

Voice parameterization for HMM speech recognition systems

Oscar Medina Duarte

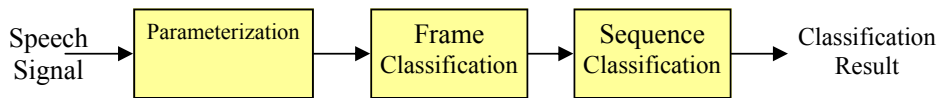
January 2007

1. Introduction

1.1 HMM based voice recognition system's architecture

In order to implement a speech recognition system based on Hidden Markov Models we first need to have an appropriate representation for the signal, which is generally accomplished by a frequency based approach. The frequency reshaping and parameterization of the original signal is made by taking small pieces (frames) of voice of a short duration, typically 30ms and overlapped by a time duration of 10ms typically. The next step is to quantize this frames in a proper way so that they have a reduced number of de-correlated components for the HMM sequence classification module to work properly.

We present the basic building blocks of a speech recognition system:



Frame classification.- This is typically achieved by using Vector Quantization and is out of the scope of this project.

Sequence Classification.- This is where the recognition takes place, this is usually performed by using the Viterbi algorithm. This is also out of the scope of the project.

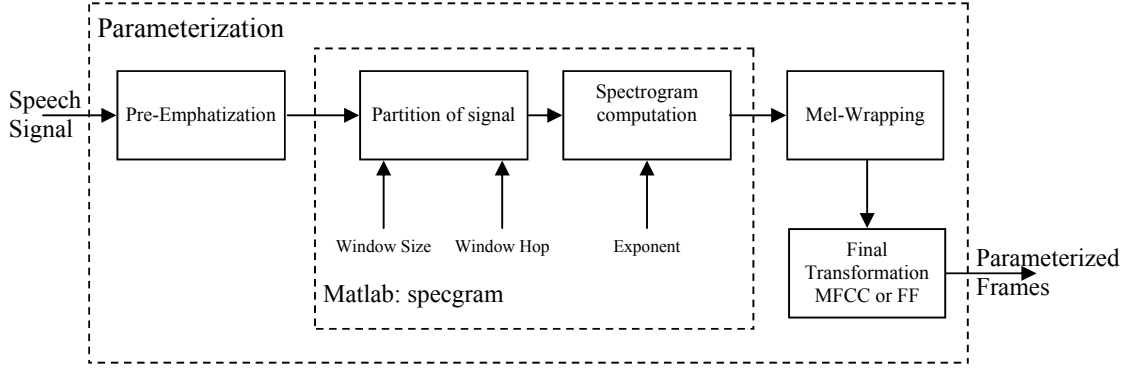
Parameterization (Speech segmentation and analysis).- This is the core of our task, it consists in transforming an input signal (ie. A wave signal) and transforming it into parameterized frames, the first block consists of a high pass filter in order to emphasize the signal we will use a filter of the shape:

$$H(z) = 1 - Kz^{-1} \quad (1)$$

Where K is variable but defaults to .97. The next step is to slide a window trough the original signal, in order to convert it into small time overlapping frames, for which later we compute the spectrogram and its magnitude with a certain exponent. The parameters to this step are the length of the window, the hop or overlap and the exponent of the magnitude computation, which are typically set to 30ms, 10ms and 2 respectively.

Mel-Wrapping or scaling is the application of a bank of filters to the spectrum in order to make it follow a perceptual model inspired by the human ear.

The final block is the application of a specific model of representation for the frames, in this case we are going to implement two among the most important and used today, Mel Frequency Cepstrum Coefficients and Frequency Filtering.



1.1.1 Mel-Scale Wrapping

The Mel scale of a certain spectrum is given by:

$$S[m] = \ln \left[\sum_{k=0}^{N-1} |X_a[k]|^2 H_m[k] \right], \quad 0 < m \leq M \quad (2)$$

Where,

$$H_m[k] = \begin{cases} \frac{2(k - f[m-1])}{(f[m+1] - f[m-1])(f[m] - f[m-1])} & k < f[m-1] \\ \frac{2(f[m+1] - k)}{(f[m+1] - f[m-1])(f[m] - f[m-1])} & f[m-1] \leq k \leq f[m] \\ \frac{2(f[m+1] - k)}{(f[m+1] - f[m-1])(f[m+1] - f[m])} & f[m] \leq k \leq f[m+1] \\ 0 & k > f[m+1] \end{cases} \quad (3)$$

Where $f[m]$ are the frequency boundaries given by:

$$f[m] = \left(\frac{N}{F_s} \right) B^{-1} \left(B(f_l) + m \frac{B(f_h) - B(f_l)}{M+1} \right) \quad (4)$$

Where F_s is the sampling frequency of the original signal in Hz, N is the size of the FFT (X_a), f_l and f_h are the lowest and highest frequencies of the filter bank in Hz. And B is given by:

$$B(f) = 1125 \ln \left(1 + \frac{f}{700} \right) \text{ and its inverse: } B^{-1}(f) = 700 \left(e^{\frac{f}{1125}} - 1 \right) \quad (5)(6).$$

1.1.2 MFCC

To obtain this final transformation MFCC of $S[m]$, we apply a DCT by using the following formula:

$$c[n] = \sum_{m=0}^{M-1} S[m] \cos \left(\frac{\pi n(m-1/2)}{M} \right), \quad 0 \leq n < M \quad (7)$$

Note that this representation is not more a frequency representation but what is called a quefency representation.

1.1.3 Frequency Filtering

To obtain this final transformation we must apply a filter to $S[m]$ whose impulse response is given by $h(k) = \{1, 0, -1\}$ with a transfer function $H(z) = z - z^{-1}$ which s equivalent to the matrix multiplication $\underline{F} = \underline{H}\underline{S}$ where the convolution matrix \underline{H} is given by:

$$\underline{H} = \begin{pmatrix} 0 & 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & \dots & 0 & -1 & 0 \end{pmatrix} \quad (8)$$

The final result of this representation is still in frequency domain and is a de-correlated version of the spectrum.

2. Exercises

Exercise 1. Generation of MFCC and FF coefficients. This exercise consists in reading or creating a wave file in matlab. And generate different representations and transformations of it until we are able to compute ift MFCC and FF coefficients.

1) Learn how to capture a sound signal in matlab by experimenting with the wavrecord function and write a function with prototype:

```
function [samples] = recordRT(time, SR)
```

where time is given in seconds, and SR is the sampling rate.

2) Write a function with the following prototype:

```
function [PS] = computePowerSpectrum(samples, sr, preemph, wintime, hoptime, squared)
```

Where,

samples: is the sampled sound signal.

sr: is the sampling rate.

preemph: is the K coefficient of eq.1.

wintime: is the window size in seconds.

hoptime: is the window sliding in seconds.

squared: is the exponent applied to the spectrum after computing its magnitude.

Which applies a pre-emphasis filter, creates the frames of the signal (wintime, hoptime in seconds), computes the Power Spectrogram of the sequence. Tip 1: Keep in mind the order, Tip 2: Keep the theory part of the exercise in mind. Tip 3: The proper number of fft points is the nearest higher power of two to the number of samples. Tip 4: The number of overlapping points for the specgram function is the difference of the window size in points and the window hop (Matlab: filter, specgram, abs).

3) Modify the previous function to have a prototype like:

```
function [PS MWPS] = computePowerSpectrum(samples, sr,preemph, wintime, hoptime, squared, nbands, minfreq, maxfreq, sumPower)
```

Where the added returning argument MWPS is the Mel-Wrapped power spectrum, and the parameters:

nbands: is the number of filter bands. (M from eq.4).

minfreq and maxfreq: are the minimum and maximum frequencies, f_l and f_h from eq.4.

sumPower: is 1 if we want to use eq.2 as it is or 0 if we want to use the variation:

$$S[m] = \ln \left[\sum_{k=0}^{N-1} (|X_a[k]| H_m[k])^2 \right], \quad 0 < m \leq M$$

4) Implement the MFCC transformation, in a function with prototype:

```
function [transform] = spectrum2cepstrum(aspectrum, numcep, liftingCoef)
```

Where,

aspectrum: is S[m].

numcep: is the number of cepstral coefficients (M from eq.7)

liftingCoef: is an optional parameter in to emphasize the transformed coefficients by raising them to the power of liftingCoef.

5) Implement the Frequency Filtering technique in a function with prototype:

```
function [FFSpectra] = frequencyFiltering(theSpectrum, liftingCoef)
```

where theSpectrum is the Mel-Wrapped spectrum S[m] and liftingCoef has the same meaning as above.

Tip: Try to implement the matrix multiplication without creating the matrix of expression 8, by doing so, you will save both, memory and CPU time.

Solution:

4) The final computePowerSpectrum function looks like this:

```
function [PS MWPS] = computePowerSpectrum(samples, sr,preemph,
wintime, hoptime, squared, nbands, minfreq, maxfreq, sumPower)
% Si existe el parametro emphatizar, si no, pues no...
if preemph ~= 0
    % High pass filter
    samples = filter([1 -preemph], 1, samples);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Compute FFT power spectrum %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% convert time to points according to sampling rate
winpts = round(wintime*sr);
steppts = round(hoptime*sr);

% Number of fft points to nearest high power of 2
NFFT = 2^(ceil(log(winpts)/log(2)));
WINDOW = hamming(winpts);
NOVERLAP = winpts - steppts;
SAMPRATE = sr;

PS =
abs(specgram(samples*32768,NFFT,SAMPRATE,WINDOW,NOVERLAP)).^squared
;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Audio spectrum scale
%%% Mel-wrapping          %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[nfreqs,nframes] = size(PS);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Matrix to convert fft to mel %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

bFrecs = computeBoundaries(nbands, minfreq, maxfreq, NFFT, sr); %
mine
wts = computeHmBank(bFrecs,winpts); %mine

% This cuts wts into a smaller array
% to make sure it fits PS's size
wts = wts(:, 1:nfreqs);

% Integrate FFT bins into Mel bins, in abs or abs^2 domains:
if (sumPower)
    aspectrum = wts * PS;
else
    aspectrum = (wts * sqrt(PS)).^2;
end

MWPS = log(aspectrum);
```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%% Filter Bank creation %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Hms] = computeHmBank(bFrecs,Spw)
% Spoken Language Processing, PH PTR
% Eq 6.140
% This function can be optimized by
% taking the zeroed samples out of the loop !!!
eMe = length(bFrecs);
Hms = zeros(eMe+2, Spw); % sightly wrong
for m = 2:(eMe - 1)
    for k = 1:Spw
        if k < bFrecs(m - 1)
            Hms(m-1,k) = 0;
        else if k <= bFrecs(m)
            Hms(m-1,k) = (2*(k - bFrecs(m-1)))/((bFrecs(m+1)-
bFrecs(m-1))*(bFrecs(m)-bFrecs(m-1)));
        else if k <= bFrecs(m+1)
            Hms(m-1,k) = (2*(bFrecs(m+1)-k))/((bFrecs(m+1)-
bFrecs(m-1))*(bFrecs(m+1)-bFrecs(m)));
        end
        end
    end
    end
    Hms = Hms(1:eMe-2,:);
end

function [bFrecs] = computeBoundaries(eMe, fLow, fHigh, nFFT,
sFrec)
% [bFrecs] = computeBoundaries(eMe, fLow, fHigh, nFFT, sFrec)
% Parameters :
% eMe - Number of filters
% fLow - Lower frequency bound
% fHigh - Higher frequency bound
% nFFT - Number of FFT points
% sFrec - Sampling frequency in Hz
%
% Spoken Language Processing, PH PTR
% Eq 6.142

bFrecs = zeros(eMe + 2,1);
for m = 1:(eMe+2)
    bFrecs(m) = invB_MelScale(B_MelScale(fLow) + (m
*((B_MelScale(fHigh)-B_MelScale(fLow))/(eMe + 1))));
end
bFrecs = bFrecs .* (nFFT/sFrec);

function [bM] = B_MelScale(freq)
% Spoken Language Processing, PH PTR
% Eq 2.6
bM = 1125 * log(1 + (freq/700));

```

```
function [frec] = invB_MelScale(bM)
% Spoken Language Processing, PH PTR
% Eq 6.144
frec = 700 * (exp(bM/1125)-1);
```

4) The spectrum2cepstrum function looks like this:

```
function [transform] = spectrum2cepstrum(aspectrum, numcep,
liftingCoef)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Convert to cepstra via DCT %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[nrow, ncol] = size(aspectrum);
% Make the DCT matrix
dctm = zeros(numcep, nrow);

% Creation of the DCT kernel matrix
for i = 1:numcep
    dctm(i,:) = cos((i-1)*[1:2:(2*nrow-1)]/(2*nrow)*pi) *
sqrt(2/nrow);
end
dctm(1,:) = dctm(1,)/sqrt(2); % make it unitary!
%%%

transform = dctm*log(aspectrum);

%%%
%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Lifting Coeficients %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if liftingCoef ~= 1
    liftwts = [1, ([1:(numcep-1)].^liftingCoef)];
    transform = diag(liftwts)*transform;
end
%%%
%%%
```

5) The frequencyfiltering functions looks like this:

```
function [FFSpectra] = frequencyFiltering(theSpectrum, liftingCoef)

[nrow, ncol] = size(theSpectrum);
FrecFiltT = zeros(nrow, nrow);

% % Target:
% 0 1 0 0 0 0
% -1 0 1 0 0 0
% 0 -1 0 1 0 0
% 0 0 -1 0 1 0
% 0 0 0 -1 0 1
% 0 0 0 0 -1 0

for i = 2:nrow
    FrecFiltT(i - 1, i) = 1;
    FrecFiltT(i, i-1) = -1;
```

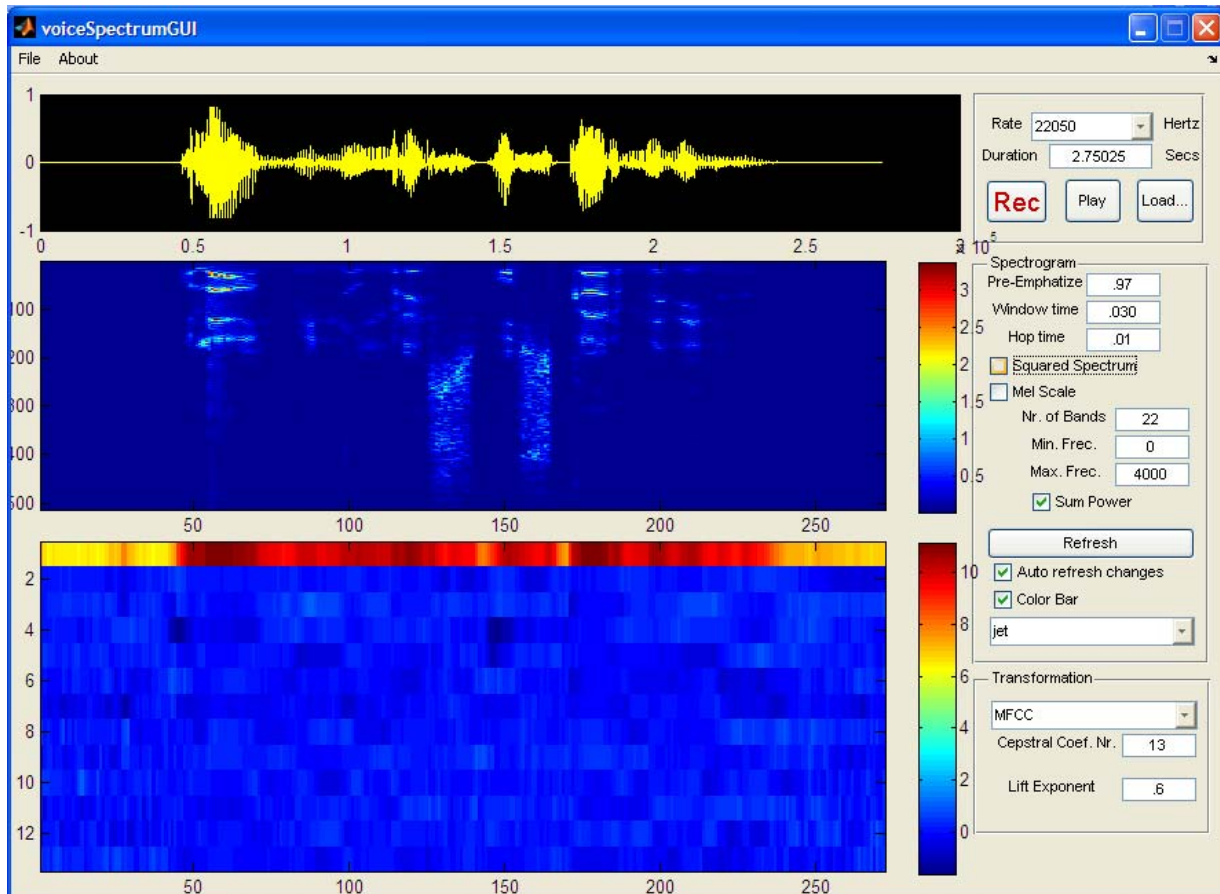
```
end

FFSpectra = FrecFiltT * theSpectrum;

[nrow, ncol] = size(FFSpectra);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Lifting Coeficients
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if liftingCoef ~= 1
    liftwts = [1, ([1:(nrow-1)].^liftingCoef)];
    FFSpectra = diag(liftwts)*FFSpectra ;
end
%%%
%%%
```

Exercise 2. Create a GUI to integrate all the functions result of the exercise 1. The target of this exercise is to make a usable GUI to interact with the different options and parameters of the previously implemented techniques. The interface has three main parts, the first one is the time domain representation of the captured or loaded signal, the second is the spectrogram representation of the signal, and the third one is the final transformation of the signal. Tip 1: Read the whole exercise before starting. Tip2: create a main function or script so that the application could be called by simply typing main at the command window of matlab.

- 1) Use GUIDE to create a GUI named voiceSpectrumGUI.fig that looks as the one in the figure.
- 2) Add functionality to the buttons on the first part of the interface by modifying the voiceSpectrumGUI.m file. The Rec Button is used to record a voice signal from the sound input interface of the computer of the specified duration and sampling Rate. Load allows the user to select a .wav file from the file system and Play is used to reproduce the sequence. Take in mind that all the three areas must be updated when a signal is loaded into the system. Tip: Make a function (for instance function `sequencePlot(handles,sequence, SF)`) that would be called every time the plots and the interface requires updating. (Matlab: `wavrecord`, `wavread`, `wavplay`, `imagesc`)
- 3) Enable the second part of the interface (Spectrogram), note that all the parameters above the Refresh button are related to the parameters in the functions you implemented previously. The elements Refresh, Auto refresh changes, color bar and color map are related to the behaviour and appearance of the interface. If the Auto refresh changes checkbox is selected, that means that changes of parameters should be computed and displayed immediately after they are made.(Matlab: `imagesc`, `colorbar`, `colormap`)
- 4) Enable the third part of the interface which allows you to select the final transformation that should be presented (MFCC or FF) in the third axes. Tip: Note that the number of coefficients field is only valid for MFCC, this means that it should be enabled/disabled or visible/invisible in the proper conditions.



Solution:

2) Note that the tags of the GUI elements depend on the one you used. This is the more important function on the GUI.

```
function sequencePlot(handles,sequence, SF)
% First We plot the time representation of the signal
numsamp = length(sequence);
durat = (numsamp/SF);
step = durat/numsamp;
timeVect = (0:step:durat+step)';

plot(timeVect(1:numsamp), sequence, 'y', 'Parent',
handles.timeAxis)
set(handles.timeAxis, 'Color', [0 0 0] );

% Then we plot the spectrum
preemph = str2num(get(findobj('Tag', 'optPreEmph'), 'String'));
wintime = str2num(get(findobj('Tag', 'optWintime'), 'String'));
hoptime = str2num(get(findobj('Tag', 'optHoptime'), 'String'));

nbands = str2num(get(findobj('Tag', 'optNBands'), 'String'));
minfreq = str2num(get(findobj('Tag', 'optMinFrec'), 'String'));
maxfreq = str2num(get(findobj('Tag', 'optMaxFrec'), 'String'));
sumPower = get(findobj('Tag', 'optSumpower'), 'Value');

if get(findobj('Tag', 'optSquared'), 'Value') == 1
    squared = 2;
else
    squared = 1;
end

[powSpec powSpecMel] = computePowerSpectrum(sequence, SF,preemph,
wintime, hoptime, squared, nbands, minfreq, maxfreq, sumPower);

% Color map
colMap = 'bone';
switch get(findobj('Tag', 'popColorMap'), 'Value')
    case 1
        colMap = 'autumn';
    case 2
        colMap = 'bone';
    case 3
        colMap = 'colorcube';
    case 4
        colMap = 'cool';
    case 5
        colMap = 'copper';
    case 6
        colMap = 'flag';
    case 7
        colMap = 'gray';
    case 8
        colMap = 'hot';
    case 9
        colMap = 'hsv';
    case 10
        colMap = 'jet';
    case 11
        colMap = 'lines';
    case 12
```

```

        colMap = 'pink';
    case 13
        colMap = 'prism';
    case 14
        colMap = 'spring';
    case 15
        colMap = 'summer';
    case 16
        colMap = 'white';
    case 17
        colMap = 'winter';
    otherwise
end

axes(handles.specAxis);
if get(findobj('Tag', 'optMelscale'), 'Value') == 1
    imagesc(powSpecMel);
else
    imagesc(powSpec);
end
colormap(colMap);

% Color Bar
if get(findobj('Tag', 'chkColorBar'), 'Value') == 1
    apos = get(handles.specAxis, 'OuterPosition');
    colorbar;
    set(handles.specAxis, 'OuterPosition', apos)
end

% Then we plot the Transformation of the spectrum
liftingCoef = str2num(get(findobj('Tag', 'edLifting'), 'String'));
numcep = str2num(get(findobj('Tag', 'edNumCep'), 'String'));

switch get(findobj('Tag', 'popTransform'), 'Value')
    case 1
        transf = spectrum2cepstrum(powSpecMel, numcep,
liftingCoef);
    case 2
        transf = frequencyFiltering(powSpecMel, liftingCoef);
end

axes(handles.transAxis)
imagesc(transf);
colormap(colMap);
% Color Bar
if get(findobj('Tag', 'chkColorBar'), 'Value') == 1
    apos = get(handles.transAxis, 'OuterPosition');
    colorbar;
    set(handles.transAxis, 'OuterPosition', apos)
end

```

2, 3, 4) All elements that implement some auto refreshable parameter contain the following lines in their callback function:

```

if get(findobj('Tag', 'chkAutorefresh'), 'Value') == 1
    btnRefresh_Callback(hObject, eventdata, handles)
end

```