Adam Opsata
430451380
Digital Audio Systems 2013
Final Review

Spectral Patchwork: Content-Dependent Spectral Processing, for Mobile Audio Devices.

A significant portion of the population listens to music on smartphones. In fact 64% of smartphone owners use them to listen to music (Pewinternet.org 2011). David Byrne asserts that music is largely determined by its context, and notes that he has yet to see a way that it has adapted to digital audio players (Byrne, 2011). Many of these devices have increasingly substantial processing power and present great potential for DSP applications and new boundaries for how music can be composed and heard. This paper is an attempt to accept that challenge. Just as the intricacies in baroque music arose from people opting to listen to live music in small chambers rather than reverberant churches, and African drumming is ideal for egalitarian societies participating in music in the anechoic savannah, this paper presents a tool that adapts music to this new context: an individual experiencing music on digital music player during a commute to work.

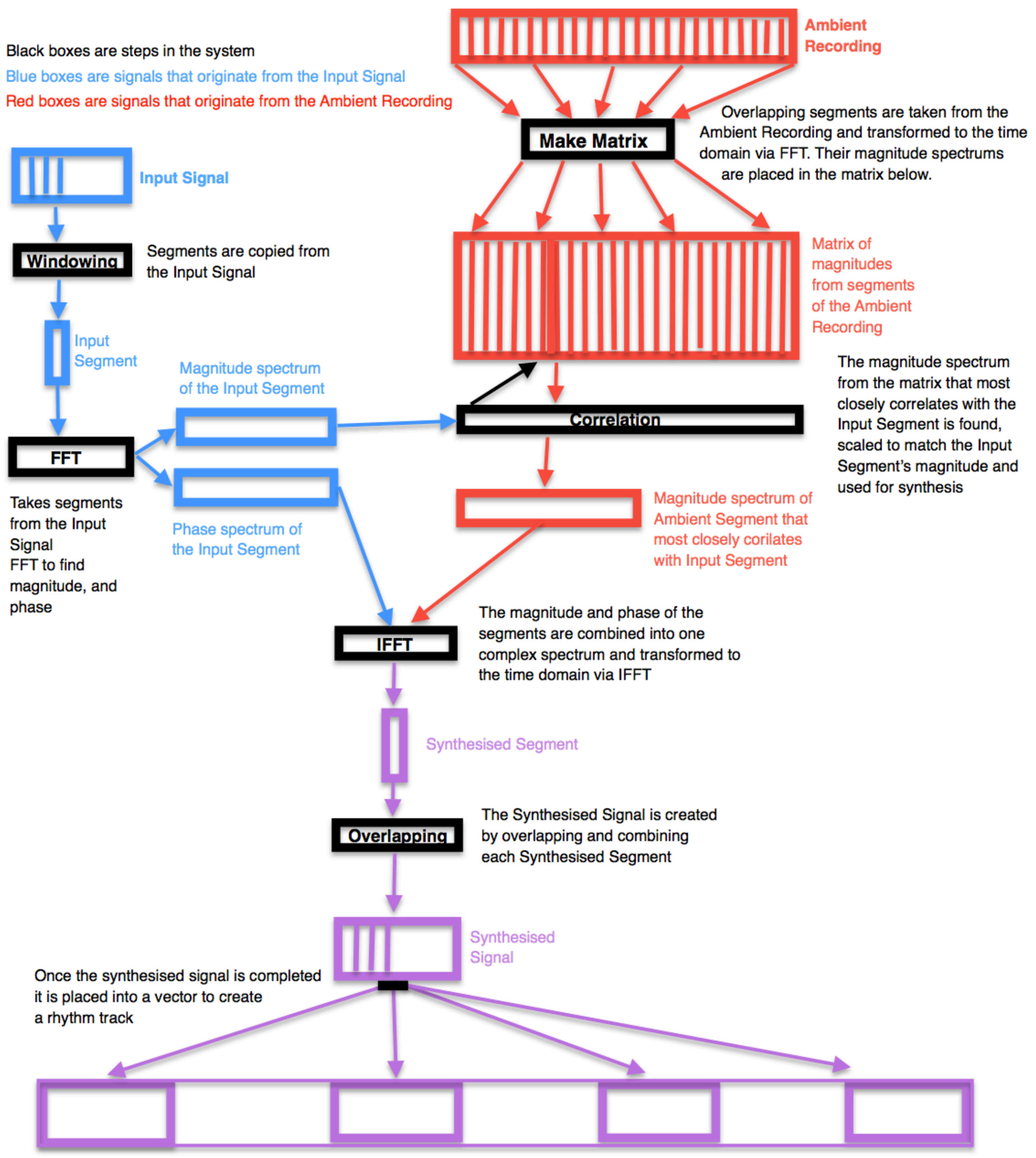
Spectral_Patchwork harnesses the processing power of a digital music player by implementing digital signal processing in semi-real-time while a listener is using it. It allows the listener to hear recorded music that is modified by the sounds currently happening around them. Every time it is run, it synthesizes a unique musical signal by using the pitches (phase information) from a signal provided by a composer, and the timbres (magnitude information) of the sounds currently happening in listener's environment. The function can then incorporate the synthesized signal into a song if the composer provides a recording of the rest of the music (the parts of the song not intended to be affected by the algorithm). The function operates in a loop: record ambient sound, analyze transform and synthesize, play back the result, and begin again. Each time it is run there is a new iteration of the input signal.

Spectral_Patchwork is intended to have 2 users, first a composer and then a listener. The composer provides the 'input_signal'. The listener simply runs the function (a script is provided). The signal flow of Spectral_Patchwork is illustrated in a block diagram provided in figure 1. It produces a synthesized_signal using the phase of the input_signal and the magnitude of a second signal. The function creates the second signal by recording the sound occurring in the listener's environment while the function is running. This second signal is called the ambient_recording.

Spectral_Patchwork is intended to be incorporated into the World Around algorithm presented in lab report two, and used in conjunction with lab report two's function Spectral_Splice. Where Spectral_ Splice is suited to modify melodic content, Spectral _Patchwork is designed for a composition's Rhythmic components. Both functions are examples of analysis - transformation – synthesis processes. They differ in that Spectral_Splice uses each signal linearly, and Spectral_Patchwork correlates the two signals and combines them by using the magnitude spectrum from the ambient_ signal that is most similar to that of the input_signal.

Figure 1:

Block Diagram for Spectral Patchwork


Black boxes are steps in the system
Blue boxes are signals that originate from the Input Signal
Red boxes are signals that originate from the Ambient Recording
Ambient Recording
Make Matrix
Overlapping segments are taken from the Ambient Recording and transformed to the time domain via FFT. Their magnitude spectrums are placed in the matrix below.
Input Signal
Windowing
Segments are copied from the Input Signal
Matrix of magnitudes from segments of the Ambient Recording
Input Segment
Magnitude spectrum of the Input Segment
The magnitude spectrum from the matrix that most closely correlates with the Input Segment is found, scaled to match the Input Segment's magnitude and used for synthesis
Correlation
FFT
Takes segments from the Input Signal FFT to find magnitude, and phase
Phase spectrum of the Input Segment
Magnitude spectrum of Ambient Segment that most closely corilates with Input Segment
IFFT
The magnitude and phase of the segments are combined into one complex spectrum and transformed to the time domain via IFFT
Synthesised Segment
Overlapping
The Synthesised Signal is created by overlapping and combining each Synthesised Segment
Synthesised Signal
Once the synthesised signal is completed it is placed into a vector to create a rhythm track

In this way it produces a best approximation of the input_signal by using the spectral data available in the environment around the listener. For instance, if a snare drum is used as an input signal, the function produces an interesting synthesized signal that still sounds somewhat like a snare, and can still serve the same function in the music.

By incorporating the sounds around the listener Spectral Patchwork is a new experience at every listen. It will not only cause a listener to pay greater attention to the world around them but also listen to a composition more times since it changes every time it is run.

A more detailed look at Spectral Patchwork's maths and signal flow

Analysis:

After the input_signal and ambient_recording are respectively identified and recorded, the ambient_recording is analyzed in its entirety using sinusoidal modeling (Zolzer, 2002) in order to create a matrix of it's magnitude spectrums. Spectral_Patchwork employs a function (called 'make_matrix' in the provided code) that copies many overlapping segments of the ambient_recording and multiplies each segment by a hanning window. It transforms each segment into the time domain by executing the complex Discrete Fourier Transform (DFT) via Matlab's built-in fft function, which relies on the equation:

$$X[k] = \frac{1}{N}\sum_{n=0}^{N-1} x[n]\, e^{-j2\pi kn/N}$$

where x[n] is the signal in the time domain , X[k], is the signal in the frequency domain. N is the number of samples in each segment, both n and k run from 0 to N-1, and j is the square root of -1.

The next few paragraphs are a little explicit regarding how the magnitude and phase spectrum are achieved and the content may be a review for some readers. Fourier analysis is founded on the fact that any complex wave can be created by adding many pure (sine and cosine) waves (Cano 2000). The DFT returns an array of complex numbers that describes a series of pure waves that, when added together, equal the wave in the segment the function is currently analyzing. Each number in the array can be represented by z in the equation

$$z = x + \mathrm{i}\, y,$$

where x is the real value and y is the imaginary value. For illustration purposes each point z can be plotted the complex plane as seen in figure 2.

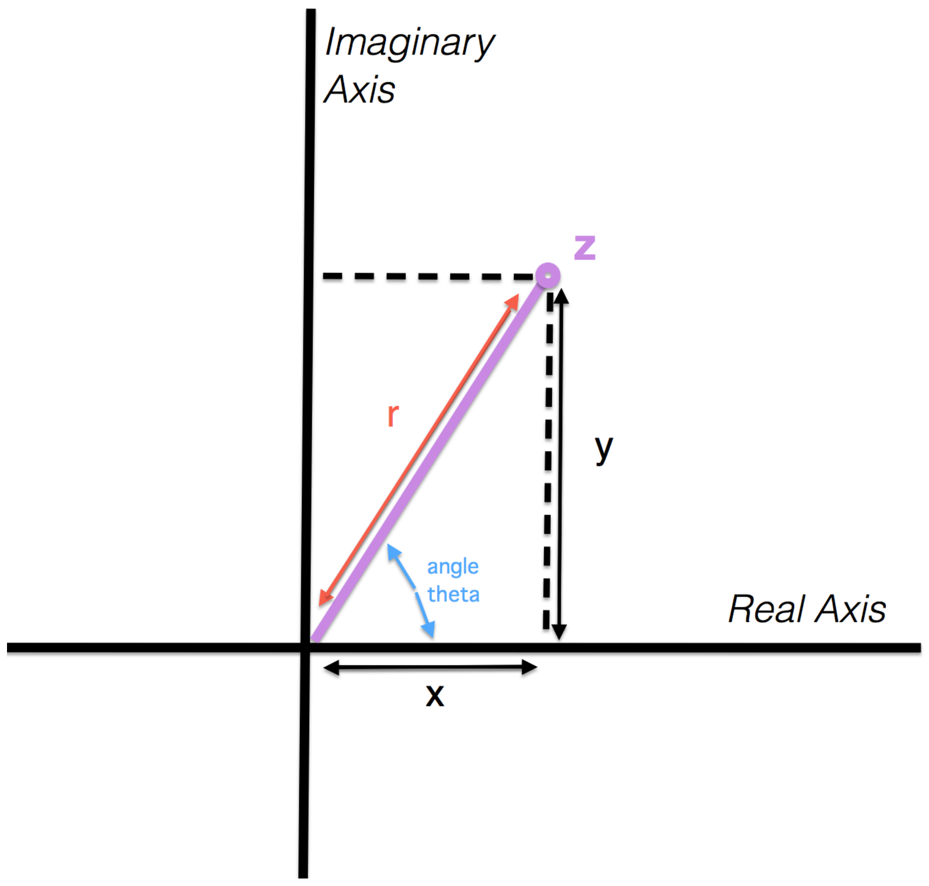


Figure 2: each value in the complex array can be plotted on the complex plane, to get a better understanding of the wave it represents.

The magnitude of each wave is equal to its distance from the origin. Simple geometry can be employed to find distance by using the Pythagorean theorem

$$r = \sqrt{x^2 + y^2}$$

where r is the distance from the origin, x is the real number and y is the imaginary number. When z is complex the Pythagorean theorem is the same as finding z's absolute value. In this way, Spectral Patchwork obtains the magnitude spectrum of each segment from the ambient_recording by transforming it to the frequency domain and finding the absolute value of each of its component waves. It then adds the magnitude spectrum of each segment into a matrix and saves it for later use. The ambient_recording's phase is not used by the function, except to generate figures for this paper. It should be noted that only segments that exceed a set rms threshold are eligible to have their magnitude entered into the matrix. If no rms threshold were set the synthesized_sigal would contain more noise, be more similar to the input and less similar to the ambient recording. This would result in a less interesting signal.

The window size of the segments could be made smaller to better represent the signals time resolution, however this would be at the detriment of low frequency resolution and processing speed. Through trial and error, It was determined that window size of 900 samples resulted in a pleasing synthesized_signal.

Correlation & Transformation:

Once the ‘ambient_matrix’ of ambient_recording magnitude spectrums is completed Spectral_Patchwork calls the function use_matrix, and the analysis of the input_signal begins. The signal is segmented and transformed into the frequency domain and it’s magnitude obtained in the same manner as the ambient_recording. Next, use_matrix finds the magnitude spectrum in the matrix that most resembles it. It first adds the input signal’s magnitude spectrum to the first column of the ambient_matrix and then uses Matlab’s corrcoef function to correlate the matrix using the equation

$$R(i,j) = \frac{C(i,j)}{\sqrt{C(i,i)C(j,j)}}.$$

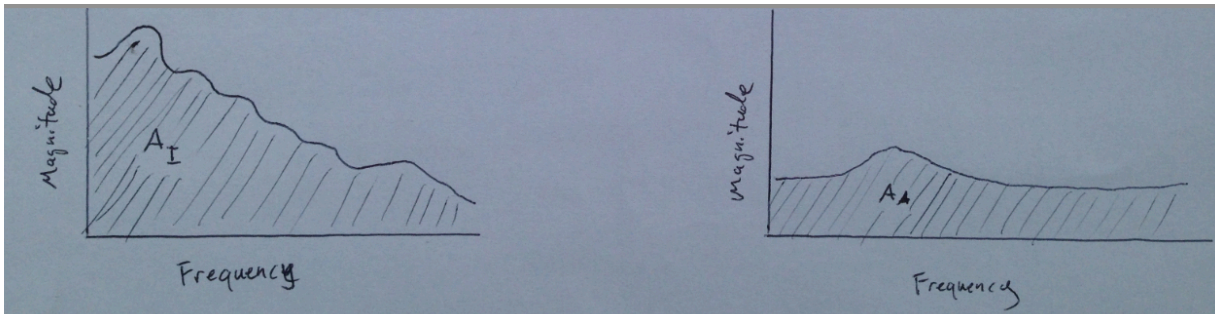
Where R is a matrix of correlation coefficients, i and j are dimensions of the ambient_matrix, and C is the covariance of the ambient_matrix. The covariance is calculated using the equation

$$\operatorname{cov}(x_1,x_2) = E\left[(x_1-\mu_1)^*(x_2-\mu_2)\right]$$

where x_1 is the magnitude spectrum of the input_segment, each column from the magnitude matrix is entered for x_2, E is the mathematical expectation and mu_i=Ex_i

The corrcoef function corilates each column in the matrix with every other column in the matrix. Since we only need to know how a column corilates with the magnitude specrtum of the input signal (which has been placed in the first column), use_matrix just reads the first row of R, and ignores the rest. The maximum correlation coefficeint from that row is found. Its coordinates corrispond to the columns in the ambient_matrix, and these coordinates are used locate the most simmalar magnitude spectrum from the ambiant_matrix.

Correlation only compares the shape of each magnitude and does not take into account the overall size of either signal. Since the ambient_segment magnitude is intended to replace the input_segment magnitude, it is necessary to obtain the level of each signal. The function employs Matlab’s trapz function in order to approximate each magnitude spectrum’s integral, or area under the curve. The integral of the two signals are shown as A_A and A_I in figure 3. An appropriate scaling coefficient is obtained by dividing the integral of the ambient_segment magnitude by the integral of the input_segment magnutude. The magnitude of the ambient_segment is multiplied by the scaling coefficient in order to match the approximate size of the magnitude of the input_segment. **Figure 3:**



The sythesised_signal uses the phase from the input_signal in order to replicate the input_signal's pitch and avoid the noise associated with random phase. Each input_segment's phase is obtained by finding z's angle from the real axis, illustrated as theta in figure 2. The angle is calculated by using the equation

$$\theta = \tan^{-1}(y/x)$$

where x is the real component of z, y is the imaginary component of z, and theta is the angle from the real axis.

Synthesis:

After finding and scaling the magnitude spectrum of the ambient _segment and calculating the phase spectrum of the input _segment, the two spectrums are combined in to a single complex frequency domain array, using the equation

complex signal = magnitude * e^(j(phase))

where j is the square root of -1 and the number e is the irrational number e (about 2.71828).

This complex array is transformed into the time domain via the inverse DFT to create a synthesized_segment. This process is repeated by use_matrix for each segment of the input _signal. Each resulting time domain segment is added and overlapped to produce a complete synthesized_signal.

The signal flow of these two segments are illustrated in the figure 4. The ambient_recording_segment is in red, the input_signal_segment is in blue and the synthesized_segment is in purple. The synthesized_segment's major peaks and troughs follow those of the input signal, which indicates it matches the signal's fundamental frequency. The finer contours of the wave match that of the ambient recording, which shows it possesses its timbre. You can see that the magnitude spectrums are similar but not identical. This is not always the case, they can be wildly different or almost identical depending on the spectral properties of the listeners environment.

Figure 4:

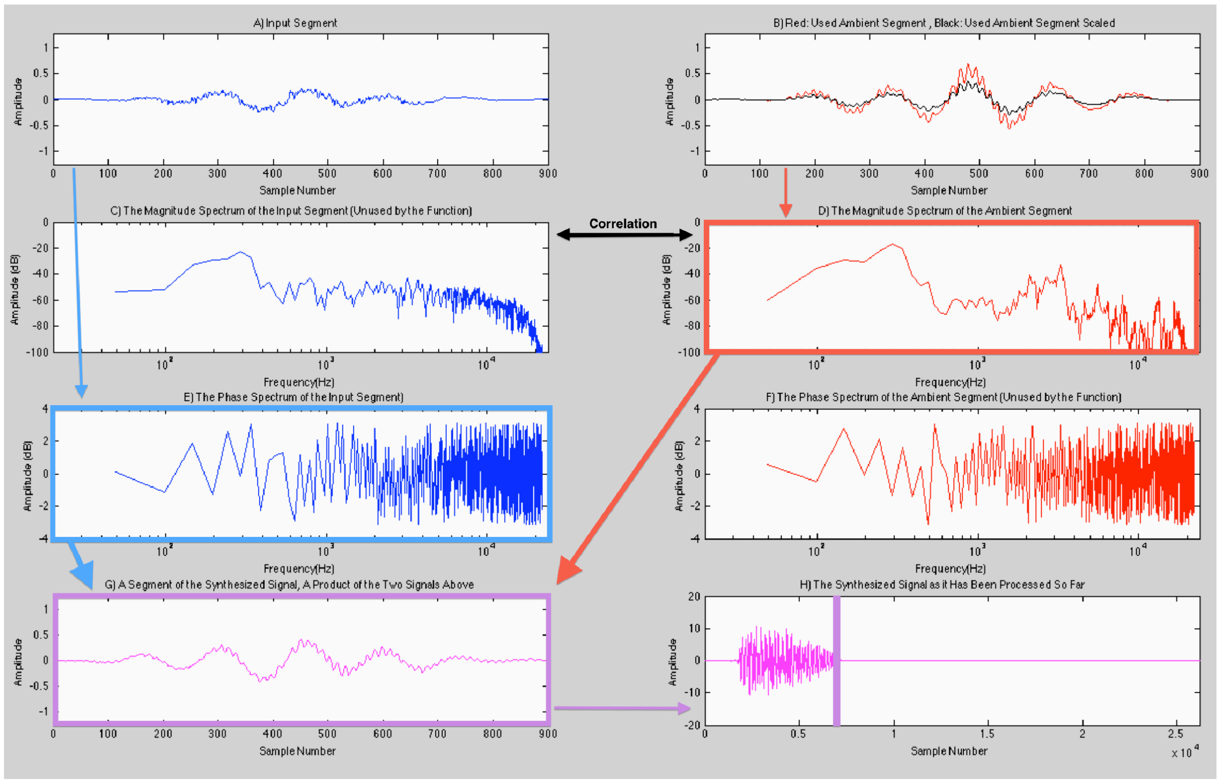


Figure 5

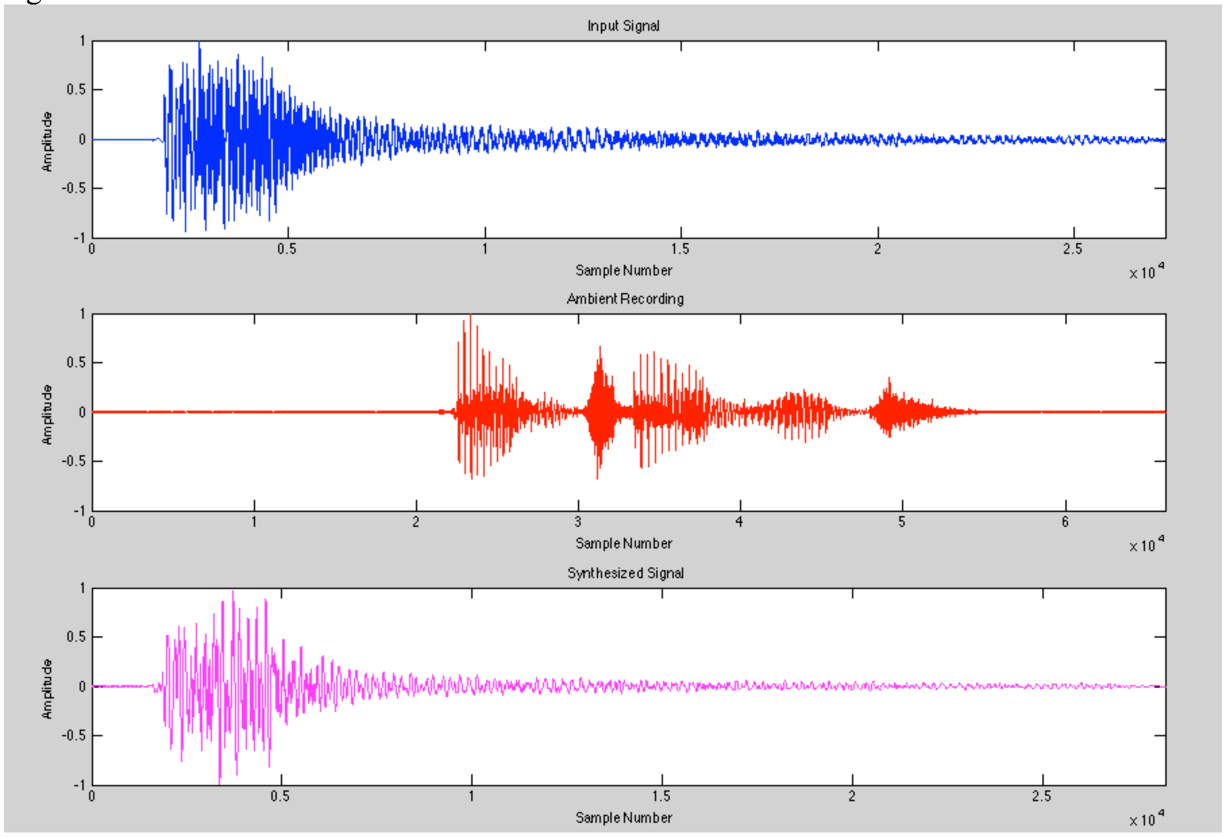


Figure 5 illustrates the waveform of the input signal ambient_recording and the synthesized_signal. In this case the input_signal is a snare drum and the ambient_recording is a person saying "ham sandwich." The altered timber of the synthesized_signal is hard to observe from the plot, however, one can observe that the sythysised_signal's waveform clearly resembles that of a snare drum. This is due to the fact that the ambient recording magnitudes are scaled and ordered with reference to the input_signal.

Examples:

The synthesized_signal is intended to be used to create a rhythm track, and although this is not implemented in the Matlab code provided, creating a longer vector by specifying where and how many times to copy they synthesized_signal, seems relatively easy to implement in Matlab. Since this whole processes takes longer than it takes for the synthesized_signal to sound (even on a laptop), it is my intention that rhythm track is played on a loop until the recording and processing of a new synthesized_signal is completed. Indeed, a whole drum kit could be synthesized if Spectral_Patchwork first uses a kick drum as an input signal and adds it to a loop, then uses a snare, a hi-hat, and any other instrument the composer would like to have transformed.

Examples:

Examples of signals have been provided in order to demonstrate the function's performance with different input_signals and ambient_recordings. A snare and a kick drum are used as inputs. Recordings of all input_signals, ambient_recordings and synthesized_signals used by the function are included for comparison.

If you wish to make your own examples please see the functioning matlab code provided. If you wish to use your own input_files they can be entered in line 32 of the Spectral Patchwork function.

References

Pewinternet.org. 2011. *Focus on Smartphone Owners | Pew Internet & American Life Project*. [online] Available at: http://www.pewinternet.org/Reports/2011/Cell-Phones/Section-2/Focus-on-Smartphone-Owners.aspx [Accessed: 7 Jun 2013].

Byrne, D. 2012. *How Music Works*. San Francisco: McSweeney's, p.1 & 27.

Zolzer, U. 2002. *DAFX: Digital Audio Effects*, John Wiley & Sons, West Sussex. pp 373-475.

Cano, P. Loscos, A. Bonada, J. de Boer, M. and Serra, X. 2000. "Voice morphing system for impersonating in karaoke applications." In *Proc. International Computer Music Conference,* pp. 109-112.