Design of a Speaker Recognition Code using MATLAB

E. Darren Ellis
Department of Computer and Electrical Engineering – University of Tennessee, Knoxville Tennessee 37996

(Submitted: 09 May 2001)

This project entails the design of a speaker recognition code using MATLAB. Signal processing in the time and frequency domain yields a powerful method for analysis. MATLAB's built in functions for frequency domain analysis as well as its straightforward programming interface makes it an ideal tool for speech analysis projects. For the current project, experience was gained in general MATLAB programming and the manipulation of time domain and frequency domain signals. Speech editing was performed as well as degradation of signals by the application of Gaussian noise. Background noise was successfully removed from a signal by the application of a 3rd order Butterworth filter. A code was then constructed to compare the pitch and formant of a known speech file to 83 unknown speech files and choose the top twelve matches.

I. INTRODUCTION

Development of speaker identification systems began as early as the 1960s with exploration into voiceprint analysis, where characteristics of an individual's voice were thought to be able to characterize the uniqueness of an individual much like a fingerprint. The early systems had many flaws and research ensued to derive a more reliable method of predicting the correlation between two sets of speech utterances. Speaker identification research continues today under the realm of the field of digital signal processing where many advances have taken place in recent years.

In the current design project a basic speaker identification algorithm has been written to sort through a list of files and choose the 12 most likely matches based on the average pitch of the speech utterance as well as the location of the formants in the frequency domain representation. In addition, experience has been gained in basic filtering of high frequency noise signals with the use of a Butterworth filter as well as speech editing techniques.

II. APPROACH

This multi faceted design project can be categorized into different sections: speech editing, speech degradation, speech enhancement, pitch analysis, formant analysis and waveform comparison. The resulting discussion will be segmented based on these delineations.

SPEECH EDITING

The file recorded with my slower speech (a17.wav) was found from the ordered list of speakers. A plot of this file is shown in Figure (1). It was determined that the length of the vector representing this speech file had a magnitude of 30,000. Thus the vector was partitioned into two separate vectors of equal length and the vectors were written to a file in opposite order. The file was then read and played back. The code for this process can be found in Appendix A.

SPEECH DEGRADATION

The file recorded with my faster speech (a18.wav) was found from the ordered list of speakers. Speech degradation was performed by adding Gaussian noise generated by the MATLAB function randn() to this file. A comparison was then made between the clean file and the signal with the addition of Gaussian noise. The code for this process can be found in Appendix B.

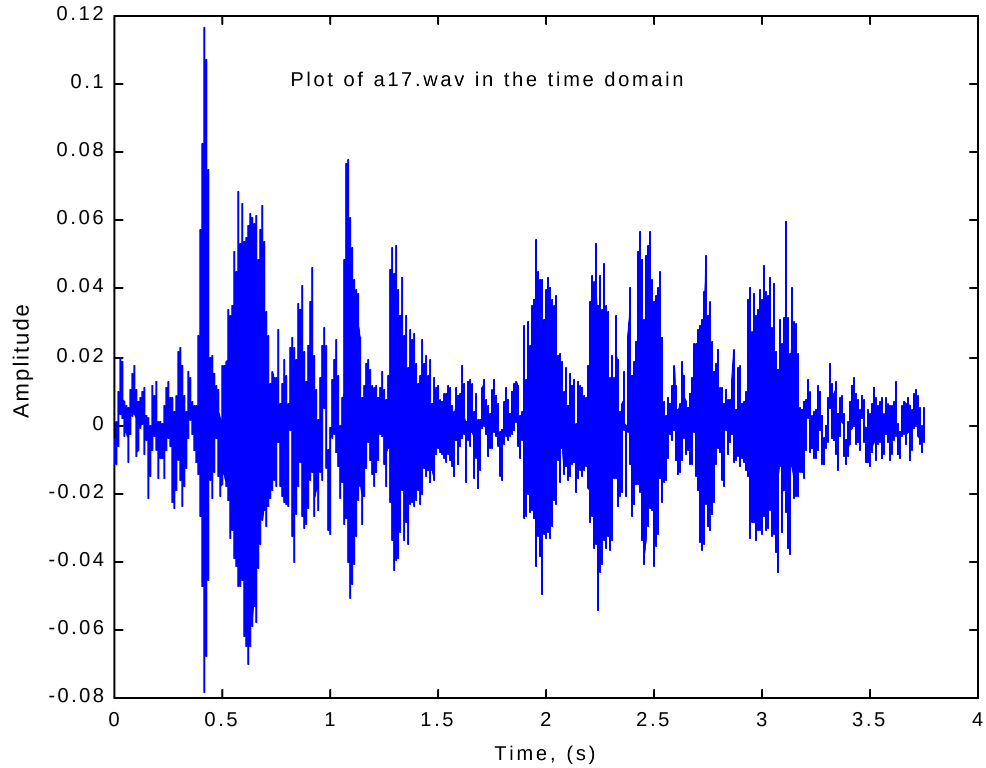

Fig 1. Time domain plot of a17.wav.

SPEECH ENHANCEMENT

The file recorded with my slower speech and noise in the background (a71.wav) was found from the ordered list of speakers. A plot of this file is shown in Figure (2). This signal was then converted to the frequency domain through the use of a shifted FFT and correctly scaled frequency vector. The higher frequency noise

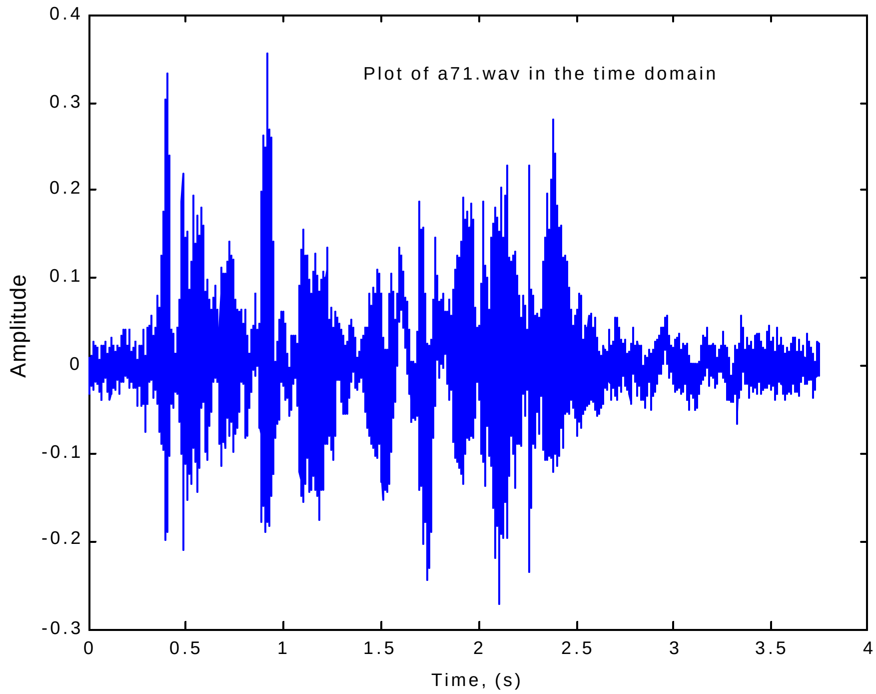

Fig 2. Time domain plot of a71.wav.

components were then removed by application of a 3rd order Butterworth low pass filter, Eq.(1), with the cutoff chosen to remove as much of the noise signal as possible while still preserving the original signal.

$$H_B(u,v) \cong \frac{1}{1+(\sqrt{2}-1)(D(u,v)/D_o)^{2n}} \qquad (1)$$

where D(u,v) is the rms value of u and v, D_o determines the cutoff frequency, and n is the filter order. The Butterworth filter is a reasonable choice to use as it more closely approximates an ideal low pass filter as the order, n, is increased.

The resulting filtered signal was then scaled and plotted with the original noisy signal to compare the filtering result. The code for this process can be found in Appendix C.

PITCH ANALYSIS

The file recorded with my slower speech (a17.wav) was found from the ordered list of speakers. Pitch analysis was conducted and relevant parameters were extracted. The average pitch of the entire wav file was computed and found to have a value of 154.8595 Hz. The graph of pitch contour versus time frame was also created to see how the pitch varies over the wav file, Figure (3). The results of pitch analysis can be used in speaker recognition, where the differences in average pitch can be used to characterize a speech file. The code for this process can be found in Appendix D.

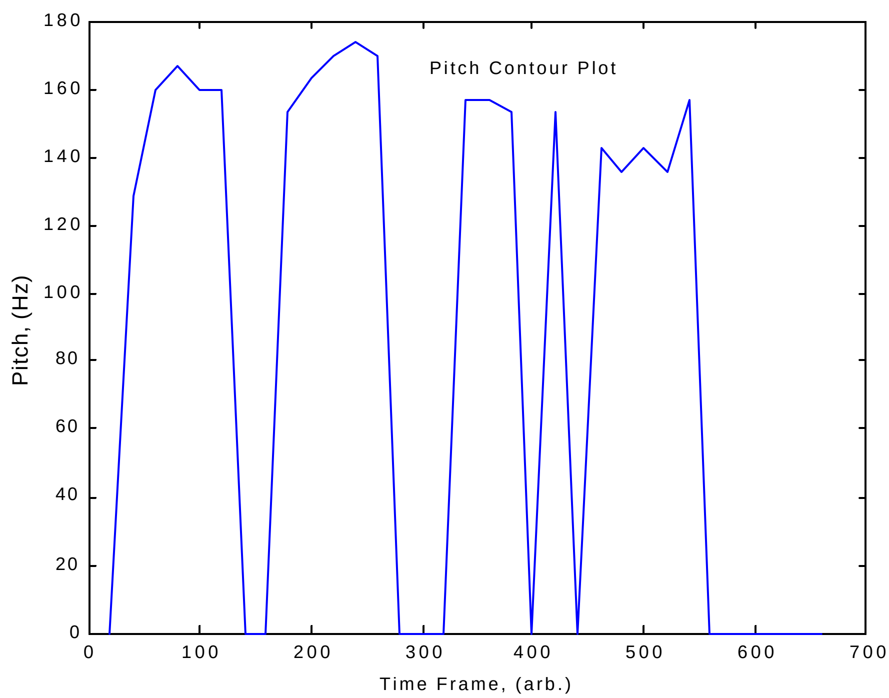

Fig 3. Pitch contour plot.

FORMANT ANALYSIS

Formant analysis was performed on my slow speech file (a17.wav). The first five peaks in the power spectral density were returned and the first three can be seen in Figure (4). Also, the vector position of the peaks in the power spectral density were calculated and can be used to characterize a particular voice file. This technique is used in the waveform comparison section. The code for this process can be found in Appendix E.

WAVEFORM COMPARISON

Using the results and information learned from pitch and formant analysis, a waveform comparison code was written. Speech waveform files can be characterized based on various criteria. Average pitch and formant peak position vectors are two

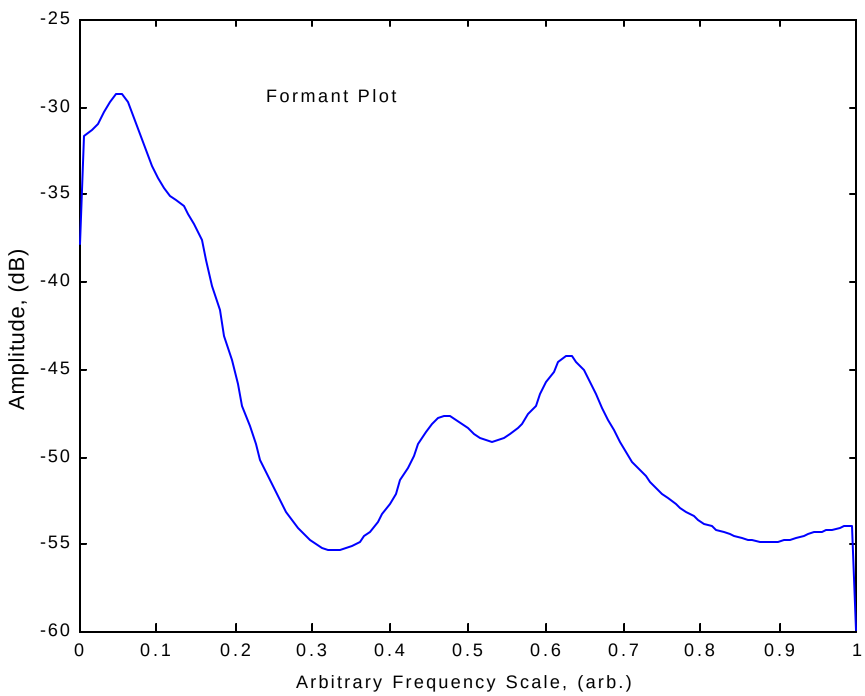

Fig 4. Plot of the first few formants of a17.wav.

such criteria that can be used to characterize a speech file. The slow speech file (a17.wav) was used as a reference file. Four sorting routines were then written to compare the files. The sorting routines performed the following functions: sort and compare the average pitch of the reference file with all 83 wav files, compare the formant vector of the reference file to all wav files, sort for the top 20 average pitch correlations and then sort these files by formant vectors, and finally to sort for the top 20 formant vector correlations and then sort these by average pitch. Sample code for the case of comparing the average pitch and then comparing the top 12 most likely matches by formant peak difference vectors is given in Appendix F. The three other cases use code from this sample to achieve their results.

III. RESULTS

Results of speech editing are shown in Figure (5). As can be seen, the phrase "ECE-310," the second half of the first plot, has clearly been moved to the front of the waveform in the second plot.

Speech degradation by the application of Gaussian noise can be seen in Figure (6). The upper plot shows the signal from wav file a18.wav in the time domain. The middle plot yields a frequency domain view of the same wav file. The bottom plot allows for a comparison between the clean signal (middle plot) and one with Gaussian noise added to it. Results of the speech enhancement routine can be seen in Figure (7). The upper plot shows the file a71.wav with natural background noise. The noise signal is more

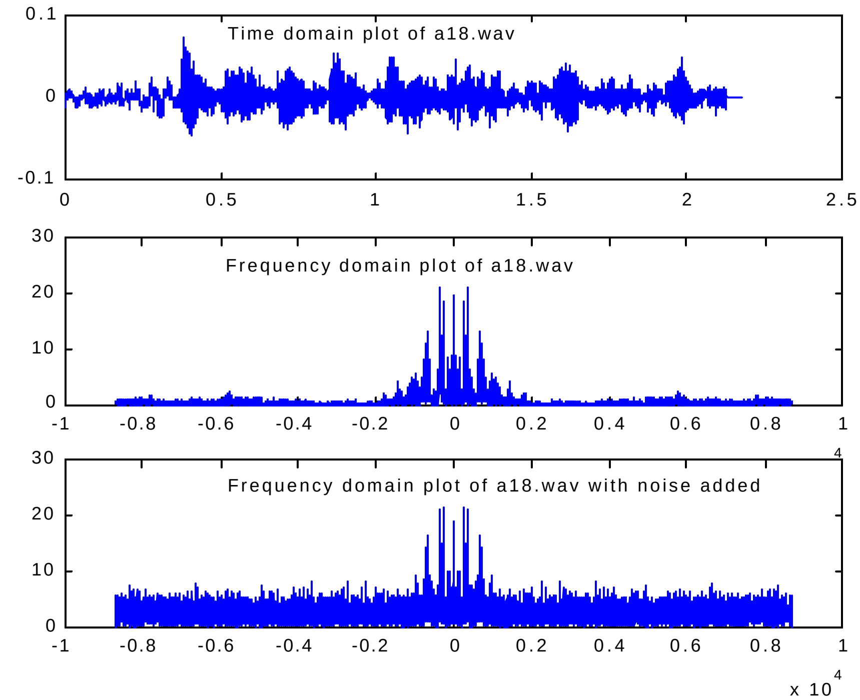

Fig 5. File a18.wav with and without Gaussian noise added to it.

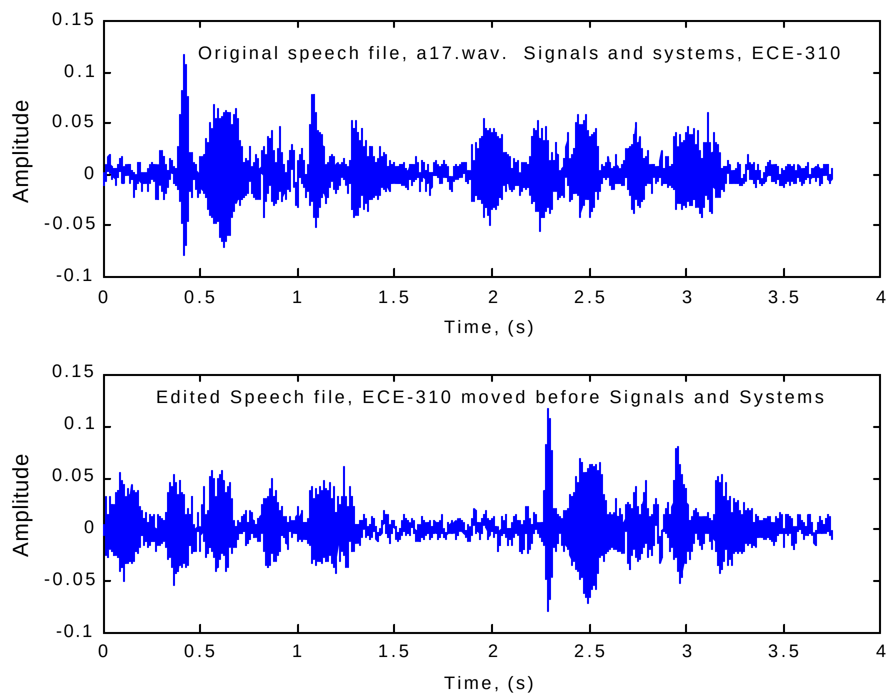

Fig 6. Example of speech editing.

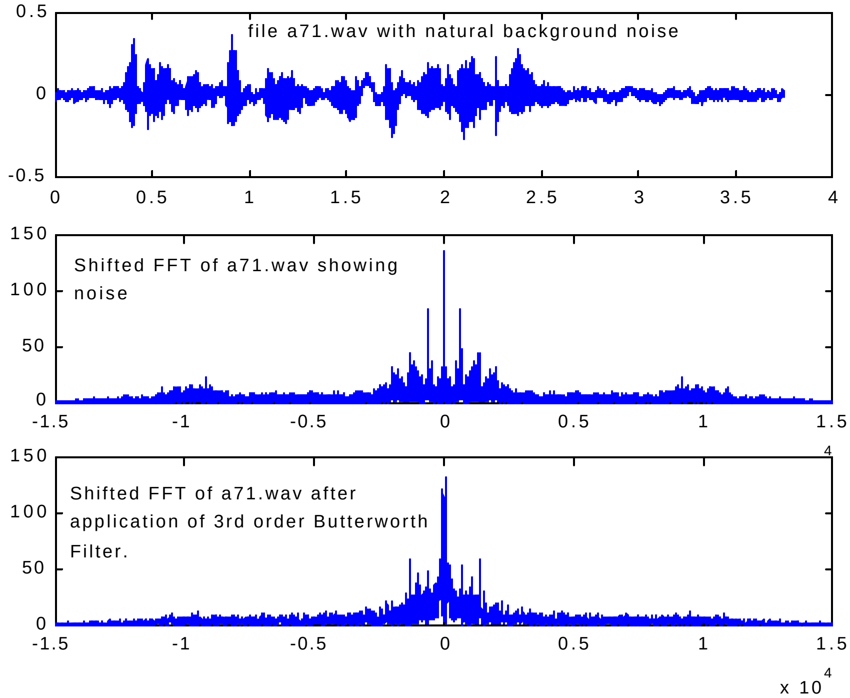

Fig 7. File a71.wav. Comparison of natural and LPF filtered signal.

prevalent in the middle figure which shows the shifted FFT of the original signal. The noise can be seen as a broad peak at approximately $1x10^4$ Hz, as well as an overall background component. The bottom figure shows the signal after application of a 3^{rd} order Butterworth filter and amplitude scaling to yield a valid comparison to the original signal.

The results of pitch analysis were used in the waveform comparison section of the speech recognition project. The results of the average pitch of all four of my speech files are summarized in Table (1).

Table 1. Summary of pitch characteristics.

Wav File Name	Average Pitch (Hz)	Characteristic of Wav File
A17.wav	154.8595	Slow speech
A18.wav	158.4562	Fast speech
A71.wav	154.8068	Slow speech with background noise
A52.wav	188.0342	Slow speech, different phrase

As can been seen from Table (1), the average pitch varies for faster speech utterances as well as for different phrases. The addition of background noise affects the average pitch very little, however, speaking a different phrase produces a change of greater than 30 Hz.

A plot of the Power spectral density, Figure (8), for my four speech files shows the location of the first few formants present in each file. Good agreement between the peak locations of file a17.wav and a18.wav is seen in the first and second plots, where the same phrase is spoken but at different rates. However, file a71.wav, with the background noise shows a large background component over a wide frequency range and shields the location of some of the lower amplitude peaks. Also, the last plot of the PSD of a phrase different than the upper three plots shows the location of the formant peaks slightly shifted in frequency, as would be expected. One of the routines used in the waveform comparison section of the project calculates the vector difference between peak locations in the PSD and compares this vector to the same characteristic of all the other wav files

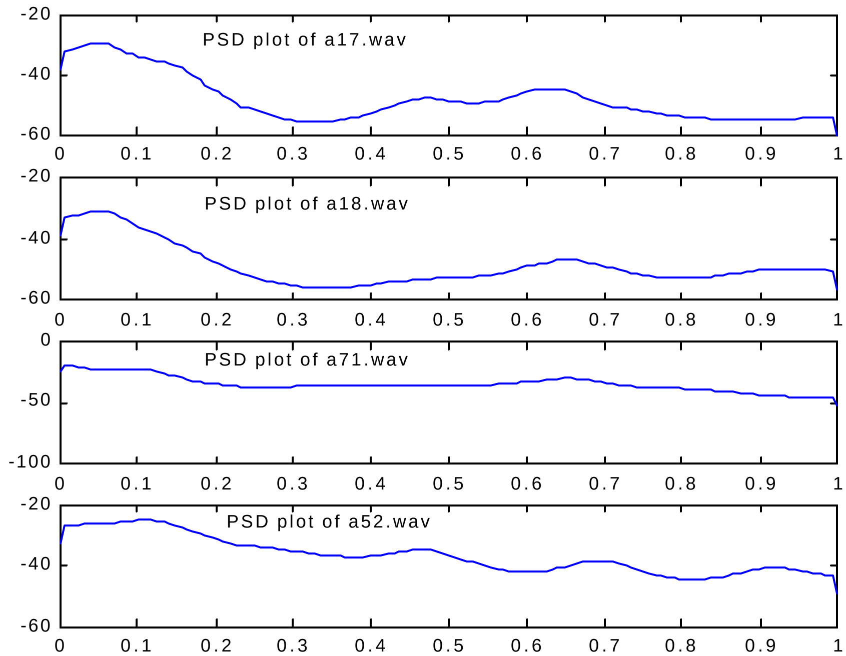

Fig 8. Comparison of PSD of the wav files.

.

In order to create a speech recognition algorithm, criteria to compare speech files must be established. This section of the project compares four different methods of comparing the data. First, the wav files are compared to a reference file and sorted based on the average pitch of the file only (Method 1). The files were then compared and sorted based entirely on the location of the formants present in the PSD of the signal (Method 2). A third method compared the average pitch present and ranked the matches in ascending order and then compared the top 12 most likely matches by formant location in the PSD (Method 3). Finally, the inverse routine was performed where the files were compared and sorted by the location of the formants present and then the top 12 most likely matches based on this data were compared and sorted by pitch (Method 4). Table (2) compares the results of this work.

Table (2). Comparison of the four different comparison methods.

Method 1	My file? (*)	Method 2	My file? (*)	Method 3	My file? (*)	Method 4	My file? (*)
a17.wav	*	a17.wav	*	a17.wav	*	a17.wav	*
a71.wav	*	a12.wav		a63.wav		a63.wav	
a19.wav		a07.wav		a65.wav		a65.wav	
a08.wav		a52.wav	*	a71.wav	*	a72.wav	
a73.wav		a63.wav		a73.wav		a03.wav	
a63.wav		a72.wav		a8.wav		a07.wav	
a15.wav		a53.wav		a19.wav		a12.wav	
a01.wav		a03.wav		a14.wav		a52.wav	*
a20.wav		a65.wav		a01.wav		a13.wav	
a18.wav	*	a13.wav		a15.wav		a36.wav	
a65.wav		a36.wav		a18.wav	*	a40.wav	
a14.wav		a40.wav		a20.wav		a53.wav	

As can be seen from Table (2), all four methods were able to correctly pick out the reference file. However, the two methods that utilized comparison based on average pitch were most successful in picking other matches. Of these two, the method that made comparisons based on average pitch alone had the most accuracy, correctly choosing two of my files as the top two most likely matches. Formant comparisons were not as successful, at most only correctly finding two of my files out of the group. This result is counter to what I had assumed before beginning this project. However, the reduced accuracy of the formant comparison could have several contributing factors. Differences in recording levels and conditions could have impacted the results. Also, the differences in phrases spoken during the recording phase would introduce shifted formant frequencies, as would be expected due to differing average format frequencies between different vowels, making comparison based on this criteria troublesome. Improvements in this respect would be to compare like phrases only, under better/more controlled recording conditions.

IV. Conclusion

A crude speaker recognition code has been written using the MATLAB programming language. This code uses comparisons between the average pitch of a recorded wav file as well as the vector differences between formant peaks in the PSD of each file. It was found that comparison based on pitch produced the most accuracy, while comparison based on formant peak location did produce results, but could likely be improved. Experience was also gained in speech editing as well as basic filtering techniques. While the methods utilized in the design of the code for this project are a good foundation for a speaker recognition system, more advanced techniques would have to be used to produce a successful speaker recognition system.

REFERENCES

Speech Production, Labeling, and Characteristics. Handout given in class.

Voice Recognition. Handout given in class.

http://everest.radiology.uiowa.edu/~jmr/lecture/node64.html

APPENDIX A

```
%File to cut and paste parts of a wav file in reverse order
%Author = E. Darren Ellis 05/01

[y, fs, nbits] = wavread('a17.wav');  %read in the wav file
sound(y,fs)                           %play back the wav file
t = 0:1/fs:length(y)/fs-1/fs;         %create the proper time vector
subplot(211)                          %create a subplot
plot(t,y)                             %plot the original waveform


yfirst=y(1:15000);               %partition the vector into two parts
ysecond=y(15001:30000);

save darren ysecond yfirst -ascii     %save the vector in reverse order
load darren -ascii                    %read back in the new file


subplot(212)                          %prepare a new subplot
plot(t,darren)       %plot the new file to compare it to the original
pause(2)             %create a 2 second pause
sound(darren,fs);    %play back the new sound file
```

APPENDIX B

```
%Code to add gaussian noise to a signal and then plot the original
%signal in the time domain, the shifted FFT of the original signal in
%the frequency domain %and the shifted FFT of the original signal with
%gaussian noise added to it in the frequency domain.

%Author = E. Darren Ellis 05/01


[y, fs, nbits] = wavread('a18.wav');   %read in the wav file
t = 0:1/fs:length(y)/fs-1/fs;       %generate the correct time vector
subplot(311)                           %set up a subplot
plot(t,y)                           %plot the signal in the time domain

%%%%%code provided by Dr. Qi to generate gaussian noise%%%%%
sigma = 0.02;
mu = 0;
n = randn(size(y))*sigma + mu*ones(size(y));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

signal=n+y;             %add the gaussian noise to the original signal
yfft=fft(y);               %take the FFT of the original signal.
xfft=fft(signal);          %take the FFT of the signal with noise added
f = -length(y)/2:length(y)/2-1;  %generate the appropriate frequency
                                 %scale.
ysfft=fftshift(yfft);      %calculate the shifted FFT of the original
                           %signal
xsfft=fftshift(xfft);      %same as above but for the signal with noise
                           %added

subplot(312)

%plot the shifted FFT of the original signal in the frequency domain
plot(f,abs(ysfft));

subplot(313)

%plot the shifted FFT of the original signal with noise added in the
%frequency domain
plot(f,abs(xsfft));
```

APPENDIX C

```
%Code to plot a noisy signal, take the shifted FFT of teh noisy signal
and apply a
%Butterworth filter to it.  The filtered signal is then scaled and
plotted to compare
%to the original signal

%Author = E. Darren Ellis 05/01


[y, fs, nbits] = wavread('a71.wav');   %read in the wav file
t = 0:1/fs:length(y)/fs-1/fs;       %generate the correct time vector
subplot(311)                        %create a subplot
plot(t,y)                           %plot the signal in the time domain
sound(y,fs)                               %play back the wav file

yfft=fft(y);                     %take the FFT of the original signal
f = -length(y)/2:length(y)/2-1;           %create the appropriate
                                          %frequency vector
ysfft=fftshift(yfft);                     %Shift the FFT of the
                                          %original signal

subplot(312)
plot(f,abs(ysfft));        %plot the shifted FFT of the orginal signal


%%%%%code provided by Dr. Qi to generate and apply the Butterworth
%filter%%%%%
order = 3;
cut = 0.05;
[B, A] = butter(order, cut);
filtersignal = filter(B, A, ysfft);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


subplot(313)
plot(f,21*abs(filtersignal));             %plot the scaled and filtered
                                          %signal to compare
```

APPENDIX D

```
%Code for pitch analysis of a wav file.  This code needs the pitch.m
%and pitchacorr.m files to be in the same directory.  A plot of pitch
%contour versus time frame is created and the average pitch of the wav
%file is returned.

%Author = E. Darren Ellis 05/01



[y, fs, nbits] = wavread('a17.wav');      %read in the wav file
[t, f0, avgF0] = pitch(y,fs)              %call the pitch.m routine
plot(t,f0)                           %plot pitch contour versus time frame
avgF0                                     %display the average pitch
sound(y)                                  %play back the sound file

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%

%
% Function:
%    Extract pitch information from speech files
%    pitch can be obtained by obtaining the peak of autocorrelation
%    usually the original speech file is segmented into frames
%    and pitch contour can be derived by plot of peaks from frames
%
% Input:
%    x: original speech
%    fs: sampling rate
%
% Output:
%    t: time frame
%    f0: pitch contour
%    avgF0: average pitch frequency
%
% Acknowledgement:
%    this code is based on Philipos C. Loizou's colea Copyright (c)
%1995
%

function [t, f0, avgF0] = pitch(y, fs)

% get the number of samples
ns = length(y);

% error checking on the signal level, remove the DC bias
mu = mean(y);
y = y - mu;

% use a 30msec segment, choose a segment every 20msec
% that means the overlap between segments is 10msec
fRate = floor(120*fs/1000);
updRate = floor(110*fs/1000);
```

```
nFrames = floor(ns/updRate)-1;

% the pitch contour is then a 1 x nFrames vector
f0 = zeros(1, nFrames);
f01 = zeros(1, nFrames);

% get the pitch from each segmented frame
k = 1;
avgF0 = 0;
m = 1;
for i=1:nFrames
  xseg = y(k:k+fRate-1);
  f01(i) = pitchacorr(fRate, fs, xseg);
  % do some median filtering, less affected by noise
  if i>2 & nFrames>3
    z = f01(i-2:i);
    md = median(z);
    f0(i-2) = md;
    if md > 0
      avgF0 = avgF0 + md;
      m = m + 1;
    end
  elseif nFrames<=3
    f0(i) = a;
    avgF0 = avgF0 + a;
    m = m + 1;
  end
  k = k + updRate;
end

t = 1:nFrames;
t = 20 * t;

if m==1
  avgF0 = 0;
else
  avgF0 = avgF0/(m-1);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Pitch estimation using the autocorrelation method
% Modified based on colea Copyright (c) 1995 Philipos C. Loizou
%

function [f0] = pitchacorr(len, fs, xseg)

% LPF at 900Hz
[bf0, af0] = butter(4, 900/(fs/2));
xseg = filter(bf0, af0, xseg);

% find the clipping level, CL

i13 = len/3;
maxi1 = max(abs(xseg(1:i13)));

i23 = 2 * len/3;
maxi2 = max(abs(xseg(i23:len)));
```

```
if maxi1>maxi2
  CL=0.68*maxi2;
else
  CL= 0.68*maxi1;
end

% Center clip waveform, and compute the autocorrelation
clip = zeros(len,1);
ind1 = find(xseg>=CL);
clip(ind1) = xseg(ind1) - CL;

ind2 = find(xseg <= -CL);
clip(ind2) = xseg(ind2)+CL;

engy = norm(clip,2)^2;

RR = xcorr(clip);
m = len;

% Find the max autocorrelation in the range 60 <= f <= 320 Hz
LF = floor(fs/320);
HF = floor(fs/60);

Rxx = abs(RR(m+LF:m+HF));
[rmax, imax] = max(Rxx);

imax = imax + LF;
f0 = fs/imax;

% Check max RR against V/UV threshold
silence = 0.4*engy;

if (rmax > silence)  & (f0 > 60) & (f0 <= 320)
 f0 = fs/imax;
else % -- its unvoiced segment ---------
 f0 = 0;
end
```

APPENDIX E

```
%Code to calculate and plot the first three formants present in a
%speech file and
%calculate the vector differences between peak positions of the first
%five formants.
%This code requires formant.m and pickmax.m to be in the same directory

%Author = E. Darren Ellis 05/01


[y, fs, nbits] = wavread('a17.wav');      %read in my speech file.
[P,F,I] = formant(y);                     %apply formant routine and
                                          %return P, F, and I.
sound(y)                                  %play the speech file.
plot(F,P)                                 %plot formants.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Function:
%    Return the first five formants of a speech file
%
% Input:
%    The speech file "y"
%
% Output:
%    The PSD (P), the normalized frequency axis (F), the position of %
the peak (I)
%
% Author:
%    Hairong Qi
%
% Date:
%    04/25/01
%
function [P, F, I] = formant(y)

% calculate the PSD using Yule-Walker's method
order = 12;
P = pyulear(y,order,[]);
P = 10*log10(P);            % convert to DB
F = 0:1/128:1;              % normalized frequency axis

% call pickmax to pick the peaks in the PSD
% Pm is the value of the peaks, I is the index of the peaks
[Pm,I] = pickmax(P);
I = I/128;                  % normalize the index

% you should use plot(F, P) to plot the PSD
% and I tells you the location of those formant lines.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The following is also code provided by Dr. Qi
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Function: pick the index of local maxima
%
function [Y, I] = pickmax(y)

% pick the first 5 picks
Y = zeros(5,1);
I = zeros(5,1);

% get the difference
xd = diff(y);

% pick the index where the difference goes from + to -
% this is the local maxima
index = 1;
pos = 0;
for i=1:length(xd)
  if xd(i)>0
    pos = 1;
  else
    if pos==1
      pos = 0;
      Y(index) = xd(i);
      I(index) = i-1;
      index = index + 1;
      if index>5
        return
      end
    end
  end
end
```

APPENDIX F.

```
%Code to sort and compare voice files.  This code first compares the
%reference wav file to all others based on average pitch.  The top 12
%most likely matches are then compared by the differences in their
&formant peak vectors.  The resulting closest matches are then
%displayed.  This code needs pitch.m, pitchacorr.m, formant.m, and
%pickmax.m in the same directory in order to run.

%Author = E. Darren Ellis 05/01


results=zeros(12,1);          %create a vector for results.
diff=zeros(82,1);           %create a vector for differences in pitch.
formantdiff=zeros(12,1);   %create a vector for diff in formant vector

[y17, fs17, nbits17] = wavread('a17.wav');   %read in the wav file to
                                             %compare all others to.

[t17, f017, avgF017] = pitch(y17,fs17);      %call the pitch rouine for
                                             %ref. wav file.
[P17,F17,I17] = formant(y17);                %call the formant routine
                                             %for ref. wav file.

plot(t17,f017)                %plot the pitch contour of the ref. file
avgF17 = avgF017              %set the average pitch equal to avg17
sound(y17)

   pause(3)                                  %pause for 3 seconds


%This code was provided by Dr. Qi
%file name based on the index, i
   for i=1:83
      if i<10
         filename = sprintf('a0%i.wav', i);
      else
         filename = sprintf('a%i.wav', i);
  end

 [y, fs, nbits] = wavread(filename);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   [t, f0, avgF0] = pitch(y,fs);    %call the pitch.m routine for the
                                    %current wav file.

   plot(t,f0)                 %plot the current wav file contour plot.

   avgF0(i) = avgF0; %find the average pitch for the current wav file.

   diff(i,1)=norm(avgF0(i)-avgF17); %create a vector of avg. pitch diff
                                    %between current wav file and
                                    %reference wav file.
```

```
i                     %display the index to see where the comparison is.

end


[Y,H]=sort(diff)  %sort the pitch correlations in ascending order.

for j=1:12 %pick the lowest 20 pitch correlations to compare formants .

   p=H(j)         %set p equal to jth position of vector H  .
      if p<10
         filename = sprintf('a0%i.wav', p);
      else
         filename = sprintf('a%i.wav', p);
   end

filename       %display the filename of the wav file being compared.

[y, fs, nbits] = wavread(filename);
[P,F,I] = formant(y);   %call the formant.m routine for the current wav
.
sound(y)                 %play back the wav file being compared.
plot(F,P)                %plot the formants for the comparison wav file.
pause(3)       %pause for 3 seconds so sound will finish playing back.
formantdiff(j,1)=norm(I17-I); %create a vector of formant peak
                              %differences.
end
[Y1,H1]=sort(formantdiff)  %sort the vector in ascending order

for k=1:12
   results(k,1)=H(H1(k));  %calculate the numerical numbers of the
                           %closest wav matches.

end

H     %display the vector H.
H1    %display the vector H1.
results  %display the numerical numbers of the closest wav file
         %matches.
```