1ST ASSIGNMENT: THE SPEECH AND AUDIO SIGNAL

Lucas Rodés

Universitat Politècnica de Catalunya - BarcelonaTech

1. INTRODUCTION

For this work, Wavesurfer is required. Hence, the first thing to do was to have it correctly installed. Afterwards, I executed it and recorded a sample in mono at a sampling rate of $44\,\mathrm{kHz}$ and 16 bits of quantization. The recorded signal was "Osasuna y Lérida" (does not have a special meaning). Fig. 1 shows the waveform of the recorded signal.

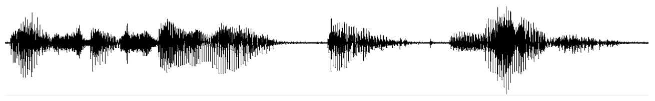

Fig. 1. Waveform

2. BANDWIDTH

The bandwidth of the recorded signal can be measured in multiple ways, depending on the threshold of dB that it is established. Fig. 2 illustrates the spectrum of a silence segment. It shows an approximately flat spectrum (note that due to the sound nature, there are peaks in the low frequency region).

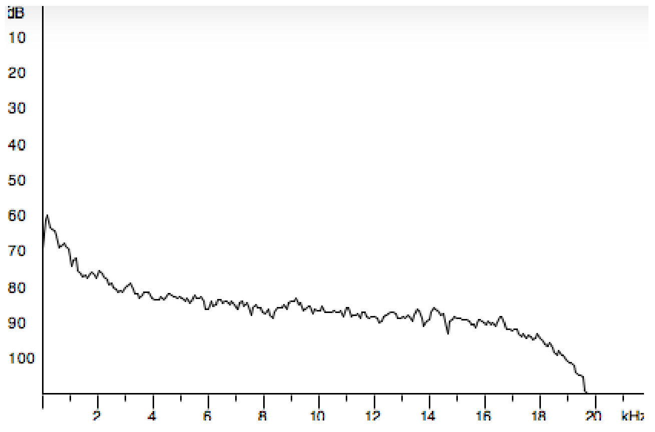

Fig. 2. FFT with 512 points, silence segment

Thereby, if we select $-80\,\mathrm{dB}$ as the threshold, we obtain a bandwidth of approximately $10\,\mathrm{kHz}$, as shown in Fig 3.

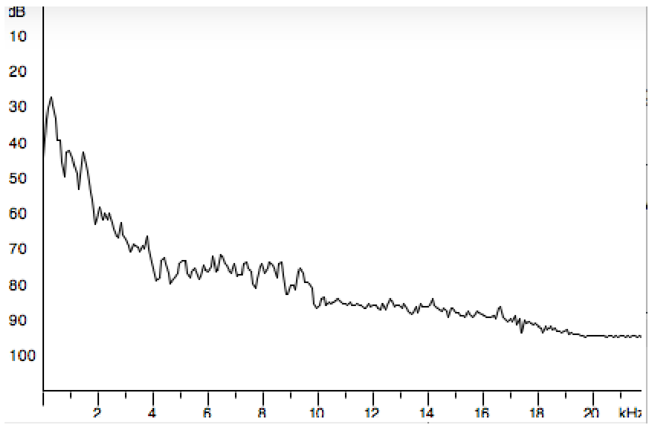

Fig. 3. FFT with 512 points, speech segment "una"

3. DOWNSAMPLING

The convert command, located in the transform menu, is used to downsample the signal. Assuming that the bandwidth obtained before is correct and considering the Nyquist theorem, a sampling rate of at least $f_s = 20\,\mathrm{kHz}$ should be enough to correctly sample the speech signal. If we downsample to $16\,\mathrm{kHz}$ the signal still can be listened with a reasonable quality. However the user might notice the decrease in quality. If we further decrease the sampling rate to $12\,\mathrm{kHz}$ or even $8\,\mathrm{kHz}$, the quality becomes much worse. Finally, at $4\,\mathrm{kHz}$ the speech segment can be easily misunderstood.

4. SOUNDS: VOICING AND FORMANTS

Fig. 4 and Fig. 5 show the time and frequency domain of a recorded voiced phoneme. As expected, a time periodicity appears. Besides, in frequency domain, the pitch is easily observed at low frequencies.

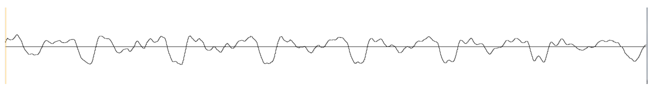

Fig. 4. Time domain of the voiced sound "u"

Fig. 7 and 6 show an unvoiced sound in time and frequency domains, respectively. It looks like noise realization.

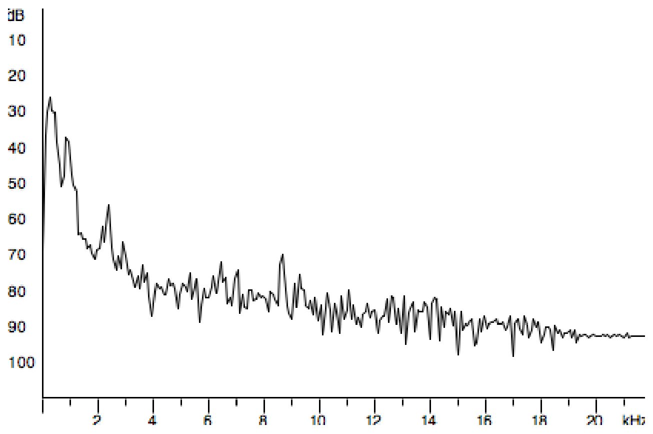
Fig. 5. Frequency domain of the voiced sound ”u”

Note that the spectral contribution at low frequencies is very reduced. On the contrary, there seems to be some peaks for higher components. In summary, voiced sounds present a periodicity whereas unvoiced sounds tend to show a more random behaviour.

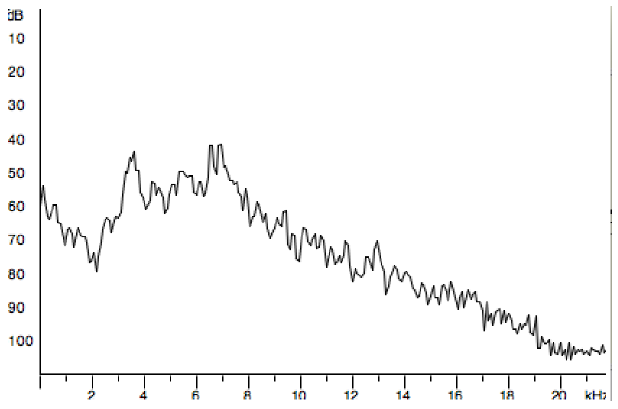
Fig. 6. Frequency domain of the voiced sound ”s”

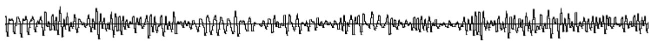
Fig. 7. Time domain of the voiced sound ”s”

Fig. 8 illustrates the recording with a transcript of the sentence (use zoom at 400%). Note, that voiced sounds present higher power. Moreover, unvoiced sounds appear twice (”s”) and voiced sounds appear multiple times (”a”, ”e”, ”i”, ”o”, ”u”, ”d”, ”n”). In addition, Fig. 9 illustrates the time evolution of the formant components (zoom for further details). This plot is obtained by using the *Formant Plot* pane tool.

Table 1 shows the values obtained from the experimental recording (In parenthesis are the values as indicated in https://es.wikipedia.org/wiki/Formante).

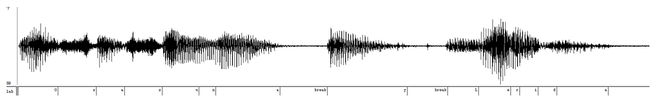
Fig. 8. Waveform of the recorded signal with transcript

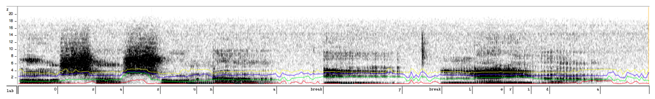
Fig. 9. Formant plot of the recorded signal with transcript

vowel	Formant 1 (Hz)	Formant 2 (Hz)
”a”	460 (1000)	1400 (1400)
”e”	350 (500)	2080 (2300)
”i”	290 (320)	2280 (3200)
”o”	514 (500)	960 (800)
”u”	303 (320)	920 (800)

Table 1. Vowel formants, experimental and reference

Finally, Fig. 10-14 show the spectrum for all 5 vowel phonemes located in the sentence (”a”, ”e”, ”i”, ”o” and ”u”).

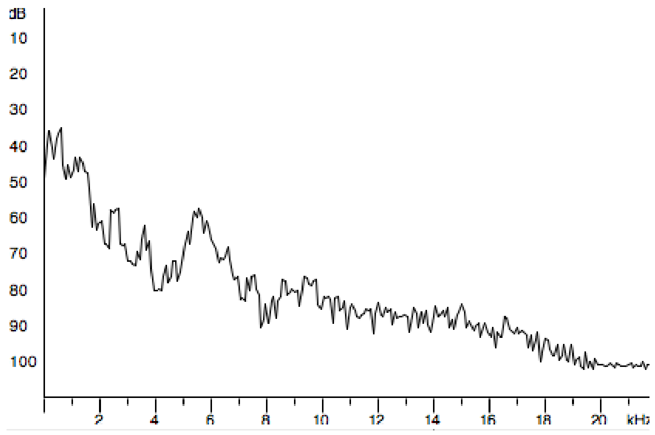
Fig. 10. ”a”

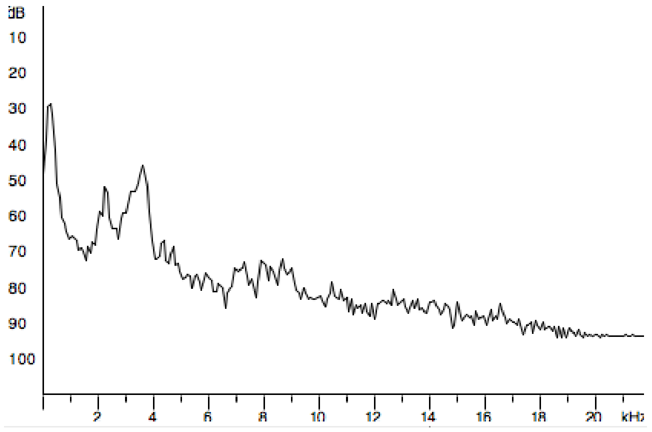

Fig. 11. ”e”

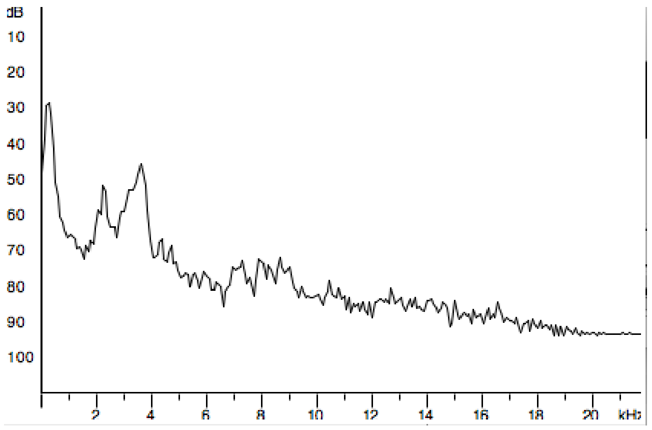

Fig. 12. ”i”

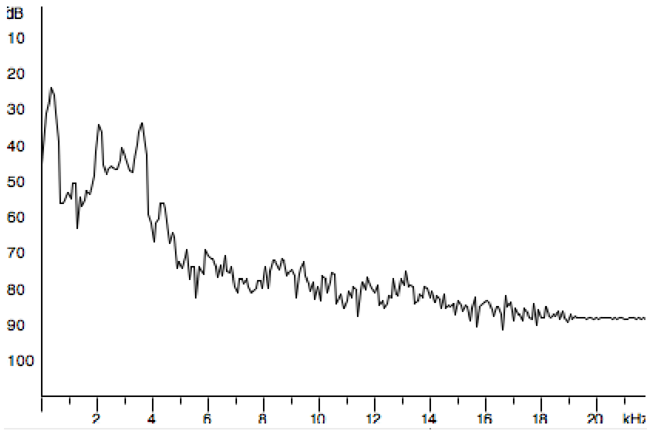

Fig. 13. ”o”

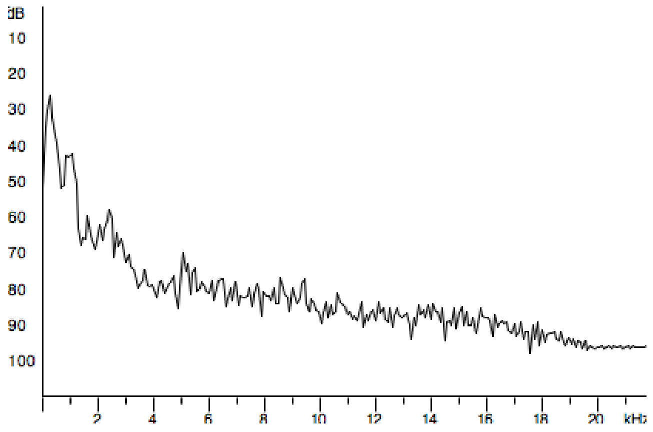

Fig. 14. ”u”

5. PITCH (F0)

In order to compute the pitch, I selected a region with a clearly distinguished voiced sound (”u”) and obtained a periodicity of $T = 0.008$s, as shown in Fig. 15. Hence, the pitch can be estimated as $1/T = 125\,\text{Hz}$.

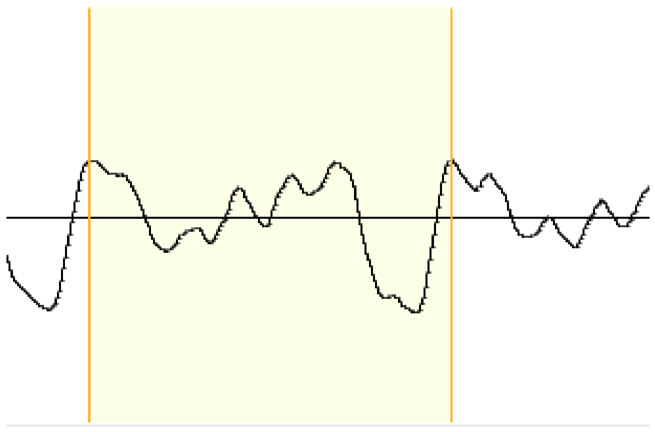

Fig. 15. ”Time domain of a segment of the voiced vowel ”u”

Next, I obtained the FFT of this segment with 2048 points and found that the first frequency peak is located at $134\,\text{Hz}$. Therefore, I can approximate the pitch frequency as the average of both values, i.e. $f_0 = 130Hz$ approximately.

Fig. 17 shows the pitch time evolution obtained by using the WaveSurfer *pitch contour* tool (use 400% zoom).

6. PITCH DETECTION: FUNCTION IMPLEMENTATION

Finally, a pitch detection method has been implemented in MATLAB. The structure of the code is the following:

- *pitchDetector.m*: Computes the pitch of a given signal segment. The code is shown in 7.

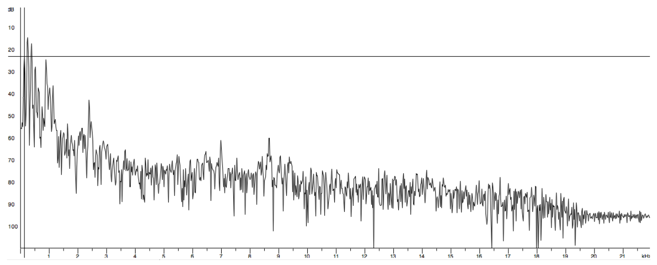

Fig. 16. Frequency domain of a segment of the voiced vowel "u" (FFT, 2048 points)

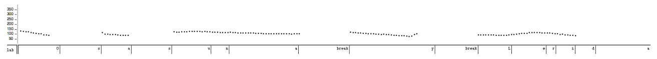

Fig. 17. Pitch contour

- *comparePitchDetection.m*: Divides a signal into L segments of N samples and obtains the pitch for each of them. Furthermore, it plots the results and compares them with the references (uploaded in ATENEA). The code is shown in 7.
- *testSelect.m* : This is a main function. Here, the user can write the file name of the signal to be analysed and compared. In addition, more than one signal can be inserted. The code is shown in 7.
- *testSelect.m* : This is a main function. Here, the pitch detector is evaluated for all the 100 .wav files of the reference database. Furthermore, it plots the Mean Square Error (MSE). The code is shown in **??**.

Regarding the pitch detection method, I employed the correlation method, where the basic idea consists on dividing an input audio signal (by using a window signal) into several shorter segments. For instance, 1 shows the segment 0 expression.

$$s[n] = x[n]w[n], \quad n = 0, \dots, N-1 \tag{1}$$

In this implementation, a rectangular window has been used. Next, the pitch is determined by finding the k that maximizes the autocorrelation $r[k]$ defined in

$$r[k] = \frac{1}{N}\sum_{i=0}^{N-k-1} s[i]s[i+k], \tag{2}$$

that is, $k_{Pitch} = \text{argmax}_k\{r[k]\}$, for $P_m < k < P_M$, where P_m and P_M are the lower and upper bounds. However, a voiced/unvoiced test was needed. In this regard, a total of four tests were implemented:

- Correlation Coefficient 1 Test: The ratio $\frac{r[1]}{r[0]}$ of the signal segment is computed. The result is compared with a threshold λ_{r1}

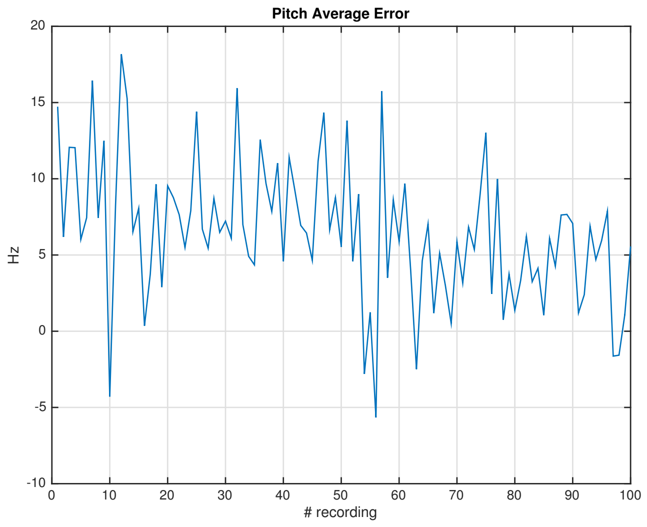

Fig. 18. Pitch average error in detection of the implemented pitch detection algorithm considering the reference values

- Zero Crossing Test: The number of zero-crossings N_{ZC} of the signal segment is obtained. This value is compared with a threshold λ_{ZC}.
- Energy Test: The energy of the signal segment E is computed. This value is compared with a threshold λ_E
- Correlation Coefficient P Test: The ratio $\frac{r[P]}{r[0]}$ of the signal segment is computed. The result is compared with a threshold λ_{rP}

For a signal segment to be considered voiced, the following expression has to be true

$$\begin{aligned} &\frac{r[1]}{r[0]} > \lambda_{r1} \quad \text{AND} \quad N_{ZC} > \lambda_{ZC} \quad \text{AND} \\ &\left(E > \lambda_E \quad \text{OR} \quad \frac{r[P]}{r[0]} > \lambda_{rP}\right) \end{aligned} \tag{3}$$

This algorithm has been chosen since it was easy to implement and did not require much computational power.

Fig. 18 shows the pitch average error (PAE) in detection taking the values from the database as the reference. Recordings 1-50 belong to male voice and recordings 50-100 belong to female voice. Note that the PAE seems to be lower for the female voice. This is computed as $1/L\sum_{n=1}^{L}\left|F0_{measured}[n] - F0_{reference}[n]\right|$, where L denotes the number of segments of the signal (i.e. number of autocorrelations computed).

7. ANNEX

In this Annex you will find the code of the implemented functions and some plots of the results. Note that in the codes, the function description might have been removed. Some plots might be also removed. Besides, the u/uv, the uv/uv and the Gross Pitch Detection is shown. For more information, please refer to the .m files.

```
function [pitch ,energy ,sonorityP ,sonority1 ,zerocrossing] = pitchDetector(s,fs ,tresh)

N = length(s); % Size of the segment
    offset = N + floor(fs/400); % Parameter to optimize the detection.

    energy = var(s) + mean(s)^2; % Compute the energy of the segment
    zerocrossing = sum(abs(s) <= 5e-5); % Number of samples crozzing the x-axis

    r = xcorr(s); % correlation
    [~,k] = max(r(offset:end)); % Obtain the sample index that maximizes
                                % the autocorrelation
    k0 = k + offset-1; % Consider the previous line shifting, remove offset
    k = k0 - N; % This is caused by the MATLAB indexing, remove offset
    sonorityP = r(k0)/r(N);
    sonority1 = r(N+1)/r(N);

    if sonority1>tresh(1) && zerocrossing < tresh(2) && (sonorityP > tresh(3)...
            || energy > tresh(4))
        pitch = fs/k;
    else
        pitch = 0;
    end

end
```

Fig. 19. pitchDetector.m, This function focusses on each of the signal segments $s[n]$ and estimates its pitch, energy, sonority and number of zero-crossings

```
function [MSE,avErr,GPE,Pv_uv,Puv_v] = comparePitchDetection(filepath, plots)
% Obtain file name, i.e. from '/path/to/file/rl001.wav' to 'rl001'
    tmp = strsplit(filepath,'/');
    tmp = strsplit(char(tmp(end)),'.');
    name = char(tmp(1));
    [x,fs] = audioread(filepath); % load file
    N = 400; % Size of the Autocorrelation window w[n]
    L = floor(length(x)/N); % Autoccorelations to be computed
    tresh(1) = 0.8; % r(1)/r(0) threshold
    tresh(2) = N/80; % Zero crossing threshold
    tresh(3) = 0.4; % r(P)/r(0) threshold
    tresh(4) = var(x(1:end/100)); % Energy threshold
    pitch = zeros(1,L); % Define vector of the pitch
    sonorityP = pitch;
    sonority1 = pitch;
    energy = pitch;
    zerocrossing = pitch;

    % Iterate and find the pitch
    for i = 1:L-1
        s = x(1+(i-1)*N:i*N); % windowed segment
        [pitch(i),energy(i),sonorityP(i),sonority1(i),zerocrossing(i)] = ...
            pitchDetector(s,fs,tresh); % obtain results for the segment
    end
    for i = 2:L-2
        if pitch(i) > 1.5*mean(pitch(pitch>0)) % correct non-sense errors
            pitch(i) = 0.5*(pitch(i+1)+pitch(i-1));
        end
    end

    % Read reference file and plot results
    fileID = fopen(['evaluation/pitch_db/' name '.f0ref'],'r');
    formatSpec = '%f';
    pitch_ref = fscanf(fileID,formatSpec);
    % Obtain the parameters of the loaded reference file
    Nref = floor(length(x)/length(pitch_ref));
    Lref = length(pitch_ref);
    t_reference = Nref/fs*(0:1:(Lref-1));
    % Interpolate the computed pitch vector, in order to compute the MSE
    t_mine = N/fs*(0:1:(L-1));
    pitch_int = interp1(t_mine,pitch,t_reference)';
    pitch_int(isnan(pitch_int))=0; % Avoid sensless values, like NaN
    % Compute the MSE and the average error in Hz
    avErr = mean(pitch_int-pitch_ref);%immse(pitch_int,pitch_ref);
    MSE = immse(pitch_int,pitch_ref);
    GPE = sum(abs(pitch_ref-pitch_int)>0.2*pitch_ref)/length(pitch_ref);
    % Probability of voiced given unvoiced
    Pv_uv = sum(pitch_int(pitch_ref==0)> 0)/sum(pitch_ref==0);
    % Probability of unvoiced given voiced
    Puv_v = sum(pitch_int(pitch_ref>0) == 0)/sum(pitch_ref>0);

end
```

Fig. 20. comparePitchDetection.m, This function analyses the pitch of a given audio signal and compares the results with a reference vector of values.

```
%   Execute this script and select a recording file to analyze.

close all;
clc

[file,path] = uigetfile('.wav'); % load file

disp('running...');

filepath = [path file];
% execute pitch detection algorithm
[MSE,avErr,GPE,Pv_uv,Puv_v] = comparePitchDetection(filepath,1);

disp(['MSE error:' num2str(MSE)]);
disp(['Average pitch error (Hz):' num2str(avErr)]);
disp(['Gross Pitch Error (20%): ' num2str(GPE*100) ' %']);
disp(['P(voiced|unvoiced) = ' num2str(Pv_uv*100) '%']);
disp(['P(unvoiced|voiced) = ' num2str(Puv_v*100) '%']);
```

Fig. 21. test.m, This function prompts a message asking for a file name input to analyse/compare

```
% Execute this script and wait for the results

close all;
clc

Nfiles = 100; % Number of files
% Define the MSE vector and the pitch average error vector
MSE = zeros(1,Nfiles);
avErr = MSE;
GPE = MSE;
Pv_uv = MSE;
Puv_v = MSE;

h = waitbar(0,'Please wait...');

% Male .wav files
for i=1:Nfiles/2
    waitbar(i/Nfiles)
    if i<10
        aux = 0;
    else
        aux = '';
    end

    filepath = ['evaluation/pitch_db/rl0' num2str(aux) num2str(i) '.wav'];
    [MSE(i), avErr(i), GPE(i), Pv_uv(i), Puv_v(i)] = comparePitchDetection(filepath,0);
end

% Female .wav files
for i=1:Nfiles/2
    waitbar((50+i)/Nfiles)
    if i<10
        aux = 0;
    else
        aux = '';
    end

    filepath = ['evaluation/pitch_db/sb0' num2str(aux) num2str(i) '.wav'];
    [MSE(50+i), avErr(50+i), GPE(50+i), Pv_uv(50+i), Puv_v(50+i)] = ...
        comparePitchDetection(filepath,0);
end
```

Fig. 22. test.m, This function shows the pitch detector performance in terms of MSE and Pitch average error in Hz for all the .wav files with respect to the reference database

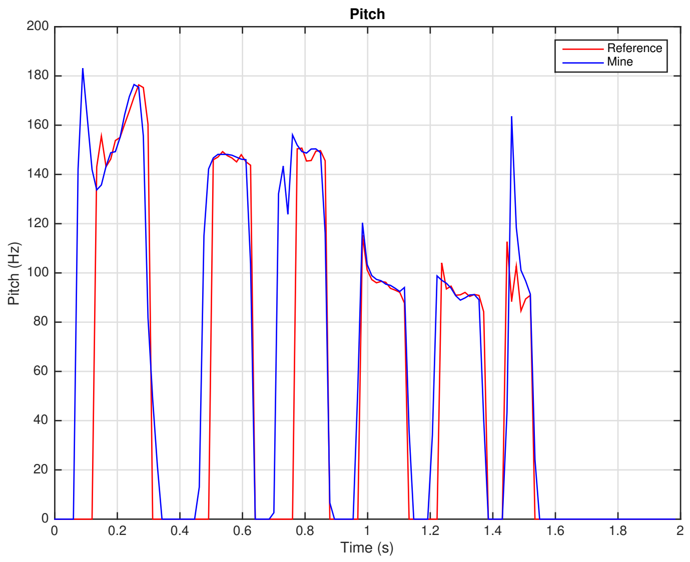

Fig. 23. Comparison between the results of the functions implemented and the reference values for the recording ’rl003.wav’

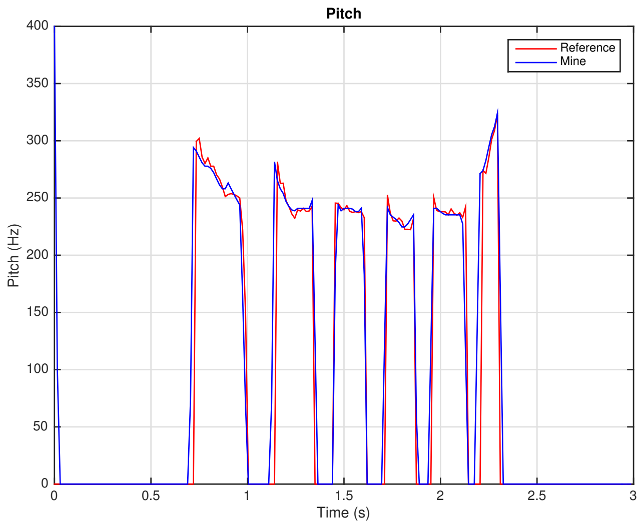

Fig. 24. Comparison between the results of the functions implemented and the reference values for the recording ’sb003.wav’

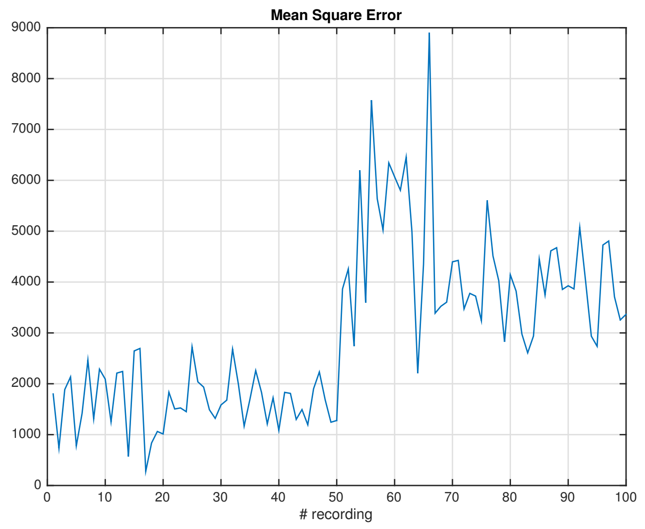

Fig. 25. Mean Square Error of the implemented pitch detection algorithm and the reference values. Note that for female recordings, larger errors are more frequently.

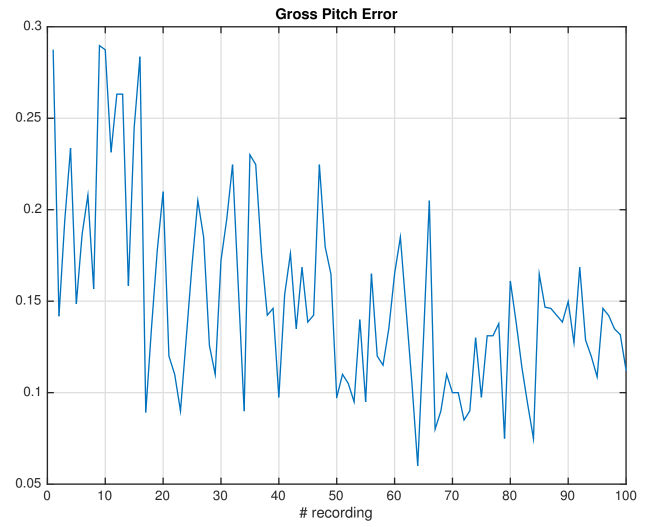

Fig. 26. This illustrates the Gross Pitch Error at 20% for all the recordings.

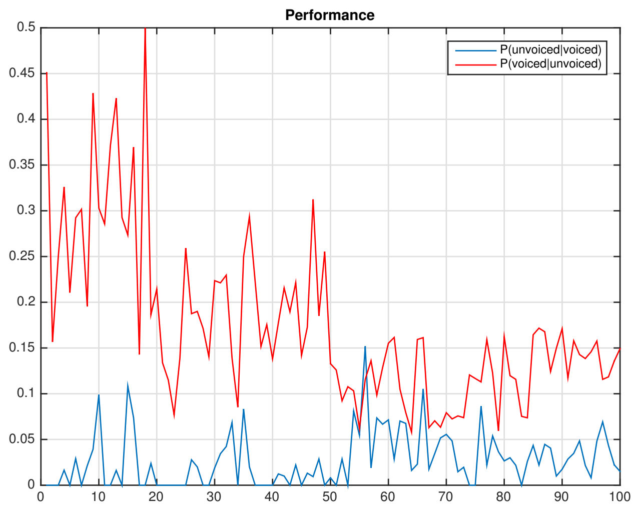

Fig. 27. This shows the $P(\text{measured} = voiced|reference = unvoiced)$ and the $P(\text{measured} = unvoiced|reference = voiced)$.