OBL 4101 – Project on voice coder (vocoder) – details & advices

This document summarizes the expected work and advices given**.**

General explanations

A voice coder also called vocoder is a system used to “code” voice artificially. The phase vocoder will mainly modify the phase of the signals, and “coding” here means “modification” and not data ou computer coding.

You will discover 3 types of voice modifications during this project:

- First, modify the speed of the voice without changing its pitch. The pitch is related to the fundamental frequency of the voice and corresponds to the level (high or low voice). The sound of the voice is the same but the words will be pronounced more slowly or faster.
- Second, modify the pitch of the voice without changing the speed.
- Third, apply an effect to transform the voice as if it comes from a robot (“robotisation”)

The main program “Vocodeur.m” is given including clues to solve these 3 aspects. You will modify parameters and complete the code to observe the signals of interest, to test on new voice signals of your choice, etc…

Goals

This project will allow you to discover the voice coding in autonomy and to apply results from the signal processing theory. It is not a real “Lab sessin” (TP) as :

- You will not only try to understand the codes given, but…
- You will create your own program when realizing the phase vocoder
- You will not only associate the observations and results to the theory, but also explain, justify, clarify with dedicated additional knowledge if needed and bibliography
- You will write a complete detailed report on your work (as already explained)
- So you will not only answer to the questions but you will also propose new ideas, new options, new signal processing treatments to improve the vocoder (questions are the minimum to be done); you can suggest whatever appears relevant to the topic, improvements, optimisations…
- You will transform the “TP” into a real personalized project!

Work to be done

The Matlab programs are given. As usual, the main parts are detailed. You should understand them and sometimes complete the code or modify it. You will have to create 2 main functions:

- The phase voice coder -> frequency interpolation
- The robotisation of the voice

Minimal structure of the main program:

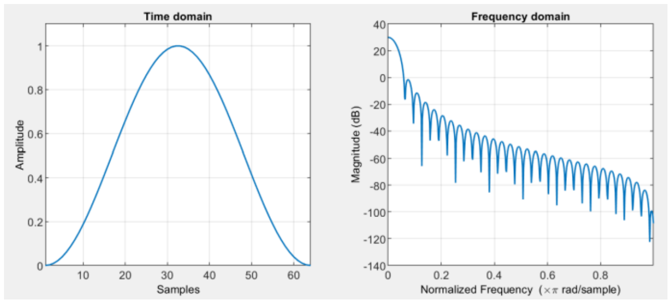
1. Speed modification -> needs function PVoc.m
2. Pitch modification -> needs function PVoc.m + ré-échantillonnage (re-sampling)
3. Robotisation of the voice -> needs function Rob.m

Details on each function:

A) Speed and pitch modifications

For these 2 effects, we will use the **Short Time Fourier Transform (STFT)** as seen en TP1 with the spectrogram introduction.
Voice signal is not strictly stationary but it can be considered as stationary over a short time period (typically 20 to 30 ms). We will work through a temporal window and calculate the FT window by window. We will use the Hanning window (see *hann* function in Matlab):

```
 hann(N) returns the N-point symmetric Hann
window in a column vector.
  hann(N,SFLAG) generates the N-point Hann
window using SFLAG window sampling.
  % Example:  Create a 64-point Hann window
    and display the result in WVTool.
  L=64;
  wvtool(hann(L))
```



The function **TFCT.m** is given to do so. And you will create (complete) the function **TFCT_Interp.m** with modifications for each window but in frequency domain.

You will come back to time domain using the inverse short time Fourier Transform, using the given function **TFCTInv.m**. The 3 functions will be called successively **PVoc.m** also given.

The difference between speed and pitch variations is that for pitch variation, you need to resample the resulting signal to keep 1 sample every Te (then keep the same sampling frequency Fe to keep the same speed). You can use the Matlab function **resample** in the program **Vocodeur.m**.

The main work in this part (effects 1 and 2) is the realization of the program **TFCT_Interp.m.** You need to understand what is done with this function and why, and implement it correctly.

NB : when you modify the pitch, you can listen to the result (and observe the time and frequency domain representations), additionning the result to the original signal … you can create duos, trios (using Extrait.wav for instance)… you can add sopranos or tenors when there are not present in the initial sound (using Halleluia.wav for instance)…

B) Robotisation

The robotisation of the voice creates a synthetic voice, mainly used in science fiction movies to make robots talking. This effect is also widely used in electronic music.

For this effect, the principle is different and is done in the time domain. You will create and use the function **Rob.m** called in the main program **Vocodeur.m**. You need to understand what this function is calculating and why, then you will implement it properly. A simple mean to obtain this effect is to use the ring modulation procedure: you will modulate the signal with 1 sinusoid (or more precisely a complex exponential) at frequency fc :

$$y_{rob}(t) = y(t).\exp(-j2\pi f_c.t)$$

As it is a complex signal, you will take the real part of it. The value of frequency fc determines the robotisation level effect. You can try with fc = 2000, 1000, 500 and 200. You will explain what happens in each case. You should notice that high frequency values are not relevant in terms of sound quality, but it helps to understand the procedure, especially in the corresponding frequency domain. Based on these trials, you will choose your own frequency of modulation fc to obtain the best sound quality.

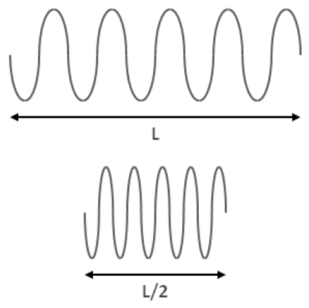
NB : the robotisation is « better » on speech than on song (test on Diner.wav for instance).
You can also test the program on your own recorded voice.

Annexes

Phase vocoder

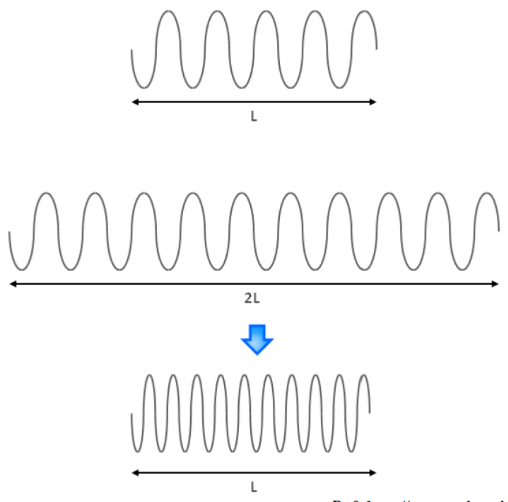
The goal is to shift the pitch of an audio signal, defined by a fundamental frequency.

Therefore this transformation is done in the frequency domain. The direct transposition will modify the time duration, see example below: 1 sinus at f0 (top fig) -> 1 sinus at 2f0 (bottom fig) which is 2 times shorter (sound 2 times faster)



Ref. http://www.guitarpitchshifter.com/algorithm.html#33

To hear the same sound but at higher level (1 octave higher for instance), you should first interpolate to increase the duration then transpose in frequency : this is the role of the function TFCT_Interp.m that you will create.

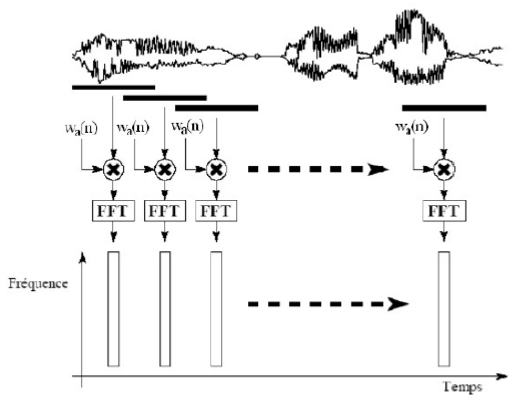


Ref. http://www.guitarpitchshifter.com/algorithm.html#33

Moreover you can not simply calculate the Fourier Transform of your signal because it is not stationary over the whole duration. You will split in frames of shorter duration (typically 20 to 30ms). In thses frames, the signal can be considered as stationary and can be transposed in the frequency domain frame after frame (using windowing and STFT).

Summary on Short Term Fourier Transform:

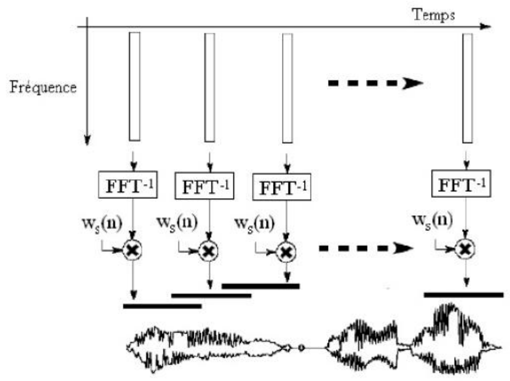
Direct STFT



$$\widetilde{X}(t_a,\nu) = \sum_{n\in\mathbb{Z}} x(n+t_a)\, w_a(n)\, e^{-j2\pi\nu n},$$

$$\widetilde{X}(t_a,\nu_p) = \sum_{n=0}^{N-1} x(n+t_a)\, w_a(n)\, e^{-j2\pi\frac{pn}{N}}$$

Synthesis by reconstruction: inverse STFT

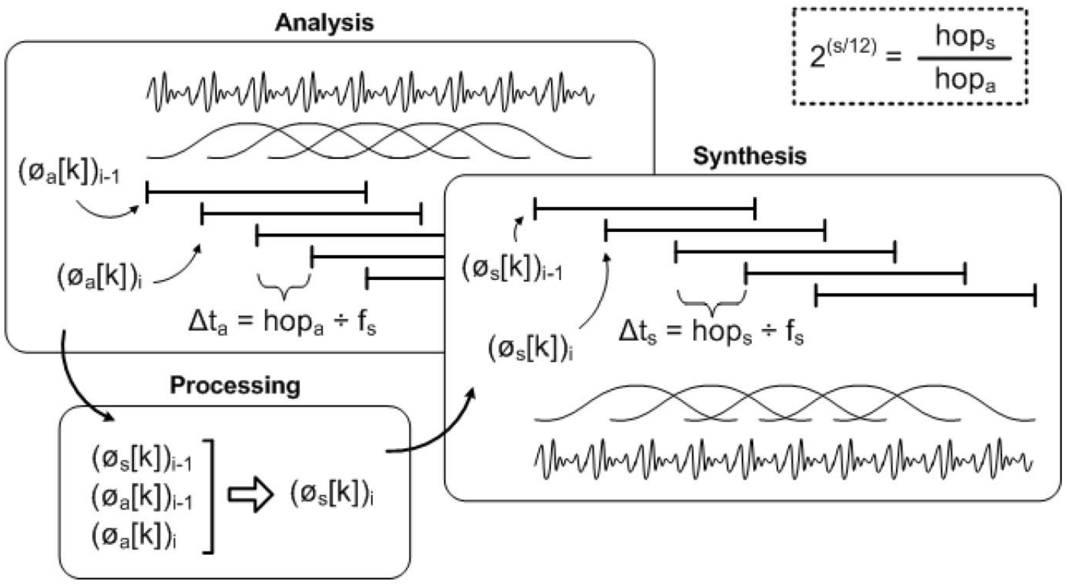


$$y(n) = \sum_u w_s(n - t_s(u))\, y_w\,(n - t_s(u), t_s(u))$$
$$\mathrm{supp}(w_s) \subset [0, N-1],\; y_w(n, t_s(u)) = \frac{1}{N}\sum_{p=0}^{N-1} Y(t_s(u), \nu_p)\, e^{j2\pi\nu_p n}$$

© Telecom Paris Tech

You need to implement 3 steps:

1. **Analysis** of the signal with STFT used to transpose in frequency domain *TFCT.m*
2. Frame after frame transposition corresponding to the signal **processing** *TFCT_Interp.m*
3. Back to the time domain to obtain the signal **synthesis** (and to be able to listen to it) *TFCTInv.m*



Ref. http://www.guitarpitchshifter.com/algorithm.html#33

Concerning Step 1. Analysis

Frames are slightly recovering on each other during a time interval Nov in programs or hop in the above figure. Before calculating the FT, you apply a Hanning window. The STFT result will be given as a matrix with N lines and N_{tr} rows. N is the number of points in the FT for each frame and N_{tr} is the number of frames. Each column represents the FT of each frame. We only keep positive frequencies in the FT (from 0 to Fe/2).

$N = 1 + Nfft/2$ if Nfft = the number of point of the calculated FT

Concerning Step 2. Processing (interpolation in frequency domain)

The reconstructed sound after inverse STFT must be as close as possible to the original. At each frequency, the phase of the modified signal should be kept close (ideally equal) to the phase of the original signal. The phase information contained between 2 frames is very important. For each frame, the processing will make the transposition (interpolation) of the spectrum (module of the FT) and a specific treatment for the phase (argument of the FT): this explains the term of phase vocoder!

Below you can find the principle of the interpolation function *TFCT_Interp.m* with the beginning given with some explanations and you will have to finish it.

```
function y = TFCT_Interp(X,t,Nov) ;
% X est la matrice issue de la TFCT
% t est le vecteur des indices sur lesquels doit être faite
% l'interpolation
% Nov est le nombre d'échantillons correspondant au chevauchement des
% fenêtres (trames) lors de la TFCT
[nl,nc] = size(X); % récupération des dimensions de X
N = 2*(nl-1); % calcul de N (= Nfft en principe)
% Initialisations
%-------------------
% Spectre interpolé
y = zeros(nl, length(t));
% Phase initiale
phi = angle(X(:,1));
% Déphasage entre chaque échantillon de la TF
dphi0 = zeros(nl,1);
dphi0(2:nl) = (2*pi*Nov)./(N./(1:(N/2)));
% Premier indice de la colonne interpolée à calculer
% (première colonne de Y). Cet indice sera incrémenté
% dans la boucle
Ncy = 1;
% On ajoute à X une colonne de zéros pour éviter le problème de
% X( : , Ncx2) en fin de boucle (Ncx2 peut être égal à nc+1)
X = [X,zeros(nl,1)];
% Boucle pour l'interpolation
%----------------------------
%Pour chaque valeur de t, on calcule la nouvelle colonne de Y à partir de 2
        %colonnes successives de X
```

Algorithm principle (general indications):

- Each frame (column) of signal Y (numbered with Ncy) is centered à time (tn) of the time vector (t). For each value (tn), we calculate the new column of Y based on 2 successive columns of X. You need to create a loop on the time values (t):

- Start of the loop: for each (tn), we collect the 2 columns of X: we need to define indix Ncx1 and Ncx2 depending on the time position (tn)
- We calculate the module My resulting from interpolation (see below)
- We calculate the column of Y: $\hat{Y}(:, Ncy) = My \,.\!* \exp(j*phi)$
- We change the phase *phi* for the next column estimation (next frame) – be careful with continuous phase
- We increase the column index of Y: $Ncy = Ncy + 1$
- End of the loop

- Explanation of frame samples calculation number Ncy of modulus $|Y|$ at time (tn):

$$My = \alpha X(:, Ncx1) + \beta(:, Ncx2)$$

$$\beta = tn - floor(tn)$$
$$\alpha = 1 - \beta$$

And phase variation for the next column estimation (next frame):

$$d\varphi = angle\big(X(:, Ncx2)\big) - angle\big(X(:, Ncx1)\big) - d\varphi_0$$

With $d\varphi = d\varphi - 2\pi.round(\frac{d\varphi}{2\pi})$ leading to new phase : $\varphi = \varphi + d\varphi + d\varphi_0$

Concerning Step 3. Synthesis

We use the inverse STFT to transpose back in the time domain by combining the windowed frames and applying the inverse FT. The function will create a vector = audio sound.