BEE 235 Final Project

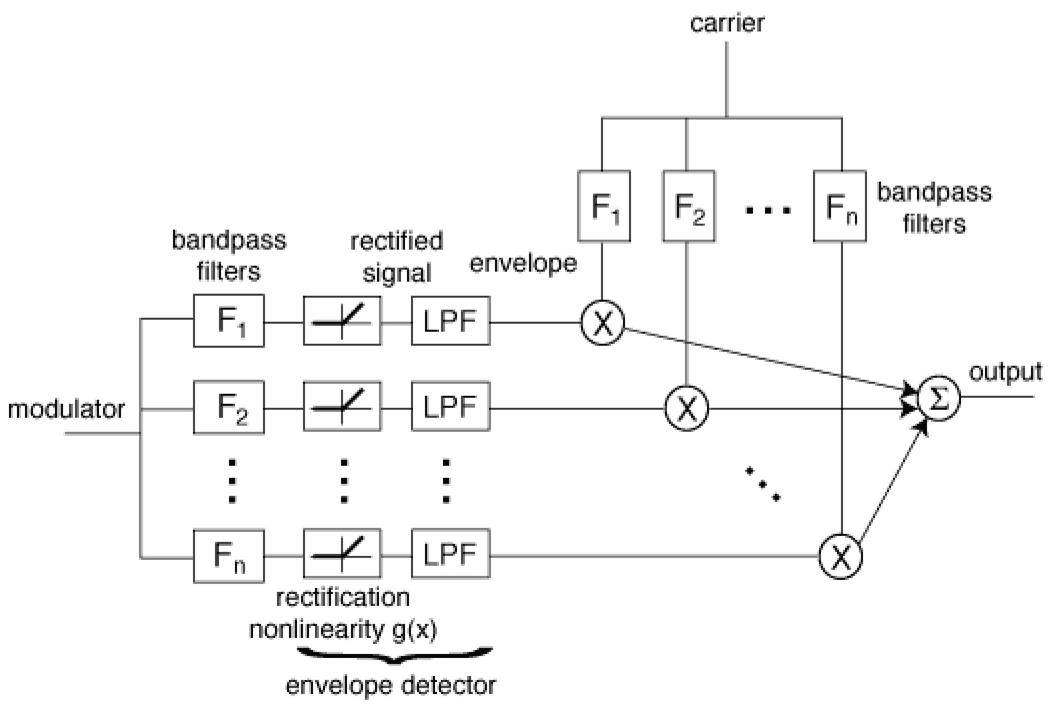
(for extra credits)

Matlab implementation of channel vocoder

The channel (or voice) vocoder operates as a bank of filters that breaks two incoming sound sources (the carrier and the modulator) into compatible frequency regions. The envelope within each subband of the modulator is imposed on the appropriate subband of the carrier, and the resulting sounds are added together. As shown below, the rectification nonlinearity followed by a lowpass filter approximates the envelope of the sound within the band. The channel vocoder can be used to generate a classic robotic-voice when modulated with speech, and it has found extensive use as a special effect in Hollywood. The channel vocoder is also often used to simulate the experience of hearing speech transduced by a cochlear implant.

Please check for details: http://120years.net/the-voder-vocoderhomer-dudleyusa1940/

The channel vocoder can be interpreted as a filter-bank that imposes the envelope of one sound (the modulator) onto the waveform of another (the carrier). The process is also similar to amplitude modulation. The envelope operation (represented here by the application of a rectification nonlinearity g(x) followed by a lowpass filter) is applied separately within each frequency band.



The following figure shows the actual implementation of a channel vocoder.

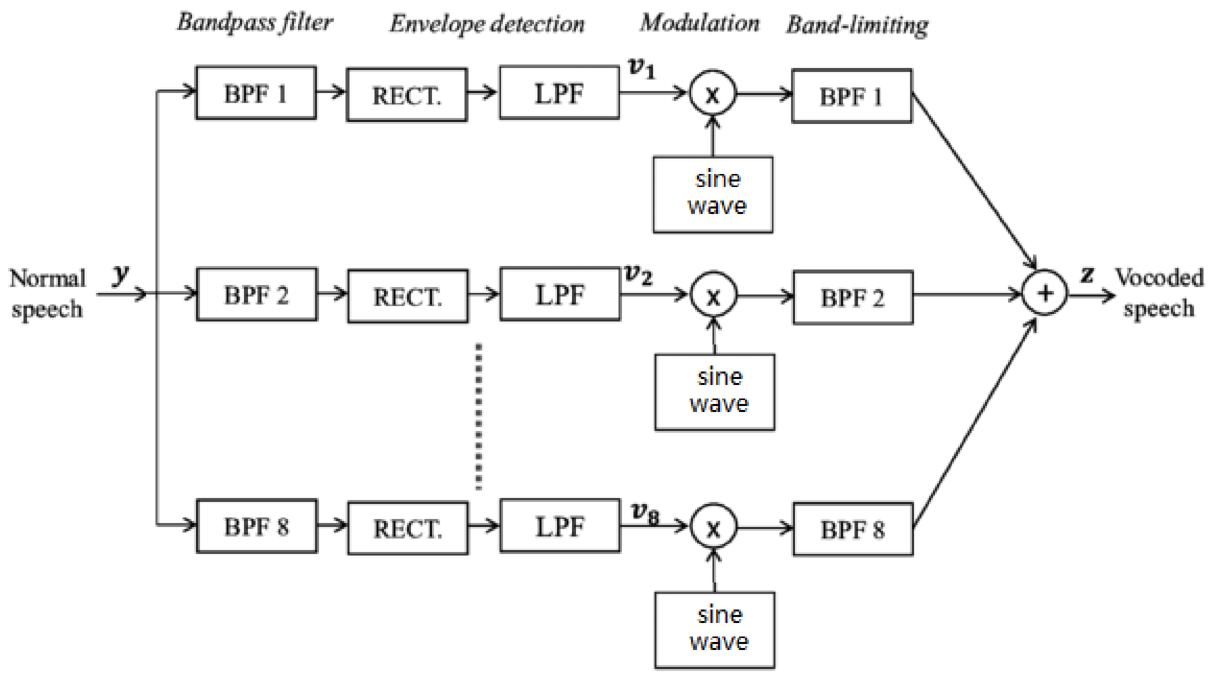


Fig. 2 The block diagram of an 8-channel voice vocoder (**RECT means rectification and it can be implemented by taking the absolute value of a sound signal; LPF=lowpass filter and BPF=bandpass filter**)

Matlab implementation of the channel vocoder

Tone-vocoded speech

Tone-vocoded speech is created by dividing the speech signal into logarithmically-spaced frequency bands. In each frequency band, the amplitude envelope (half-wave or full-wave rectifier and a low-pass filter) is extracted and then used to modulate a sine wave carrier at the center of each band. Finally, each of the modulated bands is band-passed again (same band-pass filter), and then they are recombined (summation) to create the tone-vocoded sentence. (**the band-pass filters for band-limiting are not required for tone vocoders; they are only necessary for noise vocoders**)

The example of a 6-band vocoder is as follows. A vocoded sound signal can be created by dividing the speech signal into six logarithmically-spaced frequency bands. The boundary and center frequencies of each band are shown in the table below,

	Frequency (/Hz)		
Band	Min	Center	Max
1	50	140	229
2	229	394	558
3	558	860	1161
4	1161	1713	2265
5	2265	3278	4290
6	4290	6145	8000

For different settings, the following Matlab function can be called to generate the boundary frequencies of an N-band vocoder. The output vector contains N+1 frequencies with each adjacent pair for one band-pass filter. For example, Fco=[50 229 558 1161 2265 4290 8000] for N=6 as shown in the table above. Please set fmin to 300 Hz and fmax to 6000 Hz. You can vary the number of bands N.

```
%----------------design of the logarithmic filter bank------------------------
--
```

```
% fmin: the lowest frequency to cover
% fmax: the highest frequency to cover
% N: number of bands or number of channels
% The output fco contains N+1 frequency values for N bandpass filters. For
example, [fco(i) fco(i+1)] is for the ith bandpass filter.

function fco=bands_cutoff(fmin, fmax, N);

xmin = log10(fmin/165.4+1)/2.1;
xmax = log10(fmax/165.4+1)/2.1;    %relative value

dx = (xmax-xmin)/N;
x = xmin:dx:xmax;
fco=zeros(1,N+1);

for i=1:N+1
   fco(i)=165.4*(10^(x(i)*2.1)-1);
end;
```

The band-pass filter bank in Fig. 2 can be implemented with the Matlab command butter. Try to type in 'help butter' to see how to use the command. Please set the order of each band-pass filter to 3 and the order of the low-pass filter to 2. You can use the command freqz to view the frequency response of a filter.

The low-pass filters in Fig. 2 can also be implanted with the same butter command. Please set the low-pass cutoff frequency to 400 Hz for all the bands. You can also use the command freqz to view its frequency response.

For example, the following codes demonstrate how you can design a band-pass filter from 1,000 Hz to 3,000 Hz and then plot its frequency response:

```
Fs=16000;   % sampling frequency

[BB,AA]=butter(3,[1000 3000]/(Fs/2));

% The filter order is set to 3 and [1000 3000]/(Fs/2) are normalized frequencies between
0 and 1.

[H,F]=freqz(BB,AA,256,Fs);  % Fs is the sampling frequency

plot(F,abs(H));
```

Then you can use the filter command in Matlab to filter the input signal. For example Y = filter(BB,AA,X); % X is the input and Y is the filtered signal.

Type 'help butter' in Matlab to learn how to design a band-pass or low-pass filter.

The tone or sine carrier in Fig. 2 for each band can be generated with the following commands:

```
t=(0:length(orig)-1)/Fs;  % creating a time vector

tone_carrier=sin(2*pi*Fc*t);
```

The sampling frequency of the input .wav file is Fs and the center frequency of the corresponding band is at Fc. Here, orig means a vector containing the sound signal loaded by the audioread command in Matlab.

Requirements:

1) Write your script file (e.g. using a '**for**' loop) in Matlab to implement the N-band tone vocoder as shown in Fig 2. The Matlab program should be able to process any sound file **(.wav)** you choose.

   The following code structure is for your reference only:

```
%  load a.wav file
%  call the bands_cutoff function to generate N+1 cutoff frequencies
%  initialize a vector with all zeros to store the sum of all bands.  Its length is
%  the same as the sound signal's length
%  for loop (i=1:N)
        % Design the ith band-pass filter (BPF)
        % apply the BPF
        % rectification
        % design a low-pass filter (LPF)
        % apply the LPF to the rectified signal to extract its envelope
        % calculate the center frequency of the ith band
        % create a time vector (same length as the sound signal's)
        % generate a sine wave at the center frequency of the ith band
        % multiplication (you may have to transpose one of them if you
        % receive an error "Dimensions do not match" )
        % update the sum at each iteration to get the sum of all bands
% end of the loop

% Normalize the signal
% Play the sound signal
%
```

2) The script file should be able to load a .wav sound file and process it to generate a vocoded sound.

3) Try to process different types of .wav sound files (such as a music signal) that you can download from the internet. Make sure all the filters you designed to use the actual sampling frequency of your sound signal.

4) Be able to show some of the intermediate waveforms in the vocoder processing, such as band-passed signals, envelopes extracted (after rectification and low-pass filtering), and waveforms of the original sound and vocoded sound.

5) Show demos of your vocoded sounds to demonstrate that you have correctly implemented the vocoder. Make sure your sound signals are normalized to [-1, +1] when you use the Matlab command '**sound**' to play them.

6) (optional) Varying the number of bands in vocoded speech: One important variable that determines the amount that can be understood from a vocoded sentence is the number of frequency bands that are used in synthesizing the distorted speech. Sentences synthesized with fewer than 4 bands are extremely difficult to understand. Sentences synthesized with 10 or more bands can be readily understood with very little practice and without knowing the content of the speech. Try to gradually increase the number of bands from 1, 2, 4, 6, 8, 12. Process the sample sound signals at different settings of the number of bands and then randomly choose one to present it to your partner or friend. Record the number of words that can be correctly understood. Try to decide how many bands you would need to clearly understand a sentence. Make a curve to show the percentage correct as a function of the number of bands.

7) (optional for extra credits) Use the 'SPECTROGRAM' command to show the spectrograms of the original sound and the synthesized sound. Describe the differences between them. Type '**help spectrogram**' to check how to use the Matlab function.

For example:

```
% The vector 'sig' contains the signal for spectral analysis and 'Fs' is the sampling rate of the audio signal.
```

```
[s, f, t] = spectrogram(sig, hamming(512) ,32, 1024, Fs);

surf(t, f, 20*log10(abs(s)), 'EdgeColor', 'none');

colormap(jet); view(0,90);
```

8) (optional for extra credits) try to replace the sine wave carriers with filtered white noise. In the noise vocoder, the carrier signal is generated by passing a white noise (random numbers, '**rand**' in Matlab) through the same band-pass filter (BPF in band-limiting in Figure 2). Use the 'SPECTROGRAM' command to show its spectrogram. Try to listen to the processed sound from the noise vocoder.

9) (optional for extra credits) **Creating chimaeric sounds**

By Fourier's theorem, signals can be decomposed into a sum of sinusoids of different frequencies. An alternative signal decomposition, originated by Hilbert, is to factor a signal into the product of a slowly varying envelope and a rapidly varying fine time structure. To investigate the relative perceptual importance of envelope and fine structure, you can synthesize stimuli called 'auditory chimaeras', which have the envelope of one sound and the fine structure of another.

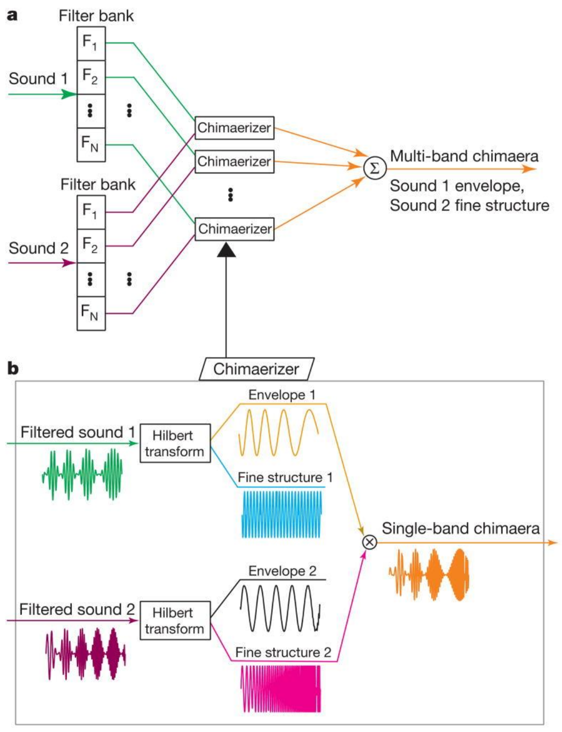
A speech signal can be viewed as a high frequency carrier signal containing the temporal fine structure (TFS) that is modulated by a low frequency envelope (ENV). A widely used method to decompose a speech signal into the TFS and ENV is the Hilbert transform. More readings about Hilbert transform: https://www.gaussianwaves.com/2017/04/extract-envelope-instantaneous-phase-frequency-hilbert-transform/

```
z = hilbert(x); % x: vector of an audio signal. To form an analytical signal (complex)

inst_amplitude = abs(z); % envelope extraction ENV

inst_phase = unwrap(angle(z)); % instantaneous phase

%Regenerate the carrier from the instantaneous phase

carrier = cos(inst_phase); % temporal fine structure TFS
```

Now, load an audio signal using audioread and then break it into **N** bands (N=1, 4 or 8). Use Hilbert transform to replace the envelope extractors (rectifier and LPF in Fig. 2). In the meantime, generate **N** temporal fine structures (TFS) using its phases. Load another audio signal and repeat the process to generate N bands of envelopes and temporal fine structures.

Finally, the envelopes and fine structures of the two audio signals were exchanged to create two chimeric stimuli. For example, use the **Envelope** of one sound and the **Fine Structure** of another sound as shown below for different bands. Make sure they are from the same frequency band when mixing them. Also, you must append zeros to the shorter sound or cut the longer signal to make them equal length before mixing them. They must have the same sampling rate too. Check here to see how to use the 'resample' function to resample a signal to match the sampling frequency of another signal:
https://www.mathworks.com/help/signal/ug/resampling.html

https://news.mit.edu/2002/hearing



Chimaeric sounds reveal dichotomies in auditory perception. Zachary M. Smith, Bertrand Delgutte, and Andrew J. Oxenham. Nature. 2002 Mar 7; 416(6876): 87–90. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2268248/

You can use different kinds of sounds to mix them. For example, the chimaeric sound has the envelopes of one speech sound with the fine structures from another speech; envelopes of one speech sound with the fine structures of another music sound; or envelopes of an English speech sound with the fine structures of another speech in a different language. Explain how you created the chimaeric sounds and play them in your presentation. Report your findings after you listen to these chimaeric sounds.

Submit a project report. The final project report should include:

Your script files

The frequency responses of all band-pass filters and low-pass filter

Envelopes from the 4 bands plotted in the same window using 'subplot' in Matlab

Waveforms of the modulated signals in the 4-channel vocoder plotted in the same window using 'subplot'

Waveforms of the original sound and its vocoded sound in the same window.

A short discussion