18-12-2015

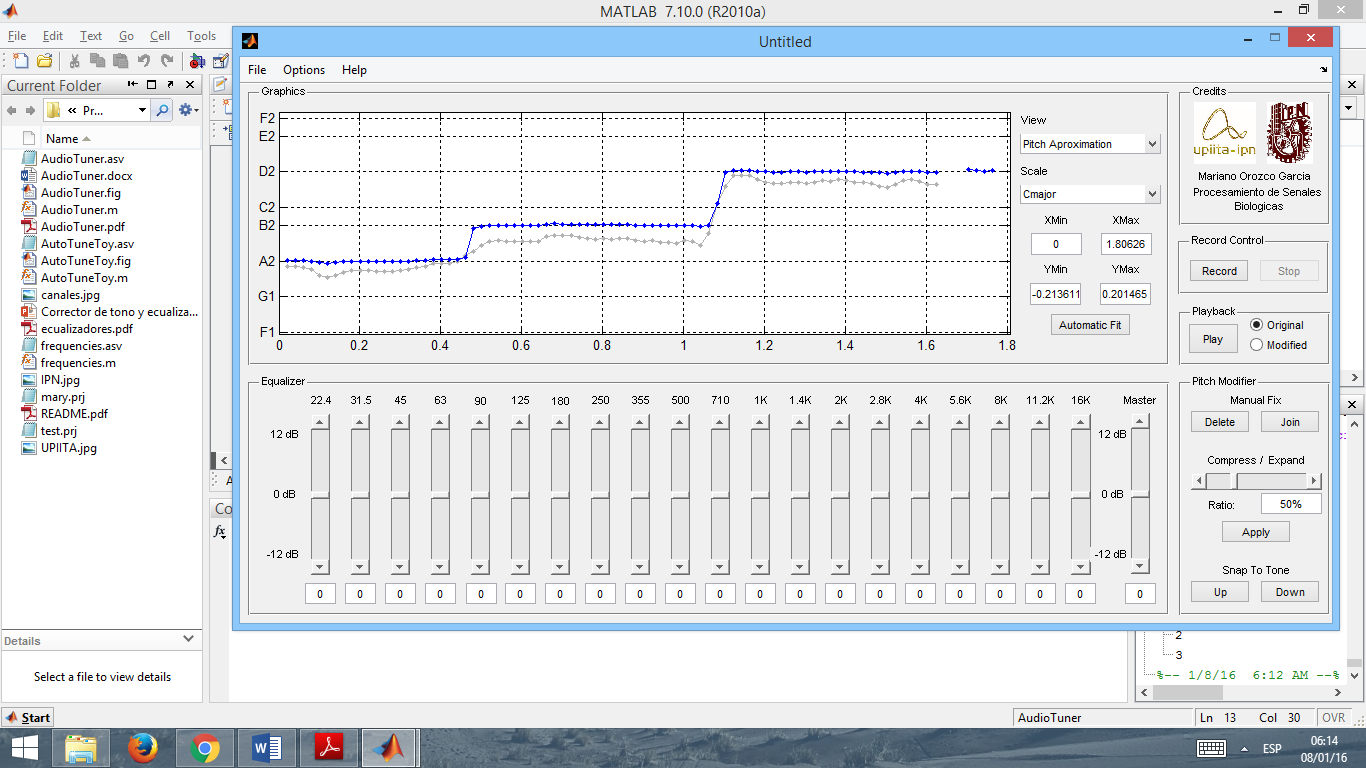
Mariano Orozco García

Procesamiento de señales biológicas, 3BM4

Tovar Corona Blanca

Audio Tuner

Pitch corrector

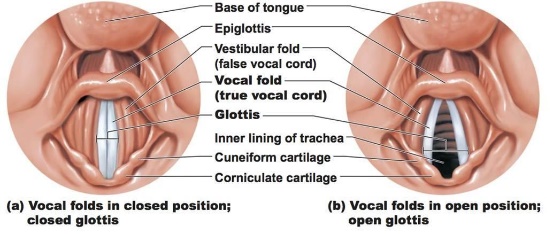


***Audio Tuner – Pitch corrector***

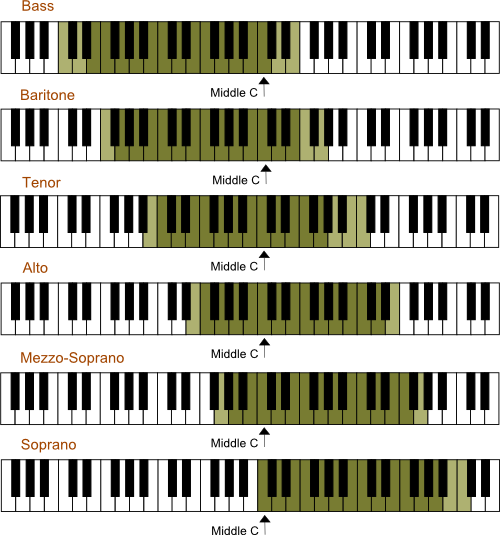
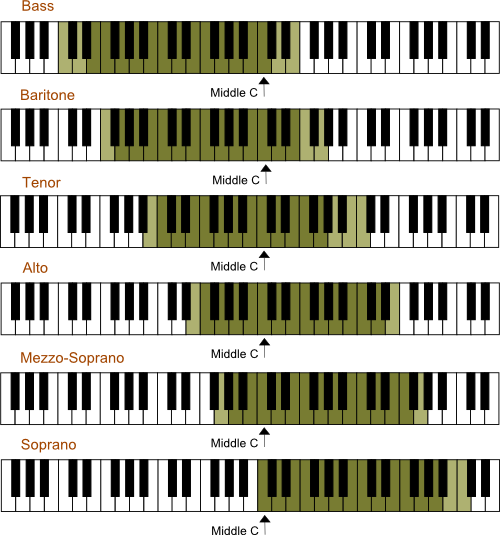
***INTRODUCTION:***

**Human Voice**

It is produced by the diaphragm action when it pushes air from the lungs through the vocal folds. This produces a periodic train of air that is shaped by the resonances of the vocal tract. The air make the vocal folds vibrate and this vibrations generate a sound wave that is a combination of several frequencies and their harmonics. The basic resonances, called vocal formants, can be changed by the action of the articulators to produce distinguishable voice sounds, like the vowel sounds.



In men and women different pitches can be archived. From bass to soprano voices the frequencies that the human voice can reach go from 82 Hz to 1056Hz.

**Pitch correction**

Pitch correction is an electronic effect that changes the intonation (highness or lowness in pitch) of an audio signal so that the pitches will be notes that correspond with the desired pitch of the song. Any pitch correction system first detects the pitch of an audio signal and then it calculates the desired change and modifies the audio signal.

The most common use of pitch correctors is to fix wrong intonation of notes sung by vocalists in popular music sound recordings. The use of pitch correction speeds up the recording process, because singers do not need to keep singing a song or vocal line and re-recording it until the pitches are correct. The pitch correction software can correct any pitch errors in the singing without the need for overdubbing or re-recording. However, it can also be used to fix intonation in recorded instrumental parts such as violin, cello or trumpet.

The use of pitch correction tools is controversial in music industry because it can make perfectly in-tune performances from a vocalist who is otherwise not skilled enough to give one. Because of this some artist make public that in their records or concerts no pitch corrector was used.

***OBJECTIVE:***

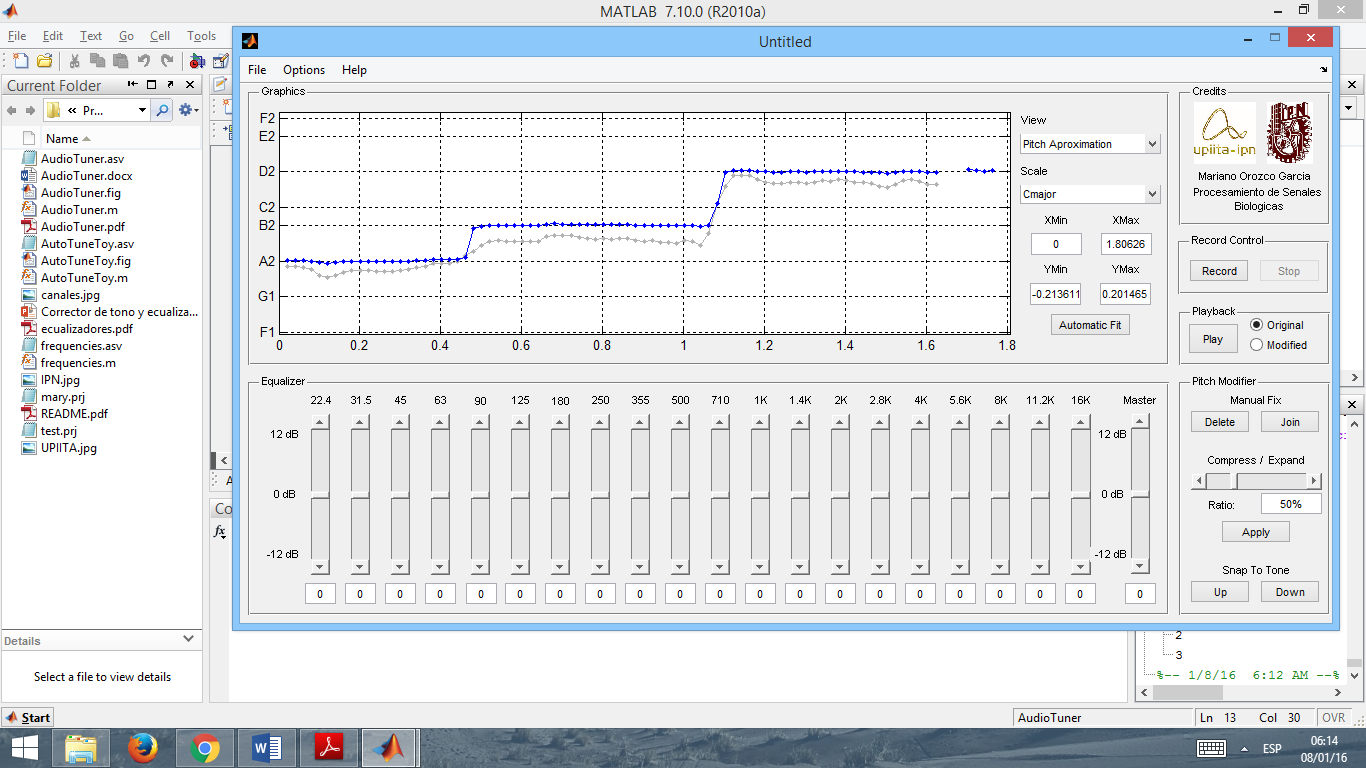
To implement a pitch corrector that add several effects to the human voice and adjust the original wave to the desired parameters in an interactive way.

***DEVELOPMENT:***

**Block Diagram**

**Implementation**

In order to implement this tool we will use a GUI to make it interactive and user-friendly.



***Figure1.*** *GUI used for the pitch corrector.*

As we can see the right panels are the ones that we will focus along with the graphics panel but we will omit the equalizer panel because it is beyond this particular job.

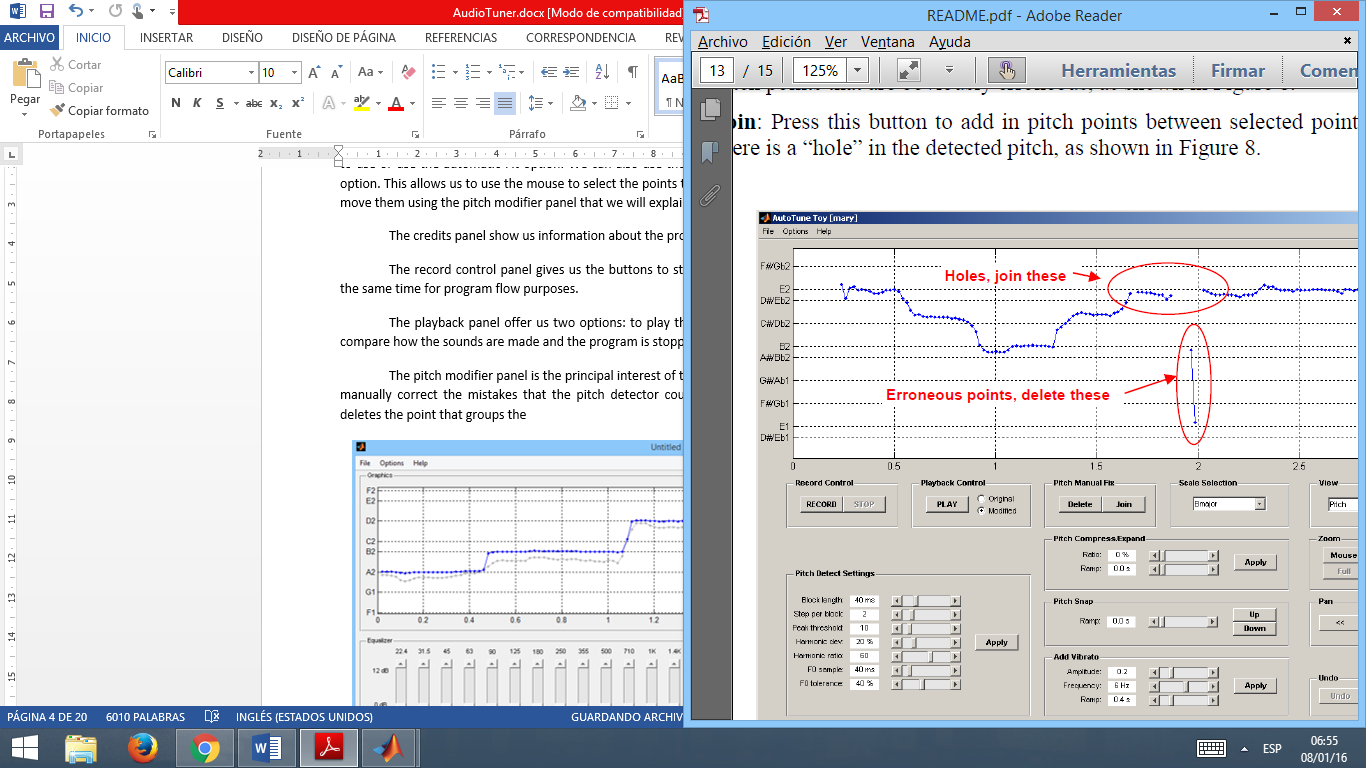
In the *graphics panel* we can choose between seen the pitch approximation view and the signal in the time domain, we can change the musical scale in which we want to work (mayor or minor of any key) and set the visualization window that we want to use or press the automatic Fit button. We can also use the graphics interactively in the pitch approximation view option. This allows us to use the mouse to select the points that are generated by the pitch detector algorithm and modify them using the pitch modifier panel that we will explain later in this section. To use the interactive plots we use two overlapped axes items in the GUI one to show the original signal in one and to move the items with the help of the mouse and the pitch corrector options in the other.

The *credits panel* show us information about the program.

The *record control* panel gives us the buttons to start recording or stop the record, only one is active at the same time for program flow purposes.

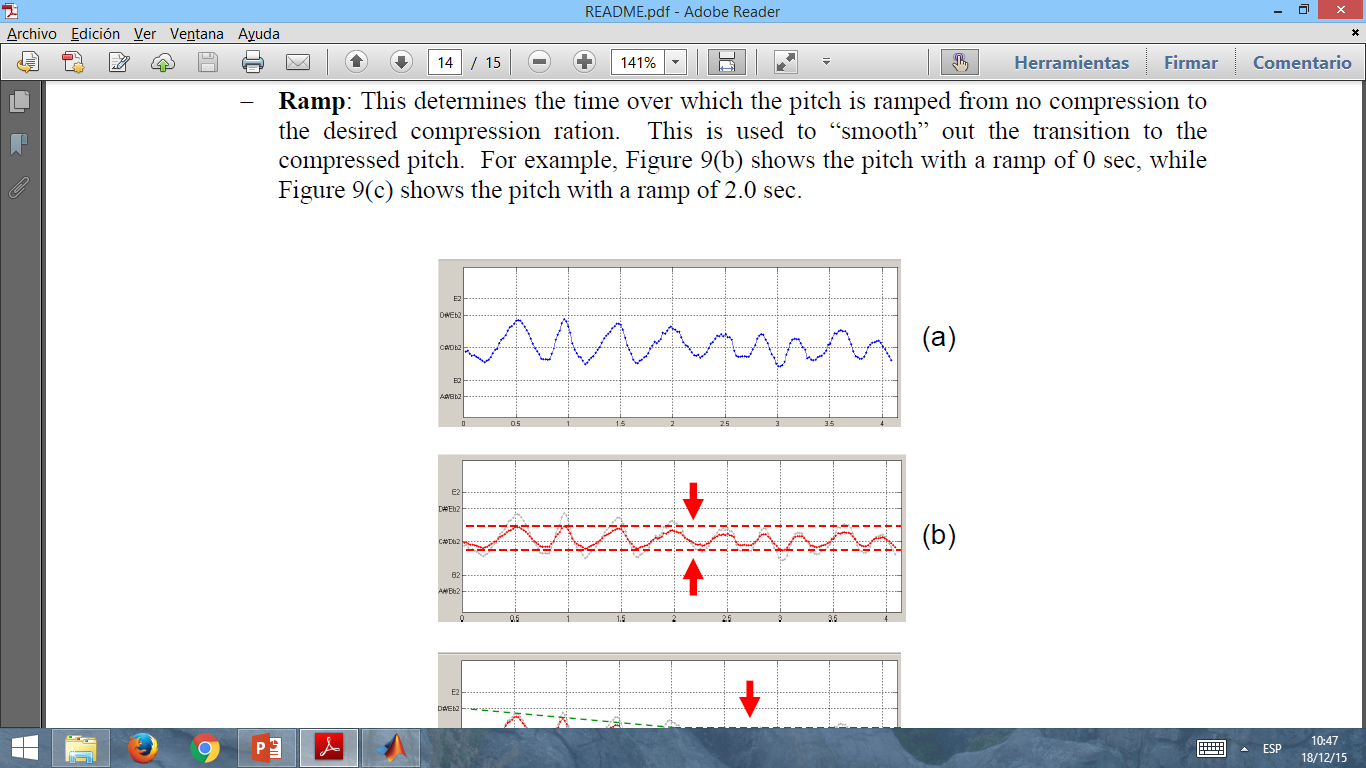
The *playback panel* offer us two options: to play the original record or the modified one. This is used to compare how the sounds are made and the program is stopped while the signal is being played.

The *pitch modifier panel* is the principal interest of this project. This includes the delete and join detected points in case to be necessary to manually correct the mistakes that the pitch detector could make. This is shown in figure 2.



***Figure2.*** *Use of delete and join buttons.*

The compress or expand option sets the compression rate (whose default is 50%) and modifies the signal when apply button is pressed. This compress or expand the selected points geometrically around the arithmetic media of the selected group. We can see an example in figure 3 of this process.

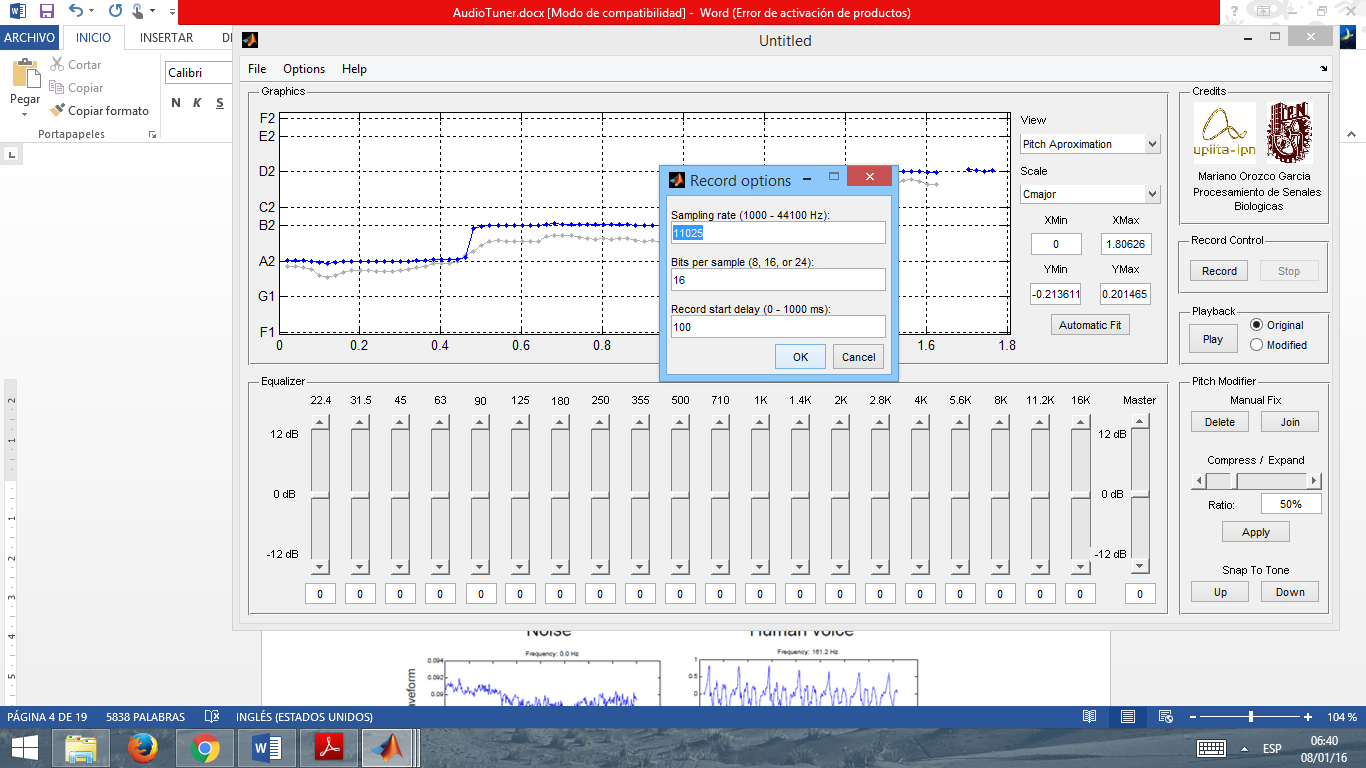


***Figure3.*** *Compression example.*

Finally up and down buttons in the snap option will place the arithmetic media of the selected group exactly in the tone that is above or below the current value and move all the selected points when accordingly. This will compose the desired frequency vector that will be used in the pitch corrector algorithm.

In the file menu we will find some other options to export and import audio or save the plots as images (not implemented yet). We can also save or load the entire project, all the values of the options as well as the original and modified signals (already implemented).

In the options menu we can modify the recording options and the pitch corrector options. Record options include: the sampling rate, the bits per sample and the start delay time. The start delay time is used because of the time the program need to open the microphone and actively start recording. Default values are shown in figure 4. Pitch corrector options will be explained in detail in algorithm explanation section.

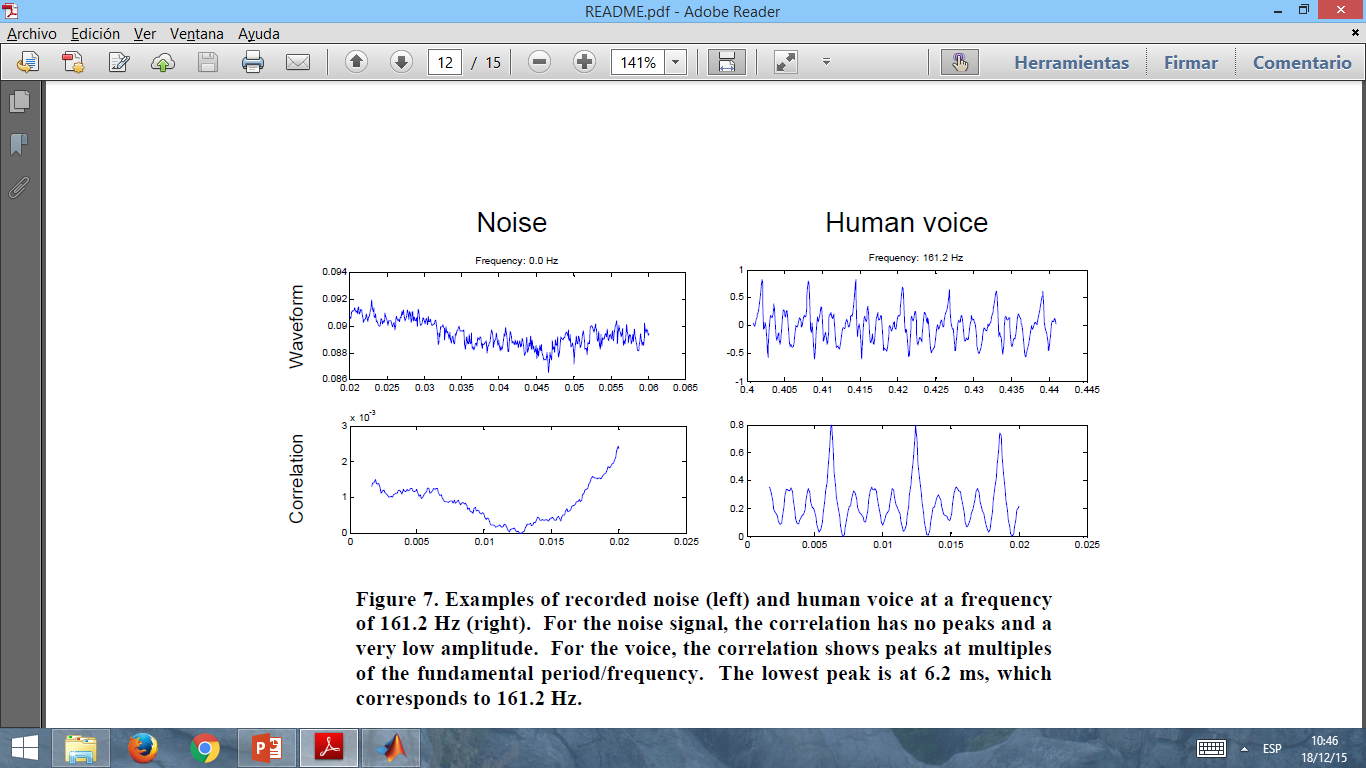


***Figure4.*** *Record options.*

In the help options we can find information about the GUI and the link to this file.

**Algorithm explanation**

The pitch detector algorithm works using an autocorrelation in time domain. First we separate de signals in blocks that will be analyzed with the correlation function and then with the help of an amplitude threshold we measure the frequency of the resulting signal. The components that have more amplitude that are minimum in the range where the pitch detector works and have harmonics that are also big in amplitude (with some tolerances) will be considered as the principal frequency.

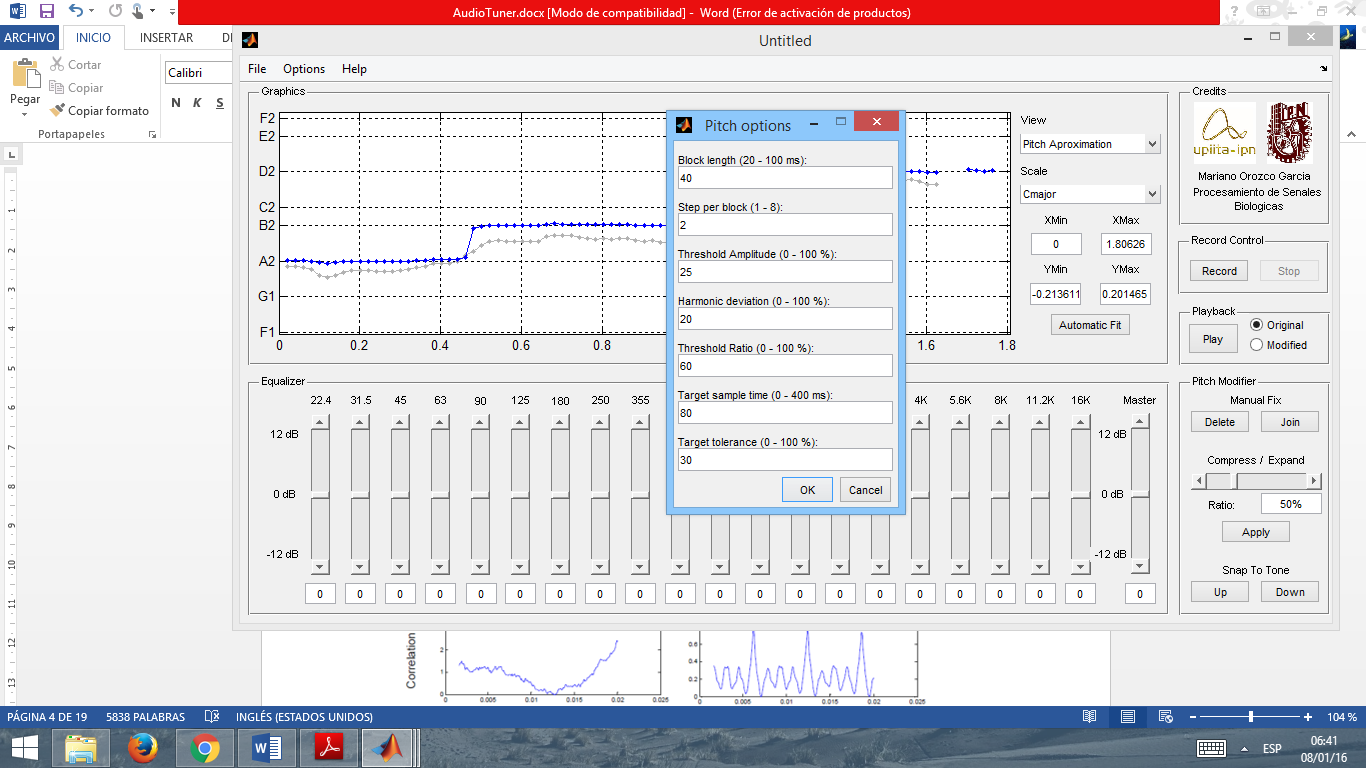


***Figure5.*** *Correlation of signals.*

The frequencies that we obtain in the block will be saved and then we will choose the next block by moving certain amount of samples (step). The algorithm is made in a way that the step is shorter than the block so the blocks overlap and this gave us a better understanding of how the principal frequency is acting along the time.

Finally this vector is averaged in sets. If the previous and the next frequency are close this is considered as the same note. If they differ we consider this as the next note. We have a target sample time for the groups that will appear in the graphics, this ones are obtained by averaging the principal frequencies that we found along this period of time.

This algorithm requires some parameters to work whose default values are shown in figure 6. Block length is the size in time of the block. The step per block is the time that will be skipped from the previous block start to the next one and should be shorter than the block length. The threshold amplitude is used to reduce noise and we will only include in the analysis those frequencies that are above the desired ratio in amplitude. Harmonic deviation and threshold ratio are parameters to detect and eliminate the harmonics of other frequencies in order to not consider them as the principal one. Target sample time is used to define the amount of time that will contain a desired frequency that is representative for that lapse. The target sample time should be bigger than the block length. Finally target tolerance defines if in that target sample time there is a representative frequency or not, this depends on how disperse or compact the data is.



***Figure6.*** *Pitch options*

The pitch corrector algorithm uses the ratio between the desired frequency and the original one to interpolate or extrapolate the wave in the time domain by compressing or expanding it and then average or add the missing points in order to maintain the same sample rate of the original signal.

***RESULTS:***

Human voice can be modified effectively and pitch correction can be clearly heard with a medium trained ear.

Tests suggest that maximum record time should be around 20 seconds since buffer is required to keep data. Time of processing and pitch algorithms get unpractical beyond this point.

The pitch detector have some mistakes and must be manually corrected in the parameters or even graphically in some cases. However this algorithm is quite stable for any voice (male or female) and the modifications that should be made manually are minimal.

The audio loses quality and can be perceived as more artificial when you modify the signal when the ratios between the desired and original frequency is very high or very low. This is caused for the approximations made in the interpolation or extrapolation of the resulting signals that modify and loose data with reference to the original one.

***PERSONAL CONCLUSIONS:***

It is possible to create a pitch corrector tool of the human voice that detects any voice. This algorithm even work with instruments or other sounds.

Frequencies have to be set and changed in the pitch detector algorithm for woman or men since they have different spectra. Or for more general (but less accurate) purposes, set a bandwidth that involves both woman and man.

Noise in the record can substantially affect the results of the pitch detector. A good audio record and a controlled ambient should be generated in order to get good results.

If the algorithm is implement in a dedicated software or a faster platform we can extend its purposes for longer records, with more definition or with some modifications we could even implement it for real-time pitch corrector system. This real-time can be only made if we know the pitch we want to reach before we start to record.

***REFERENCES:***

1. Jürgen Stutzki, Convolution, Autocorrelation, Cross-correlation, Power Spectrum: Fourier Transform and its Applications. Soummersemester 2007. Recovered from: [<http://hera.ph1.uni-koeln.de/~stutzki/teaching/FT_appl_2.pdf>].
2. R. Nave, *Vocal sound production,* Hyper Physics, recovered from: [<http://hyperphysics.phy-astr.gsu.edu/hbase/music/voice.html>].
3. *How the voice works,* American Academy of otolaryngology. Recovered from: [<http://www.entnet.org/content/how-voice-works>].
4. Tonya Reiman, *The human Voice – Pitch*, Body Language University, Recovered from: [<http://www.bodylanguageuniversity.com/public/203.cfm>].

***APPENDIX:***

**Code in Matlab:**

function varargout = AudioTuner(varargin)

% Initialization code - DO NOT EDIT

gui\_Singleton = 1;

gui\_State = struct('gui\_Name', mfilename, ...

'gui\_Singleton', gui\_Singleton, ...

'gui\_OpeningFcn', @AudioTuner\_OpeningFcn, ...

'gui\_OutputFcn', @AudioTuner\_OutputFcn, ...

'gui\_LayoutFcn', [] , ...

'gui\_Callback', []);

if nargin && ischar(varargin{1})

gui\_State.gui\_Callback = str2func(varargin{1});

end

if nargout

[varargout{1:nargout}] = gui\_mainfcn(gui\_State, varargin{:});

else

gui\_mainfcn(gui\_State, varargin{:});

end

% --- Opening and Output Functions

function AudioTuner\_OpeningFcn(hObject, ~, handles, varargin)

% This function has no output args, see OutputFcn.

% hObject handle to figure

% handles structure with handles and user data (see GUIDATA)

% varargin command line arguments to AudioTuner (see VARARGIN)

handles.output = hObject; % Choose default command line output

% Axes1 defaults

set(handles.axes1,'XTickLabel',{});

set(handles.axes1,'YTickLabel',{});

% Axes2 is just for mouse click input

set(handles.axes2,'XTick',[]);

set(handles.axes2,'YTick',[]);

set(handles.axes2,'XTickLabel',{});

set(handles.axes2,'YTickLabel',{});

set(handles.axes2,'ButtonDownFcn','AudioTuner(''mouseclick'',gcbo,[],guidata(gcbo))');

% Set sound

handles.sound.A = [];

handles.sound.A\_corrected = [];

handles.sound.Fs = [];

handles.sound.t = [];

handles.sound.f0 = [];

handles.sound.f0\_save = [];

handles.sound.f0\_corrected = [];

handles.sound.f0\_corrected\_save = [];

handles.sound.scale\_factor = [];

handles.sound.tcalc = [];

handles.sound.selected\_points = logical([]);

handles.sound.selected\_points\_save = logical([]);

% Set status

handles.status.isrecording = false;

handles.status.filename = '';

handles.status.Xlim = [];

handles.status.Ylim = [];

% Set recording default options

handles.record\_options.Fs = 11025; % Sampling frequency (Hz)

handles.record\_options.nbits = 16; % Bits of precision

handles.record\_options.initial\_trim = 0.1; % Always trim off the initial 0.1 sec

% Pitch detection defaults

handles.pitch\_options.Fmin = 50; % Min frequency to search

handles.pitch\_options.Fmax = 600; % Max frequency to search

handles.pitch\_options.block\_length = 0.04; % Length of each chuck to analyze for frequency

handles.pitch\_options.step\_per\_block = 2; % What fraction of the block width to step for each pitch detect

handles.pitch\_options.threshold\_amp = 0.25; % Threshold of autocorr to be called a peak

handles.pitch\_options.harmonic\_deviation = 0.2; % Max deviation from multiple fundamental to be considered a harmonic (0.2 = +/- 20%)

handles.pitch\_options.threshold\_ratio = 0.6; % Max ratio in heights of autocorr allowed between fundamental and harmonics

handles.pitch\_options.target\_sample\_time = 0.08; % Over what time interval to sample previous freq's to determine next target f0

handles.pitch\_options.target\_tol = 0.3; % Max tolerance from f0 target (0.2 = +/-20%)

handles.pitch\_options.compress\_ratio = 0.5; % Ratio to compress tone (0-100 %)

% Scale options

handles.scale\_options.scale = 'Cmajor'; % Scale for pitch correction

handles.scale\_options.indices = [];

handles.scale\_options.freqs = [];

handles.scale\_options.notes = {};

handles.scale\_options.fund\_index = [];

[handles.scale\_options.indices,...

handles.scale\_options.freqs,...

handles.scale\_options.notes,...

handles.scale\_options.fund\_index] = ...

get\_scale(handles.scale\_options.scale);

% Other options

handles.record\_obj = [];

% Update handles structure

handles = update\_GUI(handles);

guidata(hObject, handles);

function varargout = AudioTuner\_OutputFcn(~, ~, handles)

% varargout cell array for returning output args (see VARARGOUT);

% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure

varargout{1} = handles.output;

function axes3\_CreateFcn(hObject, ~, ~)

imshow(imread('UPIITA.jpg'));

function axes4\_CreateFcn(hObject, ~, ~)

imshow(imread('IPN.jpg'));

% --- Equalizer Create Sliders

function B1\_CreateFcn(hObject, ~, ~)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

function B2\_CreateFcn(hObject, ~, ~)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

function B3\_CreateFcn(hObject, ~, ~)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

function B4\_CreateFcn(hObject, ~, ~)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

function B5\_CreateFcn(hObject, ~, ~)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

function B6\_CreateFcn(hObject, ~, ~)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

function B7\_CreateFcn(hObject, ~, ~)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

function B8\_CreateFcn(hObject, ~, ~)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

function B9\_CreateFcn(hObject, ~, ~)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

function B10\_CreateFcn(hObject, ~, ~)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

function B11\_CreateFcn(hObject, ~, ~)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

function B12\_CreateFcn(hObject, ~, ~)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

function B13\_CreateFcn(hObject, ~, ~)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

function B14\_CreateFcn(hObject, ~, ~)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

function B15\_CreateFcn(hObject, ~, ~)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

function B16\_CreateFcn(hObject, ~, ~)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

function B17\_CreateFcn(hObject, ~, ~)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

function B18\_CreateFcn(hObject, ~, ~)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

function B19\_CreateFcn(hObject, ~, ~)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

function B20\_CreateFcn(hObject, ~, ~)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

function Volume\_CreateFcn(hObject, ~, ~)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

% --- Equalizer Create Text Box For values

function V1\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function V2\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function V3\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function V4\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function V5\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function V6\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function V7\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function V8\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function V9\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function V10\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function V11\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function V12\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function V13\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function V14\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function V15\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function V16\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function V17\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function V18\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function V19\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function V20\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function VVolume\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

% --- Graphics Create Options

function view\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function Scale\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function XMin\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function XMax\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function YMin\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function YMax\_CreateFcn(hObject, ~, ~)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

% --- Pitch Create Functions

function CompExp\_CreateFcn(hObject, ~, ~)

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

function VCompExp\_CreateFcn(hObject, ~, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

% --- Equalizer Callback Sliders

function B1\_Callback(hObject, ~, handles)

set(handles.V1,'String',get(hObject,'Value'))

guidata(hObject, handles);

function B2\_Callback(hObject, ~, handles)

set(handles.V2,'String',get(hObject,'Value'))

guidata(hObject, handles);

function B3\_Callback(hObject, ~, handles)

set(handles.V3,'String',get(hObject,'Value'))

guidata(hObject, handles);

function B4\_Callback(hObject, ~, handles)

set(handles.V4,'String',get(hObject,'Value'))

guidata(hObject, handles);

function B5\_Callback(hObject, ~, handles)

set(handles.V5,'String',get(hObject,'Value'))

guidata(hObject, handles);

function B6\_Callback(hObject, ~, handles)

set(handles.V6,'String',get(hObject,'Value'))

guidata(hObject, handles);

function B7\_Callback(hObject, ~, handles)

set(handles.V7,'String',get(hObject,'Value'))

guidata(hObject, handles);

function B8\_Callback(hObject, ~, handles)

set(handles.V8,'String',get(hObject,'Value'))

guidata(hObject, handles);

function B9\_Callback(hObject, ~, handles)

set(handles.V9,'String',get(hObject,'Value'))

guidata(hObject, handles);

function B10\_Callback(hObject, ~, handles)

set(handles.V10,'String',get(hObject,'Value'))

guidata(hObject, handles);

function B11\_Callback(hObject, ~, handles)

set(handles.V11,'String',get(hObject,'Value'))

guidata(hObject, handles);

function B12\_Callback(hObject, ~, handles)

set(handles.V12,'String',get(hObject,'Value'))

guidata(hObject, handles);

function B13\_Callback(hObject, ~, handles)

set(handles.V13,'String',get(hObject,'Value'))

guidata(hObject, handles);

function B14\_Callback(hObject, ~, handles)

set(handles.V14,'String',get(hObject,'Value'))

guidata(hObject, handles);

function B15\_Callback(hObject, ~, handles)

set(handles.V15,'String',get(hObject,'Value'))

guidata(hObject, handles);

function B16\_Callback(hObject, ~, handles)

set(handles.V16,'String',get(hObject,'Value'))

guidata(hObject, handles);

function B17\_Callback(hObject, ~, handles)

set(handles.V17,'String',get(hObject,'Value'))

guidata(hObject, handles);

function B18\_Callback(hObject, ~, handles)

set(handles.V18,'String',get(hObject,'Value'))

guidata(hObject, handles);

function B19\_Callback(hObject, ~, handles)

set(handles.V19,'String',get(hObject,'Value'))

guidata(hObject, handles);

function B20\_Callback(hObject, ~, handles)

set(handles.V20,'String',get(hObject,'Value'))

guidata(hObject, handles);

function Volume\_Callback(hObject, ~, handles)

set(handles.VVolume,'String',get(hObject,'Value'))

guidata(hObject, handles);

% --- Graphics Callback

function view\_Callback(hObject, ~, handles)

if isempty(handles.sound.A\_corrected)

handles = run\_pitch\_correction(handles);

end

handles = update\_plot(handles);

guidata(hObject, handles);

function Scale\_Callback(hObject, ~, handles)

scale = get(handles.Scale,'String');

scale = scale{get(handles.Scale,'Value')};

if ~strcmp(scale,'CUSTOM')

handles.scale\_options.scale = scale;

[handles.scale\_options.indices,...

handles.scale\_options.freqs,...

handles.scale\_options.notes,...

handles.scale\_options.fund\_index] = ...

get\_scale(handles.scale\_options.scale);

end

handles = update\_plot(handles);

guidata(hObject, handles);

function XLim\_Callback(hObject, ~, handles)

if ~isempty(handles.sound.t)

mi = max(str2num(get(handles.XMin,'String')),min(handles.sound.t));

ma = min(str2num(get(handles.XMax,'String')),max(handles.sound.t));

if mi>=ma

mi = min(handles.sound.t);

ma = max(handles.sound.t);

end

handles.status.Xlim = [mi,ma];

set(handles.XMin,'String',mi);

set(handles.XMax,'String',ma);

guidata(hObject, handles);

else

disp('error');

end

function YLim\_Callback(hObject, ~, handles)

if ~isempty(handles.sound.t)

mi = max(str2num(get(handles.YMin,'String')),min(handles.sound.A));

ma = min(str2num(get(handles.YMax,'String')),max(handles.sound.A));

if mi>ma

mi = min(handles.sound.A);

ma = max(handles.sound.A);

end

handles.status.Ylim = [mi,ma];

set(handles.YMin,'String',mi);

set(handles.YMax,'String',ma);

guidata(hObject, handles);

else

disp('error');

end

function AutomaticFit\_Callback(hObject, ~, handles)

handles.status.Xlim = [min(handles.sound.t),max(handles.sound.t)];

handles.status.Ylim = [min(handles.sound.A),max(handles.sound.A)];

handles = update\_plot(handles);

guidata(hObject, handles);

% --- Record Control

function Record\_Callback(hObject, ~, handles)

try

handles.record\_obj = audiorecorder(handles.record\_options.Fs,...

handles.record\_options.nbits,1);

record(handles.record\_obj);

handles.status.isrecording = true;

handles = update\_GUI(handles);

guidata(hObject, handles);

catch

h = errordlg('Error starting audio input device! Check sound card and microphone!','ERROR');

waitfor(h);

end

function Stop\_Callback(hObject, ~, handles)

if handles.status.isrecording

% Stop recording

stop(handles.record\_obj);

handles.status.isrecording = false;

handles = update\_GUI(handles);

% Save values

handles.sound.A = getaudiodata(handles.record\_obj,'single');

Nstart = round(handles.record\_options.initial\_trim\*...

handles.record\_options.Fs); % Trim initial samples

handles.sound.A = handles.sound.A(Nstart:end);

handles.sound.Fs = handles.record\_options.Fs;

handles.sound.A\_corrected = handles.sound.A;

handles.sound.t = (0:length(handles.sound.A)-1)./handles.record\_options.Fs;

% Clear old frequency data

handles.sound.f0 = [];

handles.sound.f0\_save = [];

handles.sound.f0\_corrected = [];

handles.sound.f0\_corrected\_save = [];

handles.sound.selected\_points = logical([]);

handles.sound.selected\_points\_save = logical([]);

% Clear plot limits

handles.status.Xlim = [];

handles.status.Ylim = [];

% Update

handles = run\_pitch\_detection(handles);

set(handles.XMin,'String',min(handles.sound.t));

set(handles.XMax,'String',max(handles.sound.t));

handles.status.Xlim = [min(handles.sound.t),max(handles.sound.t)];

set(handles.YMin,'String',min(handles.sound.A));

set(handles.YMax,'String',max(handles.sound.A));

handles.status.Ylim = [min(handles.sound.A),max(handles.sound.A)];

set(handles.view,'Value',3)

handles = update\_plot(handles);

Scale\_Callback(hObject, [], handles)

handles = update\_GUI(handles);

guidata(hObject, handles);

end

% --- Playback

function Play\_Callback(hObject, ~, handles)

% Decide which to play

choice = get(get(handles.PlayChoice,'SelectedObject'),'String');

choice = strcmp(choice,'Original');

if choice

if isempty(handles.sound.A)

return

end

A = handles.sound.A;

else

if isempty(handles.sound.A\_corrected)

handles = run\_pitch\_correction(handles);

end

A = handles.sound.A\_corrected;

end

wavplay(A,handles.sound.Fs);

axes(handles.axes2);

% --- Pitch Modifier

function Delete\_Callback(hObject, ~, handles)

if isempty(handles.sound.f0\_corrected) || ~any(handles.sound.selected\_points)

return

end

% Save for undoing

handles.sound.f0\_save(end+1,:) = handles.sound.f0;

handles.sound.f0\_corrected\_save(end+1,:) = handles.sound.f0\_corrected;

handles.sound.selected\_points\_save(end+1,:) = handles.sound.selected\_points;

% Pull out the frequency and time of the selected points

handles.sound.f0\_corrected(handles.sound.selected\_points) = 0;

handles.sound.f0(handles.sound.selected\_points) = 0;

handles.sound.selected\_points = false(size(handles.sound.f0));

handles.sound.A\_corrected = [];

handles = update\_plot(handles);

Scale\_Callback(hObject, [], handles)

handles = update\_GUI(handles);

guidata(hObject, handles);

function Join\_Callback(hObject, ~, handles)

if isempty(handles.sound.f0\_corrected) || ~any(handles.sound.selected\_points)

return

end

% Save for undoing

handles.sound.f0\_save(end+1,:) = handles.sound.f0;

handles.sound.f0\_corrected\_save(end+1,:) = handles.sound.f0\_corrected;

handles.sound.selected\_points\_save(end+1,:) = handles.sound.selected\_points;

% Pull out the frequency and time of the selected points

any\_changes = false;

I = find(handles.sound.selected\_points);

f0 = handles.sound.f0;

for n = 1:length(I)-1

if I(n+1)-I(n) > 1

if f0(I(n)) > 1 && f0(I(n+1)) > 1 && all(f0(I(n)+1:I(n+1)-1) < 1)

f0(I(n):I(n+1)) = logspace(log10(f0(I(n))),log10(f0(I(n+1))),length(I(n):I(n+1)));

handles.sound.selected\_points(I(n):I(n+1)) = true;

any\_changes = true;

end

end

end

handles.sound.f0 = f0;

f0 = handles.sound.f0\_corrected;

for n = 1:length(I)-1

if I(n+1)-I(n) > 1

if f0(I(n)) > 1 && f0(I(n+1)) > 1 && all(f0(I(n)+1:I(n+1)-1) < 1)

f0(I(n):I(n+1)) = logspace(log10(f0(I(n))),log10(f0(I(n+1))),length(I(n):I(n+1)));

any\_changes = true;

end

end

end

if ~any\_changes

return

end

handles.sound.f0\_corrected = f0;

handles.sound.A\_corrected = [];

handles = update\_plot(handles);

Scale\_Callback(hObject, [], handles)

handles = update\_GUI(handles);

guidata(hObject, handles);

function CompExp\_Callback(hObject, ~, handles)

set(handles.VCompExp,'String',[num2str(get(hObject,'Value')),'%'])

handles.pitch\_options.compress\_ratio = get(hObject,'Value')/100;

guidata(hObject, handles);

function Apply\_Callback(hObject, ~, handles)

if isempty(handles.sound.f0\_corrected) || ~any(handles.sound.selected\_points)

return

end

% Pull out the frequency and time of the selected points

f0 = handles.sound.f0\_corrected(handles.sound.selected\_points);

I = find(handles.sound.selected\_points);

t = handles.sound.tcalc(I(1):I(end));

t = t-t(1);

% Come up with a envelope for compressing that goes from 1 down to the desired compression factor over the desired time interval

factor = ones(size(t)).\*handles.pitch\_options.compress\_ratio;

factor = factor(handles.sound.selected\_points(I(1):I(end)));

% Scale the deviations around the mean in a log sense

f0 = 10.^(mean(log10(f0)) + factor.\*(log10(f0)-mean(log10(f0))));

handles.sound.f0\_corrected\_save(end+1,:) = handles.sound.f0\_corrected;

handles.sound.f0\_save(end+1,:) = handles.sound.f0;

handles.sound.selected\_points\_save(end+1,:) = handles.sound.selected\_points;

handles.sound.f0\_corrected(handles.sound.selected\_points) = f0;

handles.sound.A\_corrected = [];

handles = update\_plot(handles);

Scale\_Callback(hObject, [], handles);

handles = update\_GUI(handles);

guidata(hObject, handles);

function Snap\_Callback(hObject, ~, handles)

if isempty(handles.sound.f0\_corrected) || ~any(handles.sound.selected\_points) ||...

isempty(handles.scale\_options.freqs);

return

end

snap\_direction = get(gcbo,'String');

% Pull out the frequency and time of the selected points

I = find(handles.sound.selected\_points);

sp = handles.sound.selected\_points(I(1):I(end));

f0 = handles.sound.f0\_corrected(I(1):I(end));

t = handles.sound.tcalc(I(1):I(end));

t = t-t(1);

where = 1;

tmp = f0(where:end);

mean\_f0 = 10^mean(log10(tmp(sp(where:end))));

if strcmp(snap\_direction,'Down')

Iu = find(handles.scale\_options.freqs < 0.99\*mean\_f0);

mean\_new = max(handles.scale\_options.freqs(Iu));

elseif strcmp(snap\_direction,'Up')

Iu = find(handles.scale\_options.freqs > 1.01\*mean\_f0);

mean\_new = min(handles.scale\_options.freqs(Iu));

end

factor = mean\_new/mean\_f0.\*ones(size(t));

if where > 1

factor(1:where) = linspace(1,factor(end),where);

end

% Scale the deviations around the mean in a log sense

f0 = f0.\*factor;

handles.sound.f0\_corrected\_save(end+1,:) = handles.sound.f0\_corrected;

handles.sound.f0\_save(end+1,:) = handles.sound.f0;

handles.sound.selected\_points\_save(end+1,:) = handles.sound.selected\_points;

handles.sound.f0\_corrected(handles.sound.selected\_points) = f0(sp);

handles.sound.A\_corrected = [];

handles = update\_plot(handles);

Scale\_Callback(hObject, [], handles);

handles = update\_GUI(handles);

guidata(hObject, handles);

% --- Menu

function File\_Callback(hObject, ~, handles)

% File Menu

function Save\_Callback(hObject, ~, handles)

% Read data

data.sound = handles.sound;

data.status = handles.status;

data.record\_options = handles.record\_options;

data.pitch\_options = handles.pitch\_options;

% Verify if the file has a name

if ~isempty(handles.status.filename)

FilterSpec = handles.status.filename;

else

FilterSpec = 'untitled.prj';

end

% Choose path and name for file then save

[FileName,PathName,FilterIndex] = uiputfile(FilterSpec,'Save');

if ischar(FileName)

handles.status.filename = fullfile(PathName,FileName);

data.status.filename = fullfile(PathName,FileName);

save(fullfile(PathName,FileName), 'data','-mat');

end

guidata(hObject, handles);

function Load\_Callback(hObject, ~, handles)

% Warning not to loose current work

button = questdlg('Any changes since last save will be lost! Do you want to continue?',...

'Confirm load','Yes','No','Yes');

if strcmp(button,'No')

return

end

% Verify if the file has a name

if ~isempty(handles.status.filename)

FilterSpec = handles.status.filename;

else

FilterSpec = '\*.prj';

end

% Choose path and name for file then load

[FileName,PathName,FilterIndex] = uigetfile(FilterSpec,'Load');

if ischar(FileName)

load('-mat',fullfile(PathName,FileName));

handles.status.filename = fullfile(PathName,FileName);

handles.sound = data.sound;

handles.status = data.status;

handles.record\_options = data.record\_options;

handles.pitch\_options = data.pitch\_options;

% handles.scale\_options = data.scale\_options;

% handles.equalizer\_options = data.equalizer\_options;

handles = update\_plot(handles);

guidata(hObject, handles);

end

function Export\_Callback(hObject, ~, handles)

% Export Submenu

function Image\_Callback(hObject, ~, handles)

% Image

function Audio\_Callback(hObject, ~, handles)

% Audio

function Import\_Callback(hObject, ~, handles)

% Import Audio

function Options\_Callback(hObject, ~, handles)

% Options Menu

function RecordOptions\_Callback(hObject, ~, handles)

% Read actual values:

Fs = handles.record\_options.Fs;

nbits = handles.record\_options.nbits;

initial\_trim = handles.record\_options.initial\_trim\*1000;

% Create input dialog box

prompt = {

'Sampling rate (1000 - 44100 Hz):'

'Bits per sample (8, 16, or 24):'

'Record start delay (0 - 1000 ms):'

};

dlg\_title = 'Record options';

num\_lines = [1 35];

def = {

num2str(Fs)

num2str(nbits)

num2str(initial\_trim)

};

answer = inputdlg(prompt,dlg\_title,num\_lines,def);

% Save values if they are correct

empty\_error = false;

input\_error = false;

if ~isempty(answer)

if ~isempty(answer{1})

temp = round(str2num(answer{1}));

if temp < 1000 || temp > 44100

input\_error = true;

else

Fs = temp;

end

else

empty\_error = true;

end

if ~isempty(answer{2})

temp = round(str2num(answer{2}));

if ~ismember(temp,[8,16,24])

input\_error = true;

else

nbits = temp;

end

else

empty\_error = true;

end

if ~isempty(answer{3})

temp = round(str2num(answer{3}));

if temp < 0 || temp > 1000

input\_error = true;

else

initial\_trim = temp;

end

else

empty\_error = true;

end

if empty\_error

hwarn = warndlg('One or more values are empty, previous values are used!',...

'WARNING');

waitfor(hwarn);

end

if input\_error

hwarn = warndlg('One or more values out of range, previous values are used!',...

'WARNING');

waitfor(hwarn);

end

% Save values

handles.record\_options.Fs = Fs;

handles.record\_options.nbits = nbits;

handles.record\_options.initial\_trim = initial\_trim/1000;

% Update state of GUI

handles = update\_plot(handles);

guidata(hObject, handles);

end

function PitchOptions\_Callback(hObject, ~, handles)

% Read actual values:

block\_length = handles.pitch\_options.block\_length\*1000;

step\_per\_block = handles.pitch\_options.step\_per\_block;

treshold\_amp = handles.pitch\_options.threshold\_amp\*100;

harmonic\_deviation = handles.pitch\_options.harmonic\_deviation\*100;

threshold\_ratio = handles.pitch\_options.threshold\_ratio\*100;

taget\_sample\_time = handles.pitch\_options.target\_sample\_time\*1000;

target\_tol = handles.pitch\_options.target\_tol\*100;

% Create input dialog box

prompt = {

'Block length (20 - 100 ms):'

'Step per block (1 - 8):'

'Threshold Amplitude (0 - 100 %):'

'Harmonic deviation (0 - 100 %):'

'Threshold Ratio (0 - 100 %):'

'Target sample time (0 - 400 ms):'

'Target tolerance (0 - 100 %):'

};

dlg\_title = 'Pitch options';

num\_lines = [1 35];

def = {

num2str(block\_length)

num2str(step\_per\_block)

num2str(treshold\_amp)

num2str(harmonic\_deviation)

num2str(threshold\_ratio)

num2str(taget\_sample\_time)

num2str(target\_tol)

};

answer = inputdlg(prompt,dlg\_title,num\_lines,def);

% Save values if they are correct

empty\_error = false;

input\_error = false;

if ~isempty(answer)

if ~isempty(answer{1})

temp = round(str2num(answer{1}));

if temp < 20 || temp > 100

input\_error = true;

else

block\_length = temp;

end

else

empty\_error = true;

end

if ~isempty(answer{2})

temp = round(str2num(answer{2}));

if temp < 1 || temp > 8

input\_error = true;

else

step\_per\_block = temp;

end

else

empty\_error = true;

end

if ~isempty(answer{3})

temp = round(str2num(answer{3}));

if temp < 0 || temp > 100

input\_error = true;

else

treshold\_amp = temp;

end

else

empty\_error = true;

end

if ~isempty(answer{4})

temp = round(str2num(answer{4}));

if temp < 0 || temp > 100

input\_error = true;

else

harmonic\_deviation = temp;

end

else

empty\_error = true;

end

if ~isempty(answer{5})

temp = round(str2num(answer{5}));

if temp < 0 || temp > 100

input\_error = true;

else

threshold\_ratio = temp;

end

else

empty\_error = true;

end

if ~isempty(answer{6})

temp = round(str2num(answer{6}));

if temp < 0 || temp > 400

input\_error = true;

else

taget\_sample\_time = temp;

end

else

empty\_error = true;

end

if ~isempty(answer{7})

temp = round(str2num(answer{7}));

if temp < 0 || temp > 100

input\_error = true;

else

target\_tol = temp;

end

else

empty\_error = true;

end

if empty\_error

hwarn = warndlg('One or more values are empty, previous values are used!',...

'WARNING');

waitfor(hwarn);

end

if input\_error

hwarn = warndlg('One or more values out of range, previous values are used!',...

'WARNING');

waitfor(hwarn);

end

% Save values

handles.pitch\_options.block\_length = block\_length/1000;

handles.pitch\_options.step\_per\_block = step\_per\_block;

handles.pitch\_options.threshold\_amp = treshold\_amp/100;

handles.pitch\_options.harmonic\_deviation = harmonic\_deviation/100;

handles.pitch\_options.threshold\_ratio = threshold\_ratio/100;

handles.pitch\_options.target\_sample\_time = taget\_sample\_time/1000;

handles.pitch\_options.target\_tol = target\_tol/100;

% Update state of GUI

handles = run\_pitch\_detection(handles);

handles = update\_plot(handles);

guidata(hObject, handles);

end

function Help\_Callback(hObject, ~, handles)

% Help Menu

function About\_Callback(hObject, ~, handles)

h = msgbox({

' Audio Tuner'

''

' This program can add effects to voice records'

' using a pitch corrector and an equalizer.'

''

' Created by: Mariano Orozco Garcia'

' morozcog1101@alumno.ipn.mx'

},'About');

waitfor(h);

function HowTo\_Callback(hObject, ~, handles)

try

open('AudioTuner.pdf');

catch

h = errordlg({

'Cannot open Audio Tuner.pdf.'

'Please locate and open file manually for instructions'

},'ERROR OPENING HELP FILE');

waitfor(h)

end

% --- Pitch Functions

function handles = run\_pitch\_detection(handles)

% Runs only if theres a sound clip

if isempty(handles.sound.A)

return

end

% Get variables out of the handles structure

block\_length = handles.pitch\_options.block\_length;

step\_per\_block = handles.pitch\_options.step\_per\_block;

Fmin = handles.pitch\_options.Fmin;

Fmax = handles.pitch\_options.Fmax;

target\_sample\_time = handles.pitch\_options.target\_sample\_time;

target\_tol = handles.pitch\_options.target\_tol;

harmonic\_deviation = handles.pitch\_options.harmonic\_deviation;

threshold\_ratio = handles.pitch\_options.threshold\_ratio;

threshold\_amp = handles.pitch\_options.threshold\_amp;

A = handles.sound.A;

t = handles.sound.t;

Fs = handles.sound.Fs;

% Variables for size of analysis

block = step\_per\_block\*round(block\_length\*Fs/step\_per\_block); % Size of each block to find pitch

step = block/step\_per\_block; % Step (blocks are 4 times larger than "steps"

N = floor((length(A)-block)/step); % Number of frequency computations

% Initialize variables for storing results

f0 = zeros(1,N); % Initialize vector for storing frequencies

tcalc = zeros(1,N); % The time at which that frequency calculation is valid

f0\_target = []; % For keeping track of the target for the next calculation

f0\_samples = floor(target\_sample\_time/(step/Fs)); % Figure out how many f0 samples there will be

% Waitbar

hwait = waitbar(0,'Determining pitch...');

set(hwait,'Name','Please Wait');

waitbar\_count = 0;

waitbar\_update = round(N/10);

pause(0.001);

% Algorithm for pitch detection

I = 1;

for n = 1:N

% Update waitbar

if waitbar\_count > waitbar\_update

waitbar(n/N,hwait);

waitbar\_count = 0;

end

waitbar\_count = waitbar\_count + 1;

% Extract a block of the wave file

Atemp = A(I:I+block-1);

ttemp = t(I:I+block-1);

I = I+step;

% Do the autocorrelation to find the frequency

[f0(n),acorr,Nshifts,Tshifts] = ...

find\_f0\_timedomain2(Atemp,Fs,Fmin,Fmax,...

threshold\_amp,threshold\_ratio,harmonic\_deviation,...

f0\_target,target\_tol);

tcalc(n) = median(ttemp);

% If the previous and current are invalid, then make the last one invalid

% as well

if n > 2

if f0(n-2) < 1 && f0(n) < 1

f0(n-1) = 0;

end

end

% Look at the last few samples to determine the target frequency for

% the next iteration

if n >= f0\_samples

f0\_chunk = f0(n-f0\_samples+1:n);

f0\_target = f0\_chunk(f0\_chunk>0);

if ~isempty(f0\_target)

f0\_target = mean(f0\_target);

end

end

end

% Close waitbar

if ishandle(hwait)

close(hwait);

end

% Save calculation

handles.sound.A\_corrected = [];

handles.sound.f0 = f0;

handles.sound.f0\_save = [];

handles.sound.f0\_corrected = f0;

handles.sound.f0\_corrected\_save= [];

handles.sound.tcalc = tcalc;

handles.sound.selected\_points = false(size(f0));

handles.sound.selected\_points\_save= false(size(f0));

handles.sound.block = block;

handles.sound.step = step;

function handles = run\_pitch\_correction(handles)

% Runs only if theres a sound clip

if isempty(handles.sound.A)

return

end

% Get variables out of the handles structure

A = handles.sound.A;

t = handles.sound.t;

f0 = handles.sound.f0;

f02 = handles.sound.f0\_corrected;

Fs = handles.sound.Fs;

block = handles.sound.block;

step = handles.sound.step;

% Variables for size of analysis

scale\_factor = zeros(size(f0));

A\_corrected = zeros(size(A));

N = length(f0);

max\_acorr\_shift = 0;

max\_acorr\_amp = 0;

% Waitbar stuff

hwait = waitbar(0,'Correcting pitch...');

set(hwait,'Name','Please Wait');

waitbar\_count = 0;

waitbar\_update = round(N/10);

pause(0.001);

% Algorithm for correcting pitch

I = 1;

for n = 1:N

% Update waitbar

if waitbar\_count > waitbar\_update

waitbar(n/N,hwait);

waitbar\_count = 0;

end

waitbar\_count = waitbar\_count + 1;

% Pull out a chunk of the signal

Atemp = A(I:I+block-1);

ttemp = t(I:I+block-1);

if f0(n) > 0 && ~isnan(f0(n))

% Calculate the scale factor by which to shift the frequency

scale\_factor(n) = f02(n)/f0(n);

% Interpolate

tp = mean(ttemp) + (ttemp-mean(ttemp)).\*scale\_factor(n);

Ainterp = interp1(ttemp,Atemp ,tp)';

Ivalid = find(~isnan(Ainterp));

Ainterp(isnan(Ainterp)) = 0;

Nperiod = ceil(1/(f02(n)/Fs));

else

% No frequency shift

scale\_factor(n) = 1;

Ainterp = Atemp;

Ivalid = find(~isnan(Ainterp));

Nperiod = ceil(1/(handles.pitch\_options.Fmin/Fs));

end

if n == 1

A\_corrected(I:I+block-1) = Ainterp;

else

% Pull out one period of the new waveform

Achunk = Ainterp(Ivalid(1):Ivalid(1)+Nperiod-1);

factor = sum(abs(Achunk))\*2;

if ~all(scale\_factor(n-1:n) == 1)

% Start doing correlation

max\_acorr\_amp = 0;

max\_acorr\_shift = 0;

for Nshift = -round(Nperiod/2)+ (1:Nperiod)

% Calculate makeshift autocorrelation

acorr = 1-sum(abs(Achunk - A\_corrected((I:I+Nperiod-1) + Nshift + Ivalid(1))))./factor;

if acorr > max\_acorr\_amp

max\_acorr\_amp = acorr;

max\_acorr\_shift = Nshift;

end

end

end

[what,where] = min(abs(Achunk - ...

A\_corrected((I:I+Nperiod-1) + max\_acorr\_shift + Ivalid(1))));

A\_corrected((I+where-1:I+length(Ivalid)-1) + max\_acorr\_shift + Ivalid(1)) = ...

Ainterp(Ivalid(where:end));

end

I = I+step;

end

% Save values on handles

handles.sound.A\_corrected = A\_corrected;

handles.sound.scale\_factor = scale\_factor;

% Close waitbar

if ishandle(hwait)

close(hwait);

end

function [f0,acorr,Nshifts,Tshifts] = find\_f0\_timedomain2(A,Fs,Fmin,Fmax,...

threshold\_amp,threshold\_ratio,harmonic\_deviation,target\_f0,target\_tol)

% This function calculates the fundamental frequency of a signal in the time domain

% OUTPUTS:

% f0: The interpolated fundamental freqency (Hz)

% acorr: The vector of "autocorrelation" values

% Nshifts: The vector of shift indices associated with the values in

% acorr

% Tshifts: The vector of time shifts associated with the values in

% acorr

% INPUTS:

% A: The input signal A(t), sampled at an interval Ts

% Fs: The sample frequency (Hz)

% Fmin: (Optional) The minimum expected frequency (Hz)

% Fmax: (Optional) The maximum expected freqency (Hz)

% threshold\_amp: (Optional) The min autocorr amplitude considered peak

% threshold\_ratio: (Optional) The min ratio between max autocorr peak and

% a given peak to be considered as a possible peak

% harmonic deviation: (Optional) Tolerance for looking for another peak

% at half the frequency of the highest

% autocorrelation peak

% target\_f0: (Optional) The probable target frequency

% target\_tol: (Optional) The max error between target\_f0 and current freq

% (0.1 = 10%)

if nargin < 9 || isempty(target\_tol)

target\_tol = 0.1;

end

if nargin < 8 || isempty(target\_f0)

target\_f0 = [];

end

if nargin < 7 || isempty(harmonic\_deviation)

harmonic\_deviation = 0.03;

end

if nargin < 6 || isempty(threshold\_ratio)

threshold\_ratio = 0.7;

end

if nargin < 5 || isempty(threshold\_amp)

threshold\_amp = 0.3;

end

if nargin < 4 || isempty(Fmax)

Fmax = 800;

end

if nargin < 3 || isempty(Fmin)

Fmin = 40;

end

f0 = 0;

acorr = [];

Nshifts = [];

Tshifts = [];

% Calculate index of min and max shift

Nshift\_min = floor(1/(Fmax/Fs));

Nshift\_max = ceil(1/(Fmin/Fs));

if Nshift\_max+Nshift\_min > length(A) || Nshift\_min >= Nshift\_max

disp('Error in find\_f0\_timedomain2.m: Chunk size must be larger!')

return

end

% Pull out the chunk of the signal and calculate a scale factor

% The scale factor is the probable maximum value

Achunk = A(1:Nshift\_max);

scale\_factor = sum(abs(Achunk))\*2;

% Calculate a vector of all shifts

Nshifts = Nshift\_min:Nshift\_max;

Tshifts = Nshifts / Fs;

% Shift and calculate a makeshift autocorrelation

acorr = zeros(1,length(Nshifts));

index = 1;

for Nshift = Nshifts

% Break out if we are shifting the chunk beyond the end of the vector A

if Nshift\_max + Nshift > length(A)

Nshifts = Nshifts(1:index-1);

Tshifts = Tshifts(1:index-1);

acorr = acorr(1:index-1);

break

end

% Calculate makeshift autocorrelation

acorr(index) = 1-sum(abs(Achunk - A([1:Nshift\_max] + Nshift)))./scale\_factor;

index = index + 1;

end

% Try to make the autocorrelation "level" and with the minimum at zero

acorr = acorr - polyval(polyfit(Nshifts,acorr,1),Nshifts);

acorr = acorr - min(acorr);

% Find a list of all of the peaks above the threshold

[maxX,maxY,Imax,maxX\_fit,peakX,peakY] = peakfind(Tshifts,acorr,5,1/Fmax,[],'',false);

% Keep only those harmonics that are within a certain height of the largest

% peak and whose height is above the threshold amplitude

% Make a counter to keep track of which peaks remain after each criteria is

% applied below...

Ipeak = 1:length(maxX\_fit);

% Keep those whose heights are above the threshold

Ipeak = Ipeak(maxY > threshold\_ratio\*max(maxY) & maxY > threshold\_amp);

maxX\_fit = maxX\_fit(Ipeak);

% If only 1 or zero peaks remain, return

N = length(Ipeak);

if N < 2

return

end

% Keep only those harmonics whose frequency is less than a certain

% deviation around a multiple of the fundamental

[maxX\_fit,Ix] = sort(maxX\_fit);

Ipeak = Ipeak(Ix);

possible\_f0 = zeros(1,N);

for n = 1:N-1

if any(abs(1 - maxX\_fit(n+1:N)./(2\*maxX\_fit(n))) < harmonic\_deviation);

if isempty(target\_f0)

f0 = 1./maxX\_fit(n);

Ipeak = Ipeak(n);

break

else

possible\_f0(n) = 1./maxX\_fit(n);

end

end

end

% Now keep only the harmonic that is closest to the desired harmonic

if ~isempty(target\_f0)

f0\_error = abs(possible\_f0./target\_f0 - 1);

[what,where] = min(f0\_error);

if what < target\_tol

f0 = possible\_f0(where);

Ipeak = Ipeak(where);

end

end

% Now "fine tune" the frequency around the peak using a parabolic fit

if f0 > 0

p = polyfit(peakX(Ipeak,:),peakY(Ipeak,:),2);

Xfit = -p(2)/2/p(1);

f0 = 1/Xfit;

end

function [maxX,maxY,Imax,maxX\_fit,peakX,peakY] = peakfind(X,Y,Nmax,Xsep,Ymin,sortmethod,peakfit)

% This function finds the maxima of the given function, and

% returns the first Nvalues, sorted in decending order

% INPUTS:

% X = a vector of X-values for the signal

% Y = a fector of Y-values for the signal

% Nmax = maximum number of values to return

% Xsep = the minimum acceptable separation between "peaks"

% Ymin = the minimum Y value to return

% sortmethod = how to sort the peaks...

% 'maxY' = Sort in decending order starting with maximum Y-valued peak

% 'minX' = Sort in ascending order starting with minimum

% X-valued peak

% peakfit = set to 'true' to do a parabolic fit and refine the maxX values

% OUTPUTS:

% maxX = the X-coordinates of all of the N peaks

% maxY = the Y-heights of all of the N peaks

% Imax = indices of all of the N peaks

% maxX\_fit = a more refined list of the X-coordinates based on a parabolic

% fit to each peak

% peakX = a Nx5 matrix with each row containing the 5 X-values around

% each peak

% peakY = a Nx5 matrix with each row containing the 5 Y-values around each

% peak

if nargin < 7 || isempty(peakfit)

peakfit = true;

end

if nargin < 6 || isempty(sortmethod)

sortmethod = 'maxY';

end

if nargin < 5

Ymin = [];

end

if nargin < 4 || isempty(Xsep)

Xsep = 0;

end

if nargin < 3 || isempty(Nmax)

Nmax = 10;

end

% Differentiate, and find change in sign

Ydiff = diff(Y);

Imax1 = find(Ydiff(1:end-1) > 0 & Ydiff(2:end) < 0);

Imax2 = find(Ydiff(1:end-2) > 0 & Ydiff(2:end-1) == 0 & Ydiff(3:end) < 0);

Imax3 = find(Ydiff(1:end-3) > 0 & Ydiff(2:end-2) == 0 & Ydiff(3:end-1) == 0 & Ydiff(4:end) < 0);

% Concatenate all of the possible peaks

maxX = [X(Imax1+1) X(Imax2+1) X(Imax3+2)];

maxY = [Y(Imax1+1) Y(Imax2+1) Y(Imax3+2)];

Imax = [(Imax1+1) (Imax2+1) (Imax3+2)];

% Re-sort if desired

if strcmp(sortmethod,'minX')

[maxX,IX] = sort(maxX,'ascend');

maxY = maxY(IX);

Imax = Imax(IX);

elseif strcmp(sortmethod,'maxY')

[maxY,IY] = sort(maxY,'descend');

maxX = maxX(IY);

Imax = Imax(IY);

end

% Remove any peaks that are below the min peak value allowed

if ~isempty(Ymin)

Irem = find(maxY < Ymin);

maxY(Irem) = [];

maxX(Irem) = [];

Imax(Irem) = [];

end

% Remove any peaks that are closer than the minimum allowed peak seperation

% in X

if Xsep > 0

Iremove = zeros(1,length(maxX));

I = 0;

for n = 2:length(maxX)

if abs(maxX(n) - maxX(n-1)) < Xsep

maxX(n) = maxX(n-1);

maxY(n) = maxY(n-1);

I = I+1;

Iremove(I) = n;

end

end

Iremove = Iremove(1:I);

maxX(Iremove) = [];

maxY(Iremove) = [];

Imax(Iremove) = [];

end

% Remove any peaks beyond the max number to return

if length(maxX) > Nmax

maxX = maxX(1:Nmax);

maxY = maxY(1:Nmax);

Imax = Imax(1:Nmax);

end

maxX\_fit = maxX;

peakX = zeros(length(Imax),5);

peakY = zeros(length(Imax),5);

% Do a 2nd-order poly fit around the peak to pinpoint the peak location

for n = 1:length(Imax);

if Imax(n) > 3 && Imax(n) < length(X)-2

I = Imax(n)+[-2:2];

if peakfit

p = polyfit(X(I),Y(I),2);

maxX\_fit(n) = -p(2)/2/p(1);

end

peakX(n,1:5) = X(I);

peakY(n,1:5) = Y(I);

end

end

% --- Plot Functions

function handles = update\_plot(handles)

val = get(handles.view,'value');

axes(handles.axes1);

if val == 1

if ~isempty(handles.sound.t) && ~isempty(handles.sound.A)

hplot = [];

legends = {};

if ~isempty(handles.sound.A\_corrected)

hplot(1) = plot(handles.axes1,handles.sound.t,handles.sound.A,'color',[0.7 0.7 0.7]);

else

hplot(1) = plot(handles.axes1,handles.sound.t,handles.sound.A,'color','b');

end

legends{1} = 'Original';

if ~isempty(handles.sound.A\_corrected)

hold on

hplot(2) = plot(handles.axes1,handles.sound.t,handles.sound.A\_corrected,'color','b');

legends{2} = 'Modified';

hold off

end

if isempty(handles.status.Xlim)

set(handles.axes1,'Xlim',[min(handles.sound.t) max(handles.sound.t)]);

else

set(handles.axes1,'Xlim',handles.status.Xlim);

end

dA = max(handles.sound.A)-min(handles.sound.A);

set(handles.axes1,'Ylim',[min(handles.sound.A)-0.1\*dA max(handles.sound.A)+0.1\*dA]);

else

cla

end

elseif val == 3

if ~isempty(handles.sound.tcalc) && ~isempty(handles.sound.f0)

hplot = [];

f0 = handles.sound.f0;

f0(f0 < 1) = NaN;

if all(isnan(f0))

cla

hwarn = warndlg('No pitch detected with current settings!','WARNING');

waitfor(hwarn);

else

hplot(1) = semilogy(handles.axes1,handles.sound.tcalc,f0,'.-','color',[0.7 0.7 0.7]);

%legends{1} = 'Original';

if ~isempty(handles.sound.f0\_corrected)

hold on

f02 = handles.sound.f0\_corrected;

f02(f02 < 1) = NaN;

hplot(2) = semilogy(handles.axes1,handles.sound.tcalc,f02,'.-','color','b');

%legends{2} = 'Modified';

if any(handles.sound.selected\_points)

f03 = f02;

f03(~handles.sound.selected\_points) = NaN;

semilogy(handles.axes1,...

handles.sound.tcalc,f03,'.-','color','r');

end

hold off

end

grid on

Ylim = [min(f0)/1.2 max(f0)\*1.2];

set(handles.axes1,'Ylim',Ylim);

set(handles.axes1,'Ytick',handles.scale\_options.freqs);

set(handles.axes1,'YtickLabel',handles.scale\_options.notes);

if isempty(handles.status.Xlim)

set(handles.axes1,'Xlim',[min(handles.sound.t) max(handles.sound.t)]);

else

set(handles.axes1,'Xlim',handles.status.Xlim);

end

end

else

cla

end

end

% Invisible axes for getting mouse information

set(handles.axes2,'Xlim',get(handles.axes1,'Xlim'));

set(handles.axes2,'Ylim',get(handles.axes1,'Ylim'));

set(handles.axes2,'Yscale',get(handles.axes1,'Yscale'));

set(handles.axes2,'ButtonDownFcn','AudioTuner(''mouseclick'',gcbo,[],guidata(gcbo))');

set(handles.axes2,'Color','none');

axes(handles.axes2);

function nokeyresponse(hObject, eventdata, handles)

% Makes nothing happen when you press a key

function mouseclick(hObject, ~, handles)

% Executes when the mouse is clicked on the plot

% Executes only if pich correction is made

if isempty(handles.sound.f0)

return

end

% Executes only on graph 3

if get(handles.view,'value') == 3

clicktype = get(gcf,'SelectionType');

if strcmp(clicktype,'normal') || strcmp(clicktype,'alt')

point1 = get(gca,'CurrentPoint'); % button down detected

finalRect = rbbox; % return figure units

point2 = get(gca,'CurrentPoint'); % button up detected

point1 = point1(1,1:2); % extract x and y

point2 = point2(1,1:2);

p1 = min(point1,point2); % calculate locations

offset = abs(point1-point2); % and dimensions

xminmax = [p1(1) p1(1)+offset(1)];

yminmax = [p1(2) p1(2)+offset(2)];

if strcmp(clicktype,'normal')

handles.sound.selected\_points(:) = false;

end

handles.sound.selected\_points(...

handles.sound.f0\_corrected > yminmax(1) & ...

handles.sound.f0\_corrected < yminmax(2) & ...

handles.sound.tcalc > xminmax(1) & ...

handles.sound.tcalc < xminmax(2) & ...

handles.sound.f0\_corrected > 0) = true;

elseif strcmp(clicktype,'extend')

% Save for undoing

handles.sound.f0\_save(end+1,:) = handles.sound.f0;

handles.sound.f0\_corrected\_save(end+1,:) = handles.sound.f0\_corrected;

handles.sound.selected\_points\_save(end+1,:) = handles.sound.selected\_points;

% Get initial click point

point1 = get(gca,'CurrentPoint'); % button down detected

point1 = point1(1,1:2);

handles.point1 = point1;

handles.single\_point = false;

% If no points have previously been selected, select the closest point

if all(~handles.sound.selected\_points)

a = axis;

tnorm = (handles.sound.tcalc - a(1))/(a(2)-a(1));

fnorm = (handles.sound.f0\_corrected - a(3))/(a(4)-a(3));

xnorm = (point1(1) - a(1))/(a(2)-a(1));

ynorm = (point1(2) - a(3))/(a(4)-a(3));

[what,where] = min((tnorm-xnorm).^2+(fnorm-ynorm).^2);

handles.sound.selected\_points(where) = true;

handles.single\_point = true;

end

guidata(hObject, handles);

pause(0.05);

% Set function to track mouse position

handles.tstart = now;

guidata(hObject, handles);

set(gcf,'WindowButtonMotionFcn','AudioTuner(''wbmcb'',gcbo,[],guidata(gcbo))');

set(gcf,'WindowButtonUpFcn','AudioTuner(''wbucb'',gcbo,[],guidata(gcbo))');

end

end

update\_plot(handles);

Scale\_Callback(hObject, [], handles)

guidata(hObject, handles);

function wbmcb(hObject, ~, handles)

% Note: added in a timer so that the figure won't try to refresh too

% quickly and crash

if (now-handles.tstart)\*24\*60\*60 > 0.08

handles.tstart = now;

cp = get(gca,'CurrentPoint');

ydiff = cp(1,2)/handles.point1(2);

handles.sound.f0\_corrected(handles.sound.selected\_points) = ...

handles.sound.f0\_corrected\_save(end,handles.sound.selected\_points).\*ydiff;

handles.sound.A\_corrected = [];

guidata(hObject, handles);

update\_plot(handles);

end

function wbucb(hObject, ~, handles)

if handles.single\_point

handles.sound.selected\_points(:) = false;

end

guidata(hObject, handles);

update\_plot(handles);

set(gcf,'WindowButtonMotionFcn','');

set(gcf,'WindowButtonUpFcn','');

function [indices,freqs,notes,fund\_index] = get\_scale(scale)

indices = [];

freqs = [];

notes = {};

fund\_index = [];

major\_indices = [0 2 4 5 7 9 11];

minor\_indices = [0 2 3 5 7 8 10];

fund\_indices = [0 2 3 5 7 8 10];

fund\_notes = {'A' 'B' 'C' 'D' 'E' 'F' 'G'};

scale\_types = {'major','minor'};

if nargin < 1 || isempty(scale)

% Return all possiblities of scales

indices = {};

for n = 1:length(fund\_notes)

for m = 1:2

indices{end+1} = [fund\_notes{n} scale\_types{m}];

end

end

return

else

fund\_index = fund\_indices(strcmp(scale(1),fund\_notes));

if strcmp(scale(2:end),'major')

indices = major\_indices+fund\_index;

elseif strcmp(scale(2:end),'minor')

indices = minor\_indices+fund\_index;

else

disp('ERROR in get\_scale.m: unknown scale!')

return

end

end

indices = [indices-12 indices indices+12 indices+24 indices+36];

indices = indices(indices >=0 & indices <=48);

[indices,freqs,notes] = get\_note\_matrix(indices);

function [indices,freqs,notes] = get\_note\_matrix(note)

if nargin < 1

note = '';

end

indices = (0:48)';

freqs = 55.\*2.^(indices./12);

notes = {

'A1'

'A#/Bb1'

'B1'

'C1'

'C#/Db1'

'D1'

'D#/Eb1'

'E1'

'F1'

'F#/Gb1'

'G1'

'G#/Ab1'

'A2'

'A#/Bb2'

'B2'

'C2'

'C#/Db2'

'D2'

'D#/Eb2'

'E2'

'F2'

'F#/Gb2'

'G2'

'G#/Ab2'

'A3'

'A#/Bb3'

'B3'

'C3'

'C#/Db3'

'D3'

'D#/Eb3'

'E3'

'F3'

'F#/Gb3'

'G3'

'G#/Ab3'

'A4'

'A#/Bb4'

'B4'

'C4'

'C#/Db4'

'D4'

'D#/Eb4'

'E4'

'F4'

'F#/Gb4'

'G4'

'G#/Ab4'

'A5'

};

if ~isempty(note)

if ischar(note)

I = find(strcmp(notes,note));

if ~isempty(I);

indices = indices(I);

freqs = freqs(I);

notes = notes(I);

else

indices = [];

freqs = [];

notes = '';

end

else

note = note(note>=0 & note<=48);

indices = note;

freqs = freqs(note+1);

notes = notes(note+1);

end

end

function handles = update\_GUI(handles)

% Change button if is recording or not

if handles.status.isrecording

set(handles.Play,'enable','off');

set(handles.Record,'enable','off');

set(handles.Stop,'enable','on');

else

if ~isempty(handles.sound.A)

set(handles.Play,'enable','on');

else

set(handles.Play,'enable','off');

end

set(handles.Record,'enable','on');

set(handles.Stop,'enable','off');

end

% AutomaticFit

if isempty(handles.status.Xlim) && isempty(handles.status.Ylim)

set(handles.AutomaticFit,'enable','off');

else

set(handles.AutomaticFit,'enable','on');

end