Lab Report – Vocoder

The Matlab code I have chosen was taken from William A. Sethares' personal web page and is a working channel vocoder. The vocoder is designed to make musical instruments 'sing'. It creates a synthesized output from two inputs.

The vocoder has two input sounds, a carrier signal, often a harmonically rich sound source (often a square/pulse wave) and a modulator signal (often the human voice). This particular vocoder first determines whether it is dealing with a mono or stereo signal, both signals must be either stereo or mono; they cannot be one of each. Ideally the input signals must also be of the same length for vocoding, this vocoder actually calculates the length of each input and shortens one or the other as necessary.

With the traditional analogue vocoder, filters would be used to separate the input signal into multiple band passes. Using computers, this is often performed by an FFT. The FFT (fast Fourier transform) determines the frequency and amplitude information of the modulator signal, and splits that signal into multiple bands that will eventually feed the output signal. This split signal is then sent through an envelope (amplitude) detector. The carrier signal is also split into the same frequency bands that the modulator signal was split into. The carrier signals amplification is determined by the modulators envelope detection.

The output of the vocoder imposes the envelope of the modulator sound onto the waveform of the carrier signal. The resulting sound will have the frequency and amplitude of the modulator signal whilst using the sound (timbre) of the carrier input. Finally, the synthesized sound generated by the vocoder's filters is combined and outputted as a wav file.

The 3 standardized parameters that were included in Sethares' initial code include channel, bands and overlap.

The **channel** parameter determines the length of the FFT. It must be a multiple of two. This determines how many 'samples' the FFT will capture. A higher number generally results in a clearer signal that is more representative of the original input signal. If you listen to the examples, you will hear that there seems to be less noise and distortion in the second and third channel examples because they capture a much more detailed (larger) 'image'.

The **bands** parameter determines how many frequency bands to break the signal down into. This also impacts the end resulting sound, generally the higher the frequency bands, the more "accurate" the sound. If you select a very low number of frequency bands (say 2) the signal will only be split into two bands (eg. 0-10000Hz and 10000-20000Hz) the resulting signal is incredibly difficult to articulate (see bandtest1.wav). If the band is increased, the signal is broken down into a wider range of frequency bands and becomes more articulate. The higher the number of bands means a smaller ratio of frequencies per band, which results in more accurate synthesis (e.g 16 bands each an octave wide). Interestingly, there seems to be more noise generated in the third signal (32 bands) than in the second signal (16), suggesting that bigger isn't inherently better, and some experimentation with settings will determine the best results for each different instrument input.

The **overlap** determines the amount of overlap between successive FFT frames. It is a ratio between 0 and 1. Closer to zero means there will be a very close overlap of the FFT resulting in a more accurate signal. The closer to one, the 'muddier' the sound becomes as there is less of an overlap between the signals, meaning that spectral information may

be lost/less accurate. If you listen to the examples, the third signal is much more distorted than the previous two examples.

I have included a few more parameters in order to make the function more appropriate for my needs, these include:

The **fs** determines the sampling frequency of the inputs and is then used when writing the output wav file so that it has the same sampling frequency as the inputs and thus is as close a representation of the original signal as possible (if the sampling frequency were too high or low, the output would sound too different from the input).

And **output** is the resulting wav file from the vocoder processes. A wav file will be generated each time the function is run, this file can then be used in other applications.

The parameters can be best explained by listening to examples of the results. For each of the three parameters that will affect the outcome of a signal (excluding fs), I have performed three tests for each parameter. Listen to each example and refer to the parameters inputs below (the change in each parameter is in **bold**). Each wav file is named in correspondence with the output of each function.

```
function y = chanvocoder(carrier, modul, chan, numband, overlap, fs,
output)
```

Channel test 1 **LOW**	`a = chanvocoder(car, mod, 128, 32, 0.25, fs, channeltest1)`
Channel test 2 **MID**	`b = chanvocoder(car, mod, 512, 32, 0.25, fs, channeltest2)`
Channel test 3 **HIGH**	`c = chanvocoder(car, mod, 1024, 32, 0.25, fs, channeltest3)`

Band test 1 **LOW**	`d = chanvocoder(car, mod, 512, 2, 0.25, fs, bandtest1)`
Band test 2 **MID**	`e = chanvocoder(car, mod, 512, 16, 0.25, fs, bandtest2)`
Band test 3 **HIGH**	`f = chanvocoder(car, mod, 512, 32, 0.25, fs, bandtest3)`

Overlap test 1 **LOW**	`g = chanvocoder(car, mod, 512, 32, 0.1, fs, overlaptest1)`
Overlap test 2 **MID**	`h = chanvocoder(car, mod, 512, 32, 0.25, fs, overlaptest2)`
Overlap test 3 **HIGH**	`i = chanvocoder(car, mod, 512, 32, 0.9, fs, overlaptest3)`

Bibliography

Matlab code: http://sethares.engr.wisc.edu/vocoders/chanvocoder.m
Audio signals/how to: http://sethares.engr.wisc.edu/vocoders/channelvocoder.html