Odds and Ends*

Michael Lawrence

This work is produced by OpenStax-CNX and licensed under the
Creative Commons Attribution License 1.0†

This module exists solely to tie up loose ends left over from the previous modules. More precisely, in this module may be found the Matlab code and workspace used for the purposes of this project and the sample waveforms against which we tested our program. First, though, a discussion on up-sampling:

1 Up-Sampling

Up-sampling (that is, representing something with few samples as something with many samples) is relatively straight forward when one deals with rational multiples. First, one converts the signal into the frequency domain using the Fast Fourier Transform discussed in a previous module. The samples are then "spread out" (zeros are added) based on the rational multiple by which one is up-sampling. Then a low-pass filter and IFFT later, you're back to an up-sampled version of the original signal.

2 Limitations

No project is complete without first recognizing the limitations inherent in whatever was accomplished. The most significant drawback in our program in terms of realizing our final goal is the lack of automating the threshold detection. Without an "intelligent" program, perhaps based on a neural network, we have little hope of filtering out a particular instrument in the data representing a full orchestra. We are quite capable, however, of detecting multiple instruments so long as the multiple is not too large and we are allowed to set the threshold ourselves.

The computational complexity increases with the number of instruments (samples) tested. We create no explicit infrastructure to break a song into component tracks (at least conceptually) so that we may analyze each one against a particular set of samples representing a single instrument. Also, we would be well-advised to input the frequency domain representation of the samples to decrease computational complexity.

A further limitation is the need to input several samples from the same instrument. Ideally, we would input merely the sound of that instrument playing and modulate that one sound to create as wide a range of tones as was required. The idea here being that a given instrument has a unique frequency fingerprint that remains intact over all frequencies. This is not perfectly true (each instrument has its own idiosyncracies relating to its real-world implementation), but might prove accurate enough for our proposed analysis.

3 Future Instrument and Note Recognition Endeavors

And no limitations section is complete without some mention of how to surpass those limitations. The goal of any project is to refine the product to the point beyond which refinement is no longer possible. Because this is in practice impossible to accomplish, we will list a set of future "next steps" we or others who follow may be encouraged to take.

*Version 1.2: Dec 14, 2004 9:32 pm -0600
†http://creativecommons.org/licenses/by/1.0

To intelligently detect the relative volume of noise in a given sample, one might best be served to create a statistical filter which recognizes random noise. This statistical filter would, in theory, identify the windows which most resemble random noise. From knowledge of which windows cause noise, one might derive the volume-level (read: power-level) associated with said noise and set the threshold at some point beyond that. The upper-bound of the threshold could be found as the lowest power value for any other non-noise (as indicated by the statistical filter) window.

The threshold detection for specific instruments is more complicated: our suggestion is to develop some method of correlation or detection as-of-yet unknown to these authors (but likely known to those who research these concepts). This method would likely match frequency domain signals rather than time domain (that is, match filtering two frequency domain representations; sort of a meta-Matched Filter in terms of FFTs) using some statistical algorithm.

The computation complexity issue is trivial to solve. One must simply code the infrastructure to analyze a given signal in several channels, each acting as our entire program now acts. To convert the samples into the frequency domain, one need only FFT each sample.

The final observed limitation, too, is within our grasp. We briefly attempted a method which is promising: Mellin transformation. Essentially, when one takes a signal and transforms it into the Mellin domain (by multiplying by an exponential), one is in the position to merely phase-shift the frequency domain representation to acheive a modulation. Thus, converting back from the Mellin domain after phase-shifting the original transformed signal changes one note into another (musical modulation). This also has (**many**) more applications than simply for our particular program. Image recognition over dilation comes most immediately to mind.

4 Relevant Files

NOTE: If you choose to use our files, we would like to be informed of their use. Not because we want to inhibit any potential use of our work but rather because we want to know our audience is more than a few trillion electrons searching the internet for googly content. Imitation is, after all, the sincerest form of flattery. We hope you find our work both enlightening and useful.

4.1 Matlab Code

Our primary program.[1]

Our output-processing program.[2]

Our Up-Sampling program. [3] (Expects a vector as input; outputs a vector).

Our Up-Sampling program.[4] (Expects a struct as created from Matlab's "Import Data" feature when importing a .wav file as input; outputs a similar struct).

4.2 Clarinet Samples

The samples used for analysis of the professional recordings (i.e. recordings sampled at 44100 Hz)[5]

The samples used for analysis of the unprofessional recordings (i.e. recordings sampled at 22050 Hz)[6]

One may convert these samples to any other sampling frequency by means of the up-sampling program. The samples cover from the lowest note on a Bb Clarinet (E in the chalameau register) to the highest C in the clarion register (right before reaching the altissimo register). The lowest three notes have questionable integrity (I choose to blame the microphone ;-)).

[1] http://cnx.org/content/m12485/latest/ProjectD.m
[2] http://cnx.org/content/m12485/latest/postProcessor.m
[3] http://cnx.org/content/m12485/latest/upSampleSample.m
[4] http://cnx.org/content/m12485/latest/upSampleSamples2.m
[5] http://cnx.org/content/m12485/latest/samples2.mat
[6] http://cnx.org/content/m12485/latest/samples.mat

4.3 Music Files (Signals)

A Chromatic Scale[7] , as performed on clarinet by the up-and-coming clarinetist, Michael Lawrence.

Stravinsky's **Three Pieces for Clarinet**[8] , unknown artist.

Barber's **Adagio for Strings**.[9] , Kalman Opperman Clarinet Choir.

For our program to work, .mp3 files must first be decompressed into .wav files. We used a free program found on http://www.cnet.download.com . We would post the decompressed files but, as one might imagine, they are too large to post on Connexions.

4.4 Poster

Our Poster.[10]

In the name of thoroughness, we include a copy of the poster created for an end-of-semester poster session show-casing our project. You should find a great deal of it familiar.

[7] http://cnx.org/content/m12485/latest/chromatic_slurred.wav
[8] http://cnx.org/content/m12485/latest/Stravinsky_Three_Pieces_for_Clarinet.mp3
[9] http://cnx.org/content/m12485/latest/Barber_Adagio_Clarinet_Choir.mp3
[10] http://cnx.org/content/m12485/latest/301_project.ppt