings are carried out with a mindset that the playback is to be done on a stereo system; this poses a big question of what signals should be sent to the additional channels (surround and center channels). Thus, there is no firm strategy to distribute the signals; so it is usually the call of the users based on their individual preferences or artistic design. I found the Time-Frequency domain analysis approach useful for extracting interesting features to create an immersive feel and producing a stable center channel for off-axis listening

2. INTRODUCTION

In the context of stereo recordings, one approach is to develop a stable center sound image in the front(left, center and right) speakers and also create an enhanced immersive listening experience by simulating reverberant/reflected components on the surround speakers and further maybe give users control over the size/shape of the room.

This project uses an up-mix strategy inspired by the following two main concepts:

1) **Direct/ambient sounds extraction:** The different sources/ instruments in the mix are panned among the front channels to create a dominant frontal sound image like in any stereo mix; the ambience signals are distributed among all channels. This mix creates an impression that the listener is in the center/sweet spot of an acoustic space with all the primary sounds approaching from the front (front left, center and right speakers) while perceiving ambience/reflections of the room/ hall from all around the listener (rear speakers).

2) **Source un-mixing and synthesis of panned sources :** This concept is used to develop an enhanced center/frontal image of the sound sources in a mix. This method identifies the various amplitude-panned sources in the mix and assigns a panning index for each frequency component with the evolution of time. These panning indices are tracked to re-synthesize the frontal sound image by distributing the stereo mix among the multichannel speaker array thus enabling placement of various sources by choosing appropriate panning index values. For example, most of the center panned sources will now be heard from the center speaker while the extremely panned sources are pushed towards the left and right pairs.

The basic idea extensively used in this project is comparing the STFT's (Short-Time Fourier Transforms) of the left and right stereo signals in order to extract different features and identify different components in the mix. An inter-channel coherence measure is computed to identify the ambient components, and an inter-channel similarity measure is used to identify the panning coefficients corresponding the various individual instruments in the mix. Following the identification of these components, a non-linear mapping function is used to weigh and extract the interesting time-frequency regions. After all these spectral modifications, the five signals to be played-back on the multichannel output are re-synthesized using inverse STFT via the OLA (overlap-and-add) method. The other conventional methods for up-mixing include PSD (Passive Surround Decoding), LMS- and PCA-based approaches, adaptive- and constant- power panning and Speaker-Placement correction Amplitude Panning (SPCAP) method.

In this report I have introduced the state-of-the art signal processing techniques involved in ambience extraction and un-mixing the sources. In parallel to the derivation and presentation of the key equations, I have discussed my strategy for implementing the corresponding concepts. At every stage I have also illustrated test signals and evolution of interesting features while tweaking various parameters in the algorithm. The implementation was carried out on Python 2.7 using libraries like Numpy for array handling, SciPy for wav files reading/writing and Matplotlib for plotting the waveforms.

The goal of this project is to be able to use Time-Frequency domain based analysis-modifications-reconstruction approach to implement the above mentioned concepts and test the system by plotting and tracking key features and listening to the produced output on a 5.1 Surround System.

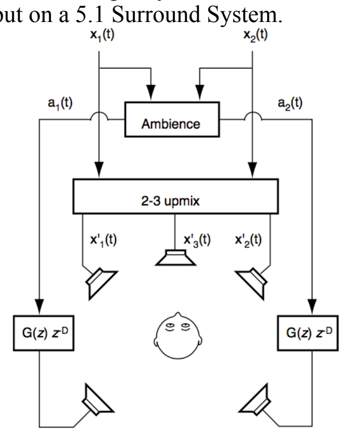


Fig.1 Block diagram of the system. *Original Stereo: $x_1(t)$ and $x_2(t)$ Extracted ambience: a(t), Up-mixed Left, Center, Right: x'(t)*

3. SIGNAL MODEL

Studio recordings usually consists of audio tracks of each instrument recorded separately and later mixed into a stereo signal. Effects like reverberation makes listeners perceive that they are in a certain acoustic space by having weakly correlated left and right impulse responses processed with the stereo mix.

A signal model for a stereo signal can be defined as:

$$x_i(t) = \underbrace{\sum_{j=1}^{N} s_j(t) * d_{ij}(t)}_{\text{Primary signals}} + \underbrace{\sum_{j=1}^{N} s_j(t) * r_{ij}(t)}_{\text{Ambience signals}} + n_i(t), \quad (1)$$

$s_j(t)$, j = 1 to N represents N sources convolved with room impulse(artificial reverberation) responses $h_{ij}(t)$ to generate the left (i = 1) and right (i = 2) stereo channels respectively.
$h_{ij}(t) = d_{ij}(t) + r_{ij}(t)$ where d and r and direct path and reverberation components in the signal and $n_i(t)$ is noise

From this kind of signal model, we can notice that sources panned to center $d_{10}(t) = d_{11}(t)$ can be easily separated from the uncorrelated ambience content. The basic idea used is based on the fact that binaural hearing and localization involve computation of the cross-correlation between the left and right channel signals. The higher processes use across-frequency coherence information and other inter and intra-channel features to determine position, distance and directionality of the sources.

4. STFT Framework

This section includes explanation to the implementation of the Short-Time-Fourier-Transform Analysis-Modification-Synthesis stages along with the state-of-the-art signal processing techniques in this Time-Frequency domain based approach:

4.1. Analysis

scipy.io.wavfile.read() function reads a wave format stereo file and stores the sampling frequency and the signal's left and right channels are stored in 2 arrays: s1 and s2.

Following values are good to start with for the STFT parameters:

```
M = 2048                         #frame size
N = 2048                         #FFT size
H = int(M/4)                     #hop size
maxFreq = 20000.0
W = 'hamming'                    #analysis window
w = get_window(W, M)
```

These parameters could be tweaked based on the signal since we know that time-frequency resolution is a trade-off in STFT.
In the analysis stage, a hamming window of size M is run across the time domain signal with a hop size such that there is a 75% overlap. M/2 zeros are appended to the beginning and end of the signal so that the first window is centered at sample 0 of s1 and s2.

For each windowed time-frame, the sound pointer is initialized to the center of the frame and that pointer keeps hopping by H. Each frame is passed as arguments to the DFT analysis function in which a zero-phase windowing is carried out to obtain a real-valued FFT:

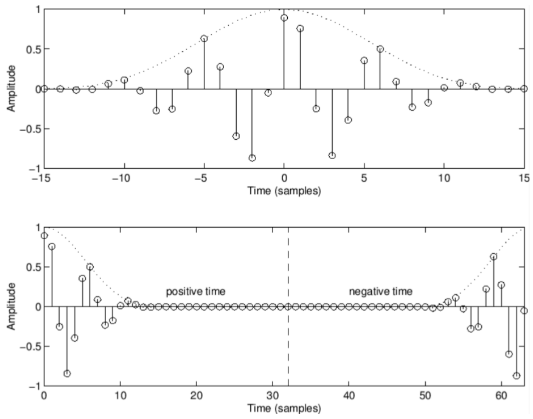


Fig.2 Zero-phase windowing and zero-padding

From this FFT Buffer, the absolute values of the samples in the first window is calculated followed by phase unwrapping to obtain and return the mX1, mX2 - magnitude spectrum and phase spectrum pX1, pX2 of the left and right channel time-frames. One implementation tip here is to set extremely small values from these arrays to 0.

4.2. Spectral modifications

This section includes implementation of the most important stage of the STFT framework where the computation of inter-channel short-time coherence(for ambience index) and similarity (for panning index) is carried out. I have briefly presented the derivation and implementation for ambience extraction, source un-mixing, smoothening techniques and other spectral modifications.

4.2.1. Ambience Extraction

The ambience extraction algorithm is similar to the structure of equation (1) but the following functions give more robust control of several parameters to measure correlation:

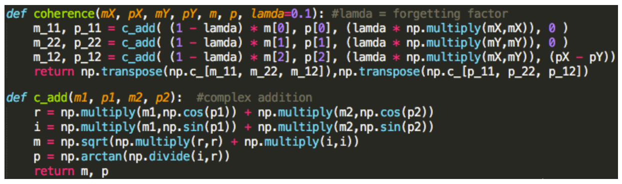
$$\Phi_{ij}(k) = \mathrm{E}\{S_i(m,k)S_j^*(m,k)\}, \quad (2)$$

$$\Phi_{ij}(m,k) = \lambda\Phi_{ij}(m-1,k) + (1-\lambda)S_i(m,k)S_j^*(m,k). \quad (3)$$

$$\Phi(m,k) = \frac{\Phi_{12}(m,k)}{[\Phi_{11}(m,k)\Phi_{22}(m,k)]^{\frac{1}{2}}}. \quad (4)$$

The above equations gives a measure of inter-channel short-time coherence between the frequencies points of the FFT frames of the left and right channels S_i and Sj, in the following code snippets represented as mX1 and mX2:
Equation (4) is the coherence function which is real and will have values close to one in time-frequency regions where the primary signal is dominant, and will be close to zero in regions dominated by the reverberation tails and ambience (surrounding noise).

Equation (3) is implemented as follows:



```
def coherence(mX, pX, mY, pY, m, p, lamda=0.1): #lamda = forgetting factor
    m_11, p_11 = c_add( (1 - lamda) * m[0], p[0], (lamda * np.multiply(mX,mX)), 0 )
    m_22, p_22 = c_add( (1 - lamda) * m[1], p[1], (lamda * np.multiply(mY,mY)), 0 )
    m_12, p_12 = c_add( (1 - lamda) * m[2], p[2], (lamda * np.multiply(mX,mY)), (pX - pY))
    return np.transpose(np.c_[m_11, m_22, m_12]),np.transpose(np.c_[p_11, p_22, p_12])

def c_add(m1, p1, m2, p2):  #complex addition
    r = np.multiply(m1,np.cos(p1)) + np.multiply(m2,np.cos(p2))
    i = np.multiply(m1,np.sin(p1)) + np.multiply(m2,np.sin(p2))
    m = np.sqrt(np.multiply(r,r) + np.multiply(i,i))
    p = np.arctan(np.divide(i,r))
    return m, p
```

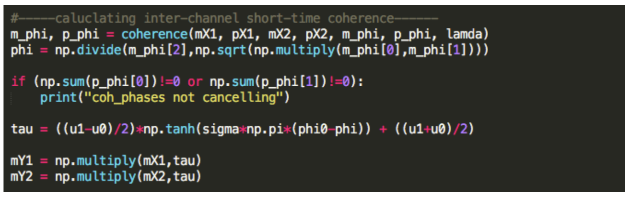
The ambience transform is given as:

$$A_i(m,k) = S_i(m,k)\Gamma[\Phi(m,k)] \quad (5)$$

Equation (5) forms a general form to weigh the channel short-time transform with a non-linear function of the short-time coherence, given as follows:

$$\Gamma[\Phi] = \left(\frac{\mu_1 - \mu_0}{2}\right)\tanh\{\sigma\pi(\Phi_0 - \Phi)\} + \left(\frac{\mu_1 + \mu_0}{2}\right) \quad (6)$$

The implementation of (4), (5) and (6) requires the script to store the magnitude and phase coherence measure(m_phi, p_phi) of the previous STFT window as well:



```
#-----caluclating inter-channel short-time coherence------
m_phi, p_phi = coherence(mX1, pX1, mX2, pX2, m_phi, p_phi, lamda)
phi = np.divide(m_phi[2],np.sqrt(np.multiply(m_phi[0],m_phi[1])))

if (np.sum(p_phi[0])!=0 or np.sum(p_phi[1])!=0):
    print("coh_phases not cancelling")

tau = ((u1-u0)/2)*np.tanh(sigma*np.pi*(phi0-phi)) + ((u1+u0)/2)

mY1 = np.multiply(mX1,tau)
mY2 = np.multiply(mX2,tau)
```

The effects of the parameters in (6) are as follows in Fig.3:

```
u1    = 1.0     # u0 and u1 determine range of output
u0    = 0.001   # determines the floor of the function
sigma = 8       # controls the slope of the function
phi0  = 0.5     # sets the threshold
lamda = 0.1     # forgetting factor
```

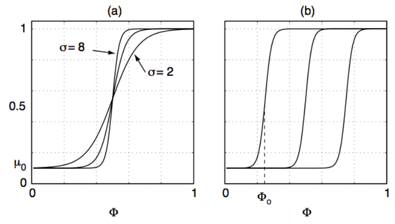


Fig.3 Mapping function for ambience index = 1 - phi with u0 = 0.1, u1 = 1.0
a) As a function of sigma, phi0 = 0.5
b) As a function of phi0, sigma = 4

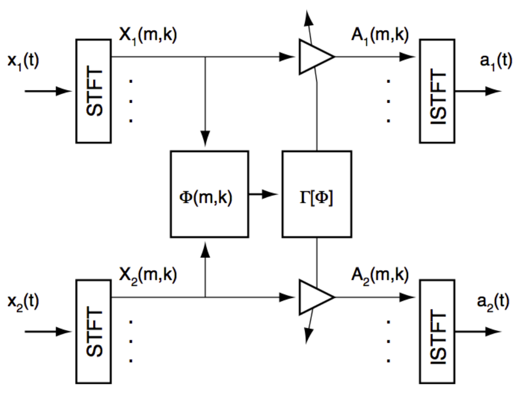


Fig.4 Block diagram of ambience extraction system

4.2.2. Identification of amplitude-panned sources

The basic idea in this technique is to compute the short-time panning index from the left and right spectrum to identify various sources in the stereo mix based on their panning coefficients (alpha). The evaluation of this term is calculated from the similarity function which is defined as follows:

$$\psi(m,k) = 2\frac{|\psi_{12}(m,k)|}{[\psi_{11}(m,k) + \psi_{22}(m,k)]} \qquad (7)$$

This equation is basically computed by setting forgetting factor lamda to 1 in the short-time coherent equation and scaled by 2 so that the output is scaled to (0, 1) bounds. This function tracks the relative levels between left and right frequency points for every spectral frame. The relation between panning coefficients and the similarity function is given as follows in (7') and plotted in Fig.5.

$$\Delta(m,k) = \psi_1(m,k) - \psi_2(m,k) \qquad (7')$$

The similarity function poses an ambiguity due to its symmetric nature that it gives the same similarity value for a source with panning coefficients 0.2 and 0.8. This is solved using the following computations to identify left vs right panned sources:

$$\Delta(m,k) = \psi_1(m,k) - \psi_2(m,k), \qquad (8)$$

$$\widehat{\Delta}(m,k) = \begin{cases} 1 & \text{if} \quad \Delta(m,k) > 0 \\ 0 & \text{if} \quad \Delta(m,k) = 0 \\ -1 & \text{if} \quad \Delta(m,k) < 0 \end{cases} \qquad (9)$$

$$\Psi(m,k) = [1 - \psi(m,k)]\,\widehat{\Delta}(m,k), \qquad (10)$$

$$\psi_i(m,k) = \frac{|\psi_{ij}(m,k)|}{\psi_{ii}(m,k)} \qquad (11)$$

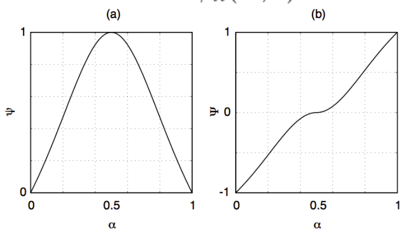
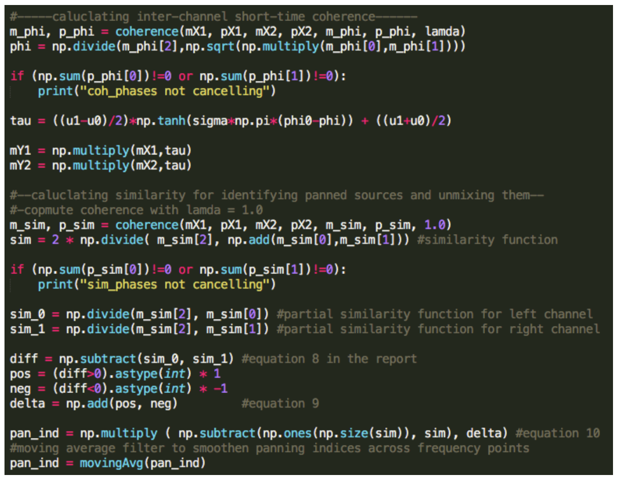


Fig.5 a) Similarity function b) Panning index

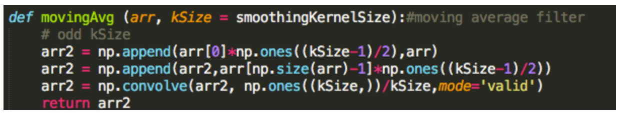
The above figure illustrates how the ambiguity discussed above is sorted. Using this domain, it is now possible to identify prominent sources in the mix by their panning coefficients(alpha)/positions in the stereo image. It is also interesting to note that choosing a custom alpha for every frequency point allows source re-panning depending on the desired artistic design of the mix. Irrespective of the original position, we can choose to hear a certain instrument / sound from a chosen position in the multi-channel speaker array.



```
#-----caluclating inter-channel short-time coherence------
m_phi, p_phi = coherence(mX1, pX1, mX2, pX2, m_phi, p_phi, lamda)
phi = np.divide(m_phi[2],np.sqrt(np.multiply(m_phi[0],m_phi[1])))

if (np.sum(p_phi[0])!=0 or np.sum(p_phi[1])!=0):
    print("coh_phases not cancelling")

tau = ((u1-u0)/2)*np.tanh(sigma*np.pi*(phi0-phi)) + ((u1+u0)/2)

mY1 = np.multiply(mX1,tau)
mY2 = np.multiply(mX2,tau)

#--caluclating similarity for identifying panned sources and unmixing them--
#-copmute coherence with lamda = 1.0
m_sim, p_sim = coherence(mX1, pX1, mX2, pX2, m_sim, p_sim, 1.0)
sim = 2 * np.divide( m_sim[2], np.add(m_sim[0],m_sim[1])) #similarity function

if (np.sum(p_sim[0])!=0 or np.sum(p_sim[1])!=0):
    print("sim_phases not cancelling")

sim_0 = np.divide(m_sim[2], m_sim[0]) #partial similarity function for left channel
sim_1 = np.divide(m_sim[2], m_sim[1]) #partial similarity function for right channel

diff = np.subtract(sim_0, sim_1) #equation 8 in the report
pos = (diff>0).astype(int) * 1
neg = (diff<0).astype(int) * -1
delta = np.add(pos, neg)         #equation 9

pan_ind = np.multiply ( np.subtract(np.ones(np.size(sim)), sim), delta) #equation 10
#moving average filter to smoothen panning indices across frequency points
pan_ind = movingAvg(pan_ind)
```

The above code snippet illustrates the implementation of equations (7), (8), (9), (10), (11).

One modification I carried out in the algorithm is the smoothening of the panning indices across the frequency points within a spectral window and this was implemented using a moving average filter implemented using convolution, as follows:



```
def movingAvg (arr, kSize = smoothingKernelSize):#moving average filter
    # odd kSize
    arr2 = np.append(arr[0]*np.ones((kSize-1)/2),arr)
    arr2 = np.append(arr2,arr[np.size(arr)-1]*np.ones((kSize-1)/2))
    arr2 = np.convolve(arr2, np.ones((kSize,))/kSize,mode='valid')
    return arr2
```

4.2.3. Source Unmix and Synthesis

To extract a center panned signal, we would find all time-frequency regions for which the panning index is zero and then synthesize a time domain function using just these components. Similarly, this procedure is applied to signals panned to other locations by selecting different panning index values.

In this kind of assignment of highly varying panning indices over the frequency points within a spectral window, there could be artifacts due to abrupt modifications of the STFT. To avoid this and also to account for possible time-frequency overlap between signal components, a narrow window centered around the panning index value corresponding to the desired panning coefficient is generated.

At this stage there is a significant trade-off between separation and distortion while choosing the window width. A wider window will be safer as it produces smoother transitions but will also allow signal components panned zero to pass, which is not ideal. This un-mixing function is defined as follows in the form of a Gaussian window function:

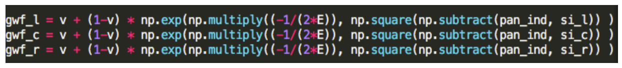
$$\Theta[\Psi] = \nu + (1-\nu)e^{-\frac{1}{2\xi}(\Psi-\Psi_0)^2} \qquad (12)$$

To unmix the desired source, the sum of left and right spectral windows are multiplied with the above gaussian window.

$$S_u(m,k) = \Theta[\Psi(m,k)](X_1(m,k) + X_2(m,k)) \qquad (13)$$

```
#defining parameters for unmixing
v = 0.001              #low floor
E = 0.125              #gaussian window width
si_l = -0.5            #panning index for left
si_c = 0.0             #panning index for center
si_r = 0.5             #panning index for right
```

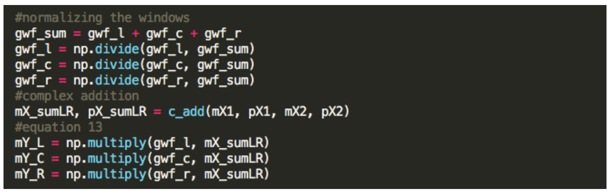
The 3 STFTs are calculated as a function of panning index values thus generating 3 gaussian windows centered around si_l,c,r. These values are initialized to -1, 0 and 1 for left, center and right respectively. The contribution by those panned frequencies outside the window is determined by v, the low floor value.



```
gwf_l = v + (1-v) * np.exp(np.multiply((-1/(2*E)), np.square(np.subtract(pan_ind, si_l)) )
gwf_c = v + (1-v) * np.exp(np.multiply((-1/(2*E)), np.square(np.subtract(pan_ind, si_c)) )
gwf_r = v + (1-v) * np.exp(np.multiply((-1/(2*E)), np.square(np.subtract(pan_ind, si_r)) )
```

Another interesting strategy I came up with to get better control over this un-mixing algorithm was to use individual window widths and low floor values for the three spectrums using a backward model to optimize the parameters.

This step was followed by normalizing the 3 windows before multiplying them with the left and right spectrums. This way the artifacts weren't present anymore in the final multi-channel mix.
gwf - Gaussian Window Function , l,c,r - left, center, right

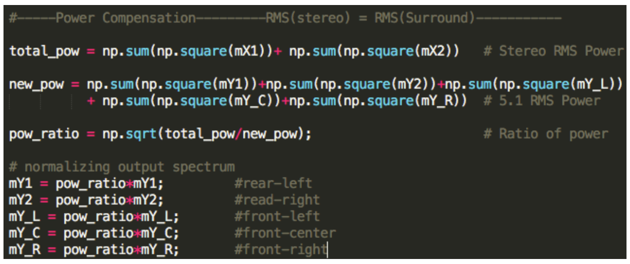


```
#normalizing the windows
gwf_sum = gwf_l + gwf_c + gwf_r
gwf_l = np.divide(gwf_l, gwf_sum)
gwf_c = np.divide(gwf_c, gwf_sum)
gwf_r = np.divide(gwf_r, gwf_sum)
#complex addition
mX_sumLR, pX_sumLR = c_add(mX1, pX1, mX2, pX2)
#equation 13
mY_L = np.multiply(gwf_l, mX_sumLR)
mY_C = np.multiply(gwf_c, mX_sumLR)
mY_R = np.multiply(gwf_r, mX_sumLR)
```

For the sub-woofer output, the kernel for a low-pass filter was implemented in time domain using a Chebyshev filter with controllable cut-off frequency and attenuation of stop band. In frequency domain, FFT(Filter) was multiplied with FFT(Stereo).

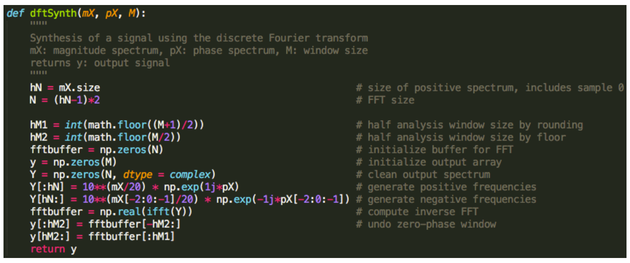
4.3. Reconstruction of time-domain signals

Before the synthesis of the time-domain signal, an RMS power compensation was carried out to make sure constant power was maintained. Otherwise, sources located at the same distance, but with a different angle the source might appear stronger or weaker in amplitude. One approach I tried was to preserve total RMS while trying to pan to the most dynamic dominant source by computing dynamic gain matrix and delays to all the surround channels. But since this caused really bad artifacts(probably because the parameters of (6) and (12) are not highly optimized yet), I stuck to a simple overall gain re-scaling technique implemented as follows:



```
#-----Power Compensation----------RMS(stereo) = RMS(Surround)-----------

total_pow = np.sum(np.square(mX1))+ np.sum(np.square(mX2))   # Stereo RMS Power

new_pow = np.sum(np.square(mY1))+np.sum(np.square(mY2))+np.sum(np.square(mY_L))
          + np.sum(np.square(mY_C))+np.sum(np.square(mY_R))  # 5.1 RMS Power

pow_ratio = np.sqrt(total_pow/new_pow);                      # Ratio of power

# normalizing output spectrum
mY1 = pow_ratio*mY1;          #rear-left
mY2 = pow_ratio*mY2;          #read-right
mY_L = pow_ratio*mY_L;        #front-left
mY_C = pow_ratio*mY_C;        #front-center
mY_R = pow_ratio*mY_R;        #front-right
```

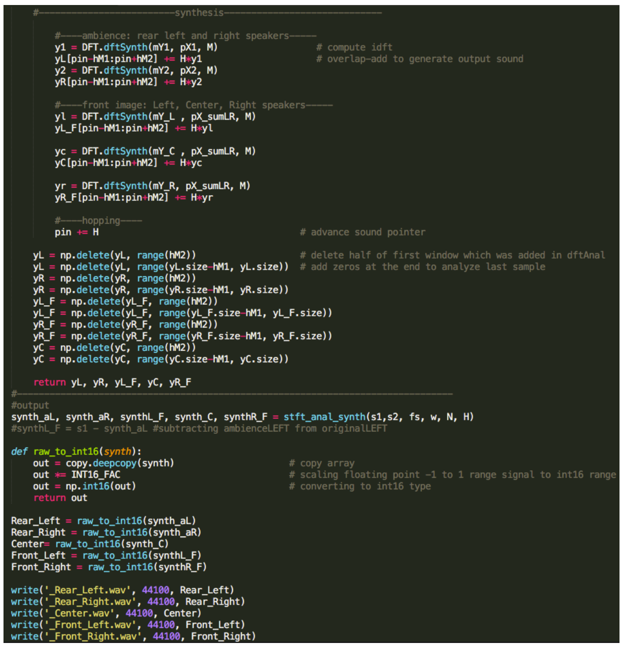
For reconstructing the time domain signals for the 5 spectrums, I implemented the dftSynth() function where the last ISTFT stage took place. In this stage the buffers were re-arranged to undo the shifts carried out for zero-phase windowing explained in section 4.1



```
def dftSynth(mX, pX, M):
    """
    Synthesis of a signal using the discrete Fourier transform
    mX: magnitude spectrum, pX: phase spectrum, M: window size
    returns y: output signal
    """
    hN = mX.size                                              # size of positive spectrum, includes sample 0
    N = (hN-1)*2                                              # FFT size

    hM1 = int(math.floor((M+1)/2))                            # half analysis window size by rounding
    hM2 = int(math.floor(M/2))                                # half analysis window size by floor
    fftbuffer = np.zeros(N)                                   # initialize buffer for FFT
    y = np.zeros(M)                                           # initialize output array
    Y = np.zeros(N, dtype = complex)                          # clean output spectrum
    Y[:hN] = 10**(mX/20) * np.exp(1j*pX)                      # generate positive frequencies
    Y[hN:] = 10**(mX[-2:0:-1]/20) * np.exp(-1j*pX[-2:0:-1])   # generate negative frequencies
    fftbuffer = np.real(ifft(Y))                              # compute inverse FFT
    y[:hM2] = fftbuffer[-hM2:]                                # undo zero-phase window
    y[hM2:] = fftbuffer[:hM1]
    return y
```

My implementation of ISTFT required some cleaning up of the arrays so that the memory is freed up for the subsequent STFT frames. Finally all the 5 time domain signals are converted to int16 format scaled from -1 to 1 so that they can be successfully exported as wav files.

In the above spectral modifications, phases were tracked and computed for faithful reconstruction of the time domain signals. In the source code several scripts for plotting the STFT frames over frequency points with evolution of time are given. Plotting the parameters and functions across the frequency domain and time domain gives a really clear intuitive picture of how different features evolve over the time-frequency planes.



```
    #-------------------------synthesis------------------------------

        #----ambience: rear left and right speakers-----
        y1 = DFT.dftSynth(mY1, pX1, M)                  # compute idft
        yL[pin-hM1:pin+hM2] += H*y1                     # overlap-add to generate output sound
        y2 = DFT.dftSynth(mY2, pX2, M)
        yR[pin-hM1:pin+hM2] += H*y2

        #----front image: Left, Center, Right speakers-----
        yl = DFT.dftSynth(mY_L , pX_sumLR, M)
        yL_F[pin-hM1:pin+hM2] += H*yl

        yc = DFT.dftSynth(mY_C , pX_sumLR, M)
        yC[pin-hM1:pin+hM2] += H*yc

        yr = DFT.dftSynth(mY_R, pX_sumLR, M)
        yR_F[pin-hM1:pin+hM2] += H*yr

        #----hopping----
        pin += H                                      # advance sound pointer

    yL = np.delete(yL, range(hM2))                    # delete half of first window which was added in dftAnal
    yL = np.delete(yL, range(yL.size-hM1, yL.size))   # add zeros at the end to analyze last sample
    yR = np.delete(yR, range(hM2))
    yR = np.delete(yR, range(yR.size-hM1, yR.size))
    yL_F = np.delete(yL_F, range(hM2))
    yL_F = np.delete(yL_F, range(yL_F.size-hM1, yL_F.size))
    yR_F = np.delete(yR_F, range(hM2))
    yR_F = np.delete(yR_F, range(yR_F.size-hM1, yR_F.size))
    yC = np.delete(yC, range(hM2))
    yC = np.delete(yC, range(yC.size-hM1, yC.size))

    return yL, yR, yL_F, yC, yR_F
#--------------------------------------------------------------------------------
#output
synth_aL, synth_aR, synthL_F, synth_C, synthR_F = stft_anal_synth(s1,s2, fs, w, N, H)
#synthL_F = s1 - synth_aL #subtracting ambienceLEFT from originalLEFT

def raw_to_int16(synth):
    out = copy.deepcopy(synth)                      # copy array
    out *= INT16_FAC                                # scaling floating point -1 to 1 range signal to int16 range
    out = np.int16(out)                             # converting to int16 type
    return out

Rear_Left = raw_to_int16(synth_aL)
Rear_Right = raw_to_int16(synth_aR)
Center= raw_to_int16(synth_C)
Front_Left = raw_to_int16(synthL_F)
Front_Right = raw_to_int16(synthR_F)

write('_Rear_Left.wav', 44100, Rear_Left)
write('_Rear_Right.wav', 44100, Rear_Right)
write('_Center.wav', 44100, Center)
write('_Front_Left.wav', 44100, Front_Left)
write('_Front_Right.wav', 44100, Front_Right)
```

5. TEST DESCRIPTIONS and RESULTS

In this section, a proof of concept of the working of my implementation is illustrated with interesting plots corresponding to the choice of parameters for the various spectral modification functions. Results from a pilot experiment conducted on a set of subjects to judge the presence of a better center image is also presented. The test signals chosen were such that there was an increasing demand on optimum choice of each parameter, without which desired output wouldn't be generated. A pre-analysis code was run to calculate optimum choice of window size, M and FFT size N depending on the attack and decay time in the envelope of the stereo signal. The signals chosen were also in the increasing order of complexity.

5.1. Ambience separation and Unmixing panned sources

To test the working of the both the concepts in general, I synthesized a stereo signal with white noise and a sine tone that sweeps from left (min panning index) to right (max panning index).

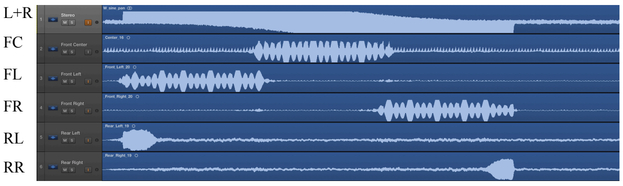


Fig.6 As desired the the ambience is successfully extracted and pushed to the surround channels. The 2 blobs present in the extreme left and right on Rear-Left (RL) and Rear-Right (RR) are because the sine tone evolves from **hard** left to **hard** right, which means zero correlation between L and R.

On the front channels, the sine tone's RMS power varies gradually thus providing a smooth transition from FL to FC and FC to FR.

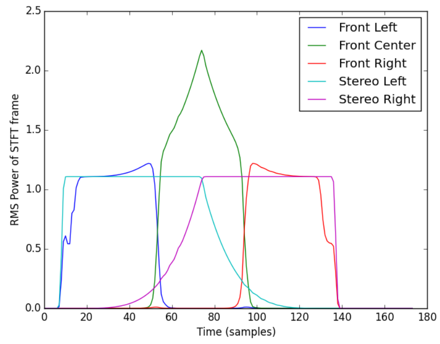


Fig.7 Evolution of RMS Power of FL, FC, FR over time

5.2. Effect of gaussian window width on unmixing

The signal generated for this scenario is bursts of 5 sine tones with white noise in the background. The 5 tones are generated with the following ratios of Left-Right panning indices:

100-0, 80-20, 50-50, 25-75, 0-100

According to the paper, a constant window width is assigned to the 3 gaussian window functions. The output for the same is as follows:

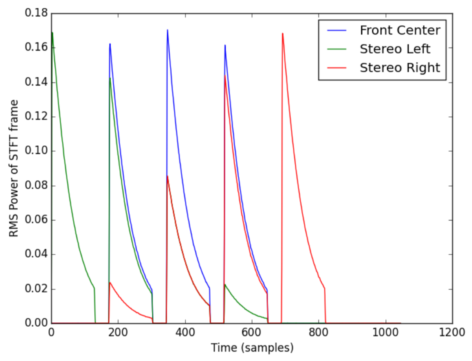


Fig.8 Evolution of Front Center panning index with time. All window widths, E = 0.125 and floor values, v = 0.1

It is noticeable that the variation of the sine tone does not vary across of the front channel enough that the listeners can perceive the movement of the tone.

In my implementation, I optimized the values of v and E, such that the windows associated to each channel could be controlled with custom values. This gave much better results and much better resolution in terms of the angle of the source in the frontal image.

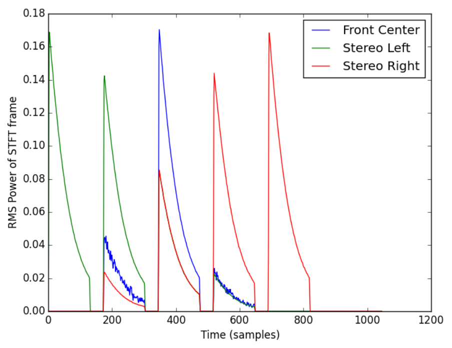


Fig.9 Evolution of Front Center panning index with time. v = 0.001, E_left = E_right = 0.01, E_center = 0.15

5.3. Effect of floor value of GWF on unmixing

The test signal chosen for this test is a stereo drum track which has kick and snare panned around the center, hi-hats between left-center and crash on right-center. The aim was to study the effect of floor value, v in the **gaussian window function** in equation (12).

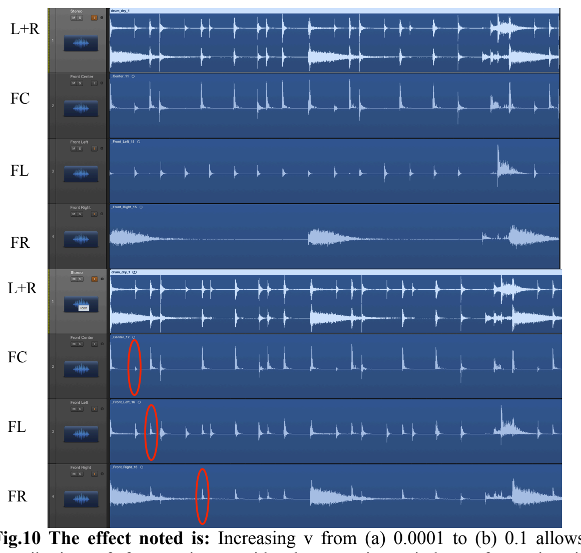


Fig.10 The effect noted is: Increasing v from (a) 0.0001 to (b) 0.1 allows the contribution of frequencies outside the gaussian window of panning index distribution, thus causing leakage of unwanted signals in the unmixing process for identifying panned sources.

5.4. Effect of sigma on ambience extraction

The signal generated for this scenario is a drum track convolved with a large hall impulse response. The aim here is to study the effect of sigma on the ambience extraction process.

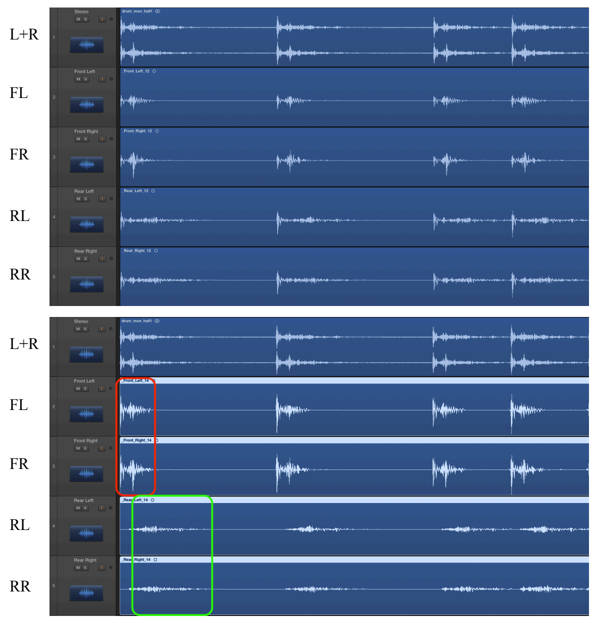
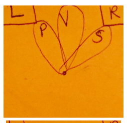
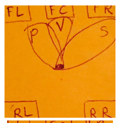
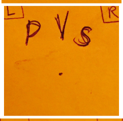
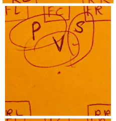
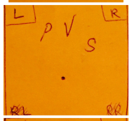
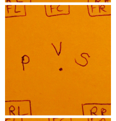
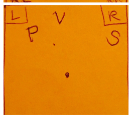
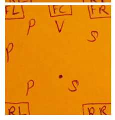


Fig.11 Increasing sigma from (a) 1 to (b) 16

The effect noticed is that sigma increases the threshold/cut off the non-linear function which is multiplied with the FFT of the Left and Right windows as seen in Fig.3. This is why, with sigma = 16, the ambience (response of the hall) is clearly extracted from the stereo drum track. In Fig.10(a), the ambience extraction function fails to identify the ambience which has slightly less coherence as compared to the drum sounds which is highly coherent.

5.5. Perception of a better / stable center sound-image

An informal pilot experiment was conducted to compare the stereo mix against the output of the 5.1 tracks generated using my implementation. The stereo mix used for this study was a 3 channel track with a vocal track and slightly panned backing piano and synth arpeggiator. The subject was asked to point to the direction of the 3 instruments heard and draw the sound image on paper:

Subject	Stereo	L - C - R	Better source separation
1			LCR
2			Stereo
3			LCR
4			LCR

The actual panning index in the mix was as follows:
[P]Piano: -0.4, **[V]**Vocals: 0.0, **[S]**Synth: 0.8, ***LCR**-LeftCenterRight*

From these results it is clear that the subjects got a stronger perception of a frontal sound image and better source separation in the up-mixed scenario. 2 subjects also mentioned that they got a more immersive feel in the up-mixed scenario.

6. CONCLUSION AND FUTURE WORK

The techniques used in this project works mainly for studio mixes that use amplitude panning methods. This project successfully implements the concepts involved in ambience extraction and source un-mixing based on time-frequency domain based techniques. The heart of this system relies on analysis of the left and right STFT frames over time to compute the coherence and similarity which eventually gives statistical parameters to track ambience and panning indices of every frequency point within a spectrum, over time.

The main modifications carried out on this algorithm involves optimizing the parameters for the various statistical functions, depending on the type of signal and artistic design of the desired output. The effects of these parameters have clearly been illustrated in section 5. Another major issue to deal with, in the implementation was to make sure that the array handling for the various parameters was carried out in an efficient manner while making sure there is a faithful reconstruction of the time domain signal after the STFT analysis and spectral modification stage. A raw implementation of the algorithm would lead to sound outputs with significant jitters, zipper noise and glitches. To avoid this, I carried out 2 levels of smoothening algorithms. One is in the frequency domain, on the panning indices across all frequency bins by applying a moving average filter based on convolution. The panning associated to each frequency bin which evolved over time was smoothened with the help of forgetting factor lambda. Another strategy applied to get rid of the glitches and to get closer to the desired output was to carry out an RMS power equalization between the FFT spectrums of the stereo and the up-mixed channels. Choosing the FFT size and STFT window frame size was another task on its own, since the user would always have to worry about time vs frequency resolution. This calls in for the first future work of applying Wavelet transforms for this project.

Another major observation I made is the compromise between distortion in output (quality of output) and the efficient working of the algorithm. This depends on the optimum choice of parameters that affect the non-linear functions that modify the original spectrums of the signals. A major future work for optimizing this would be to implement a backward control system model that optimizes the parameters like lamda, floor values, gaussian window widths, etc and this is possible using a genetic algorithm. In a stereo pair microphone based recording, the mix would have delays between the left and right signals and this would affect the commutation of the panning indices. This could work if a delay-panning index can be tracked.

From the results of the implementations of the concepts discussed above and the pilot experiment conducted, we can say that this system is capable of carrying out two-to-five upmixing thus producing a stable center channel for off-axis listening. This also perseveres the stereo image of the original recording when the listener is in the sweet spot. The ambience produced from the surround channels provide a sense of immersion to the listener as it would create a feeling of being present in a certain acoustic space. In the perspective of artistic design of this implementation, it is also interesting to note that from a stereo mix it is possible to re-pan the sources by modifying the panning indices. This idea could be used in audio-visual interactive installations and even in live concert viewings on a surround set up; in which the listener would hear the sounds produced by the audience and also pan the source of sound depending on the visual directionality of every sound source in the video.

Another future work could be to implement similar concepts for stereo to 22.2 channel up-mixing. Again, there is no clear cut definition to what 'realism' means in this context as a metric to evaluate the system. This field is highly explorable and it will be really interesting to research and implement some of the above discussed use cases.

The source code of my project is available on:
https://github.com/ajintom/audio-upmixing.git

7. ACKOWLEDGEMENT

I would like to thank my professor Dr.Philippe Depalle for giving me the opportunity for working on such an interesting project that is very relevant to my research interests. I really appreciate the constant support and help received when I was stuck with implementation aspects as well as interpretation of key concepts. I also thank Darryl Cameron for helping me set up the 5.1 Surround system to carry out my calibrations and tests for various experiments.

Carrying out this project was a really enriching experience and I would love to use the concepts I'm going to learn in the future seminar on Time-Frequency representation of Sounds to re-implement certain ideas from this project and further proceed with my research in spatial audio.

8. REFERENCES

1. C. Avendano and J. M. Jot, "Frequency Domain Techniques for Stereo to Multichannel Upmix" Audio Engineering Society Conference: 22nd International Conference: Virtual, Synthetic, and Entertainment Audio, Jun 2002
2. C. Avendano and J. M. Jot, "Ambience Extraction and Synthesis from Stereo Signals for Multichannel Audio Upmix," IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'02, Orlando, Fl.
3. D.R. Begault, 3-D Sound for Virtual Reality and Multimedia, pp. 226-229, Academic Press, Cam- bridge, 1994.
4. M. A. Gerzon, "Optimum Reproduction Matrices for Multispeaker Stereo." *AES 90th Convention*, 1991.
5. J.M. Jot, V. Larcher and J.M. Pernaux, "A Com- parative Study of 3-D Audio Encoding and Ren- dering Techniques." *AES 16th International Con- ference on Spatial Sound Reproduction*, Rovaniemi, Finland 1999.
6. V. Pulkki and M. Karjalainen, "Localization of Amplitude-Panned Virtual Sources I: Stereophonic Panning." *Journal of the Audio Engineering Society*, Vol. 49, No. 9, pp. 739-752, September 2001.
7. M. Schroeder, "An Artificial Stereophonic Effect Obtained from Single Audio Signal." *Journal of the Audio Engineering Society*, Vol. 6, pp. 74-79, 1958
8. R. Dressler, "Dolby Surround Pro Logic II Decoder - Princi- ples of Operation." URL: http://www.dolby.com/tech/l.wh.0007.PLIIops.pdf.
9. Pulkki, V., Lokki, T. and Rocchesso, D. (2011) Spatial Effects, in DAFX: Digital Audio Effects, Second Edition (ed U. Zölzer), John Wiley & Sons, Ltd, Chichester, UK. doi: 10.1002/9781119991298.ch5
10. Bai, M.R., Shih, G.-Y., Hong, J.-R.: Upmixing and downmixing two-channel stereo audio for consumer electronics. IEEE Trans. on Consumer Electronics 53, 1011–1019 (2007)
11. Dolby Laboratory, http://www.dolby.com/professional/getting-dolby- technologies/index.html
12. A. V. Oppenheim, R. W. Schafer and J. R. Buck, *Discrete-time Signal Processing*, Prentice-Hall, 1989.
13. ITU-R BS.1534-1, Method for the Subjective Assessment of Intermediate Quality Levels of Coding System, Jan. 2003.
14. Lucas O'Neil, Brendan Cassidy, 3D Spatialization and Localization, and Simulated Surround Sound with Headphones.
15. M. Bosi and R. E. Goldberg, Introduction to Digital Audio Coding and Standards, Kluwer Academic Publishers, Dec. 2002.