ASSIGNMENT 1: PITCH ESTIMATION AND VOICING DETECTION

Carlos Arenas Gallego

ABSTRACT

The present work aims to perform a research study about different methods used for voice detection and pitch estimation. To do this, it is intended to evaluate .wav audio files from *PDA-UE* (test) and *PTDB-TUB* (train) databases, using the *autocorrelation, autocorrelation error, ADFM (Average Magnitud Difference Function) and Cepstral* methods, as techniques to obtain their pitch values. The designed program allows to automatically apply these methods to the desired database by implementing the required algorithms using the *MATLAB* working environment.

After several program debugging adjustments, based on both trial and error as well as in the study of audio signals behavior, it has been possible to reduce the *Gross voiced errors* and *MSE errors*, evaluated with respect to the reference values.

1. INTRODUCTION

In this task a pitch estimator has been implemented through four methods: autocorrelation, error autocorrelation, AMDF and cepstrum. At the beginning of this document, we explain the algorithms implemented to generate these models. Below, we detail the main program, which performs from loading files to storing the results automatically. Finally, the results obtained after a comparative evaluation are presented, ending with a series of conclusions in relation to the study carried out.

The complete source code is given in the following link:
GitHub repository

2. ALGORITHM DEVELOPMENT

This section explains how the algorithms used for pitch estimation were constructed. Also, a constant reference will be made to the .m files elaborated in this first assignment.

2.1. Autocorrelation method

This method estimates the pitch by obtaining the time difference corresponding to the second maximum of the voiced frame autocorrelation (the first is the null difference).

Since autocorrelation is a symmetric function, the search for this second maximum is obtained starting at the center of the axis towards the positive side until the end is reached, as shown in figure 1.

Fig. 1: Autocorrelation method algorithm

```
function [ f_pitch ] = autocorr_algorithm( x,fs )

autocorr=xcorr(x);
autocorr=autocorr(ceil(length(autocorr)/2):end);

                                %% Pitch estimation
% Finding the maximum
[peaks,places]=findpeaks(autocorr);
[max_peak,max_peak_place]=max(peaks);
max_pos=places(max_peak_place);

T_pitch=max_pos/fs;
f_pitch=1/T_pitch;
end
```

2.2. Error autocorrelation method

This method is performed exactly like the previous one, with the difference that the autocorrelation is performed on the error signal. This signal is obtained by filtering the audio fragment with the estimated LPC filter. Figure 2 shows the procedure followed for estimating the pitch using this method.

Fig. 2: Error autocorrelation method algorithm

```
function [ f_pitch,autocorr ]=autocorr_error_algorithm( x,fs )

p=10; % prediction order
[coef_x,g]=lpc(x,p);
x_error=filter(1-coef_x',1,x);

autocorr=xcorr(x_error);
autocorr=autocorr(ceil(length(autocorr)/2)+1:end);

                                %% Pitch estimation
% Finding the maximum
[peaks,places]=findpeaks(autocorr);
[max_peak,max_peak_place]=max(peaks);
max_pos=places(max_peak_place);

T_pitch=max_pos/fs;
f_pitch=1/T_pitch;

end
```

2.3. AMDF method

Another method to estimate the pitch of a voiced speech signal that is based on time domain is the Average Magnitude Difference Function. It consist on obtaining the AMDF function of a signal. This function describes the same information as autocorrelation, but instead of being maximized in time differences in which the samples are similar, it is minimized because they are subtracted, instead of being multiplied. The AMDF function used is the following:

$$D(k) = \frac{1}{n-k-1} \sum_{j=0}^{n-k-1} |x(j+k) - x(j)| \qquad (1)$$

Therefore, in this method we have to find the minimum which indicates the delay corresponding to the pitch frequency. We have to take into account that we should not count on the null delay since the AMDF takes its minimum value that is 0.

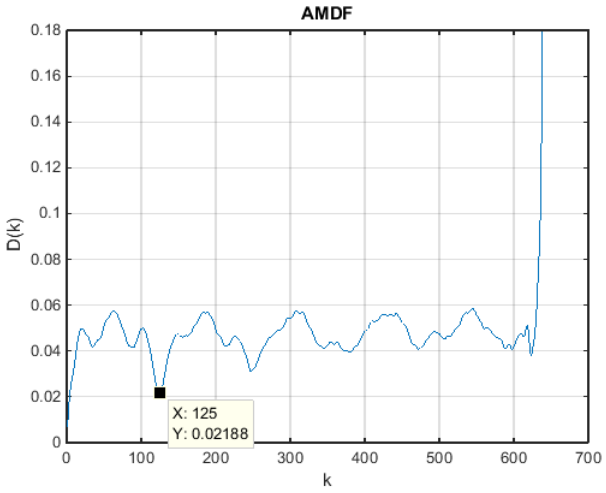
Figure 3 shows the algorithm that calculates the AMDF function of any signal or frame, as well as the estimation of its pitch.

Fig. 3: AMDF method algorithm

```
function [ f_pitch ] = amdf_algorithm( x,fs )

n=length(x);
for k=1:n-1
    addition=0;
    for j=0:(n-k-1)
        addition=addition+abs(x(j+k+1)-x(j+1));
    end
    D(k)=addition/(n-k-1);
end

% Graphical display of AMDF
    %figure(1)
    %plot(D)
    %xlabel('k'); ylabel('D(k)')
    %title('AMDF')
    %grid on

                          %% Pitch estimation
% Finding the minimum
aux_inf=floor(length(D)*0.01);
aux_sup=floor(length(D)*0.4);

[minimum min_pos]=min(D(aux_inf:end-aux_sup));
min_pos=min_pos+aux_inf;

T_pitch=min_pos/fs;
f_pitch=1/T_pitch;

end
```

An example of the AMDF function in a signal extract is given in figure 4.

Fig. 4: AMDF function example



2.4. Cepstral method

This last technique to find the pitch value works in the frequency domain and is based on the cepstrum of the signal to be analyzed.

The cepstrum of a signal is the result of calculating the Inverse Fourier Transform of a signal spectrum studied in logarithmic scale (dB). The name "cepstrum" comes from invert the first four letters of spectrum. It is complex and therefore has its real part and its imaginary part.

Cepstrum can be seen as an information of the rate of change of the different bands in the spectrum. It was originally developed to characterize the echoes of tectonic waves, which came from earthquakes and explosions. It was also used to analyze the signals captured by a radar.

The independent variable of a cepstrum chart is called "quefrency". Quefrency is a measure of frequency, but not in the sense of frequency domain. For example, if the sampling rate of an audio signal is 44.100 kHz (CD quality) and there is a large peak in the 100 sample "quefrency", this peak indicates the presence of a pitch at $\frac{44100}{100} = 441Hz$. Therefore, this peak that appears in the cepstrum indicates the period in which the harmonics of the spectrum are.

The expression for the calculation of the cepstrum is:

$$c(n) = |IDFT\{log|DFT\{x(n)\}|\}| \qquad (2)$$

When we apply the logarithm to the signal spectrum, what we are doing is to separate in sum of two logarithms with different frequency characteristics. When the IFFT or FFT is made to the logarithmic signal (both are related), the variation in the cepstral domain of the PSD of the signal (vocal tract + excitation) is represented. As the tract has a softer PSD than the excitation, the cepstral coefficients of the tract will be in

the low quefrencies whereas those of the excitation will be in the high quefrencies.

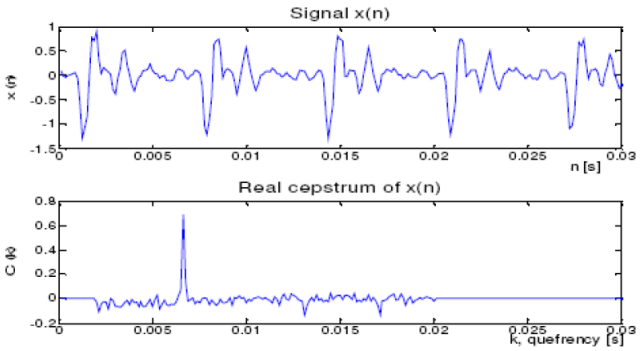
Therefore, in order to calculate the pitch frequency value we have to focus on the "high" quefrencies and know the cepstral index where the highest value peak is located.

Figure 5 shows the procedure followed for calculating the pitch value using the cepstral method, while figure 6 shows an example of an audio signal with its cepstrum values.

Fig. 5: Cepstral method algorithm

```
function [ f_pitch ] = cepstrum_algorithm( x,fs )

signalfft=fft(x);
cepstrum_signal=(ifft(log10(abs(signalfft))));


                        %% Pitch estimation
% Finding the maximum
[maximum,max_pos]=max(cepstrum_signal(20:(end/2),1));
max_pos=max_pos+20; % quefrency

f_pitch=fs/max_pos;

end
```

Fig. 6: Cepstrum



3. IMPLEMENTATION

This section explains the procedure used to obtain the final results of our study, which correspond to the pitch values of the different frames in which the audio files are decomposed. Thanks to the evaluation of these estimations we can know exactly which method best fits the actual pitch values.

For this process we use the Matlab working environment. By means of a series of .m files we will go through different steps until we reach the desired results.

The *pitch_estimation.m* script is where the entire process is executed automatically, calling to the 4 functions that implement the methods explained above when where necessary.

3.1. Loading files and setting parameters

In this first step we must choose between one of the two existing databases.

The first one is the PDA-UE and consists of 100 audio files with .wav extension and another .f0ref files where the reference pitch values corresponding to the different audio frames are found. This database is used to evaluate the estimated values of the pitch.

The second database is the PTDB-TUB and consists of a total of 4720 recorded sentences. Is composed of the same two types of files and is the one used in the case of wanting to train a prediction model. For this database an algorithm has been designed that automatically compiles all the content stored in the different subdirectories.

The sampling rate as well as the window shift are different in each database:

- PDA-UE: $f_s = 20000Hz$ and $w\_shift = 15ms$
- PTDB-TUB: $f_s = 48000Hz$ and $w\_shift = 10ms$
- In both cases the window size is 23 ms

3.2. Pitch estimation of each file

This is the most important step and also the most expensive at computational level. From a first *for* loop, the whole database is run to load and analyze the audio files.

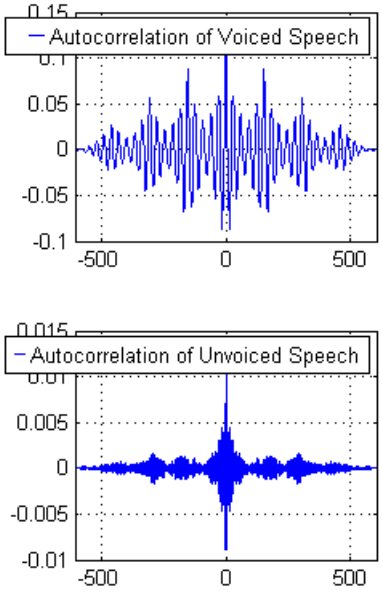
In each iteration the audio should be expanded twice the shift movement (at the beginning and end of the file) to be fully analyzed.

Then, the file is "fragmented" into different frames equal to the to the window size. Notice that each frame is overlapped with its neighbour because the window shift size is smaller than the window itself. With this we get a greater "continuity" of the temporal evolution that the pitch presents.

Once the file portion to be analyzed (frame) is selected, it is necessary to know whether it corresponds to a *voiced* or an *unvoiced* speech fragment.

We know that the autocorrelation of an audio signal is strongly related to the signal periodicity. On the other hand, we also know that the voiced signals, unlike unvoiced, have a glottal component that make the spectrum have a fundamental frequency, as well as a series of harmonics. Relating these two statements, we could say that the autocorrelation value (on average) is always greater in voiced parts. To establish an approximate threshold we are based on figure 7.

Fig. 7: Voiced and unvoiced autocorrelations



Where we see that threshold value could be between 0.01 and 0.03. After several evaluations with the test database, the threshold value was set to 0.015, so that the number of voiced frames considered as unvoiced and vice versa were the same, as we will see in the results section.

Therefore, if the frame is over the established threshold (voiced) we proceed to obtain the pitch value with the 4 methods previously explained sequentially and, in case of being under the threshold (unvoiced), is assigned as zero.

3.3. Filtering non-logical pitch values

This is an additional post-processing implementation that improves the pitch estimation in cases where randomly there are some aberrant peaks with a very high frequency value. To avoid this problem, we set the frequency values that the pitch can take, based on the frequency range presented by the speech of men and women (figure 8).

Fig. 8: Human voice frequency range

```
% Human voice range:
% Male: 85 to 180 Hz
% Female: 165 to 255 Hz

max_range=350;
```

Therefore, we set a maximum threshold at 350 Hz (A little higher than the maximum range of the man voice) so that in case of being exceeded, it is replaced by the average value of the pitch obtained by the remaining frames. After removing the peaks we apply a filtering with the function *modfilter1* to allow a ”continuity” to the temporal evolution of the pitch.

3.4. Saving .f0 files

Finally, the estimated pitch values of each .wav file are stored in a .f0 file for further evaluation.

Fig. 9: Saving estimated pitch values in .f0 files

```
audio_name=strsplit(files_wav(i).name,'.');
fileID = fopen(strcat(location,audio_name{1},'.f0'),'w');
fprintf(fileID, '%g\n', pitch_autocorr);
fclose(fileID);
```

4. RESULTS

The final results obtained after evaluating the estimated pitch values of the entire PDA-UE database with the reference ones (for the 4 methods) are shown below.

Fig. 10: Evaluation results of each method

(a) Autocorrelation method

```
### Summary
Num. frames:    22130 = 13906 unvoiced + 8224 voiced
Unvoiced frames as voiced:      1509/13906 (11%)
Voiced frames as unvoiced:      864/8224 (11%)
Gross voiced errors (+20%):     227/7360 (3.1%)
MSE of fine errors:     3.5%
--------------------------
```

(b) Error autocorrelation method

```
### Summary
Num. frames:    22130 = 13906 unvoiced + 8224 voiced
Unvoiced frames as voiced:      1509/13906 (11%)
Voiced frames as unvoiced:      864/8224 (11%)
Gross voiced errors (+20%):     159/7360 (2.2%)
MSE of fine errors:     2.8%
--------------------------
```

(c) AMDF method

```
### Summary
Num. frames:    22130 = 13906 unvoiced + 8224 voiced
Unvoiced frames as voiced:      1509/13906 (11%)
Voiced frames as unvoiced:      864/8224 (11%)
Gross voiced errors (+20%):     361/7360 (4.9%)
MSE of fine errors:     3.3%
--------------------------
```

(d) Cepstral method

```
### Summary
Num. frames:    22130 = 13906 unvoiced + 8224 voiced
Unvoiced frames as voiced:      1509/13906 (11%)
Voiced frames as unvoiced:      864/8224 (11%)
Gross voiced errors (+20%):     685/7360 (9.3%)
MSE of fine errors:     4.7%
--------------------------
```

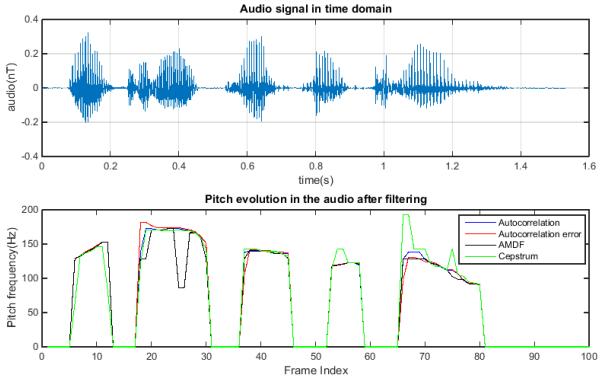
As mentioned in the previous section, the number of voiced frames considered as unvoiced and the number of unvoiced frames considered as voiced are exactly in the same proportion (11%).

This is very good news since we have found the optimal threshold that makes our decision maker very reliable. However, it is very difficult to reduce this percentage because there may be regions where the speech can be mixed version of voiced and unvoiced speech. In mixed speech, the speech signal will look like unvoiced speech, but you will also observe some periodic structure. In these cases is very easy to make a mistake.

On the other hand, the method that obtains the best results is the error autocorrelation since it is the one that has the lowest gross voice error (2.2%) and even the lowest mean square error of all (2.8%). In second place we have the autocorrelation method, followed by the AFDM method and the cepstral in the last position.

Finally, the following is the temporal evolution of the pitch estimated by the four methods for the first audio file.

Fig. 11: comparison of the 4 pitch estimation methods



5. CONCLUSIONS

This task has allowed to get better knowledge about the theoretical concepts, taking them into practice through the implementation of algorithms for voiced and unvoiced speech detection, as well as for the pitch estimation using different methods.

After the evaluation, I am able to know that the voiced and unvoiced clasification is more reliable through its autocorrelation instead of its signal amplitude and also that the pitch estimation is more reliable when is obtained using methods based on the time domain instead of on frequency domain.