Final Lab: Speaker Detection

Lydia Tollerson & Benjamin Roeder

4/23/25

# Abstract

This lab focuses on the creation of a voice detection algorithm that uses filtering methods and comparisons (“profiles”) to determine certain voices from others. After vocal samples of intended “nominal” voices and unauthorized “intruder” voices are recorded, they were be passed through the program, which would output whether or not the vocal sample is authorized or not. The conclusion of the DSP pipeline was that the algorithm achieved a **95%** for recognizing nominal voices, and **93%** for recognizing intruders.

# Introduction

The purpose of this lab is to create a program that takes audio input from a multitude of different voices and makes a distinction between them by cleaning up each individual file by removing back ground noise via FIR filter, running it through additional filters designed around an FFT function, and detecting a specific threshold of amplitudes to distinguish one voice from another.

It should be noted that all recordings were manually loaded in code via the audioread() function. Usage of the console for inputs was only required when finding out if a post-processed recording was authorized or not. Batch processing was used in the forms of loops for all audio recordings.

# Methodology

**Data Collection**

From a total of 15 different people, a total of 65 vocal samples were recorded, with everyone speaking the same phrase: “The quick brown fox jumps over the lazy dog”. 10 individuals were chosen as the “nominal” vocal sample group, and the other 5 were labeled as “intruders”. Each nominal was asked to record 5 samples each, while each intruder was asked to record only 3, as the intruders were the voices that the algorithm would be trained to reject, or pinpoint as intruders.

Additionally, the background noise of the recording room was taken in an effort to remove as much noise from the original voice recording as possible.

**Input Processing**

Once all samples were recorded, they were all run through a number of specific filters to ensure consistency. All audio was converted from stereo to mono. This is due to the matrices being an L x 2 measurement initially. Processing a linear row (L x 1) is computationally easier to navigate.

A screenshot of a computer code

AI-generated content may be incorrect.

Figure ; code section detailing the conversion from stereo sound to mono.

Additionally, utilizing the recorded silent room noise (which was also converted from stereo to mono), each audio file ensured that any background frequencies were filtered out using an FIR filter for smoother processing and recognition.

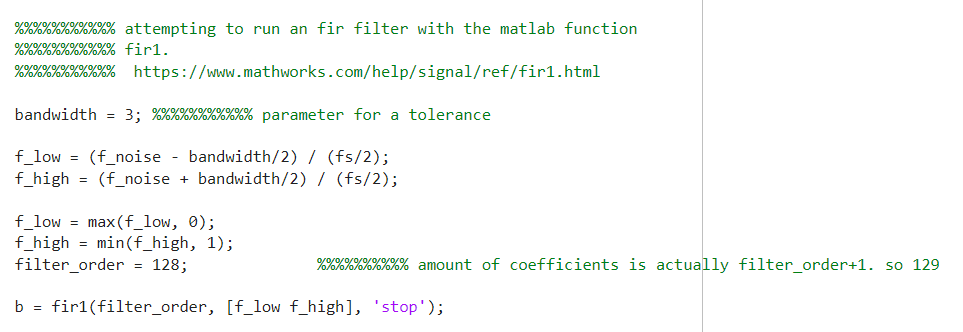


Figure ; code section detailing the construction of the background frequency FIR filter.

Each audio file is processed and given a new naming convention to distinguish it as a “clean” version, rather than the original unprocessed recording. The golden copies were not modified other than transforming them into a mono version.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure ; code section detailing the loop for filtering and renaming each audio file.

Then the magnitude of each sound file is isolated. This was selected as the best aspect of each signal that can be quantified and compared to create the required distinction between each voice. A lot of time was spent attempting to find fundamental frequencies for each speaker, but this wasn’t found to be constructive.

**A screenshot of a computer

AI-generated content may be incorrect.**

Figure ; code section for finding average magnitudes of each cleaned file.

**Algorithm**

The algorithm takes a struct comprised of each cleaned nominal audio files (named “profiles” for authorized speakers) and begins to parse it in two nested loops. The outer loop determines which speaker is being averaged, and the inner loop averages all amplitudes of all five recordings of that one speaker. After the average is found for each nominal, this is when the main zero padding begins, ensuring that each file is the same length.

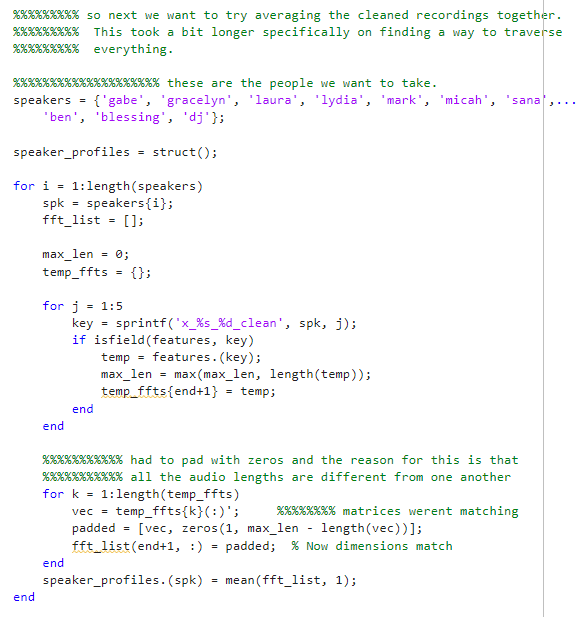


Figure ; code snapshot of the main sound file averaging loop as well as the zero padding.

From this point on in the code, the program waits for input in the console for any specific audio file to be named to be processed. Once a specific audio file is typed in correctly, the distance between the maximum and minimum amplitude in the audio file is determined and compared to the averaged data. Using a carefully calculated threshold of **8.5**, the voice is identified and matched as either a nominal voice or an intruder. The threshold was refined and adjusted based on detection of profiles with the test console output later listed in this report.

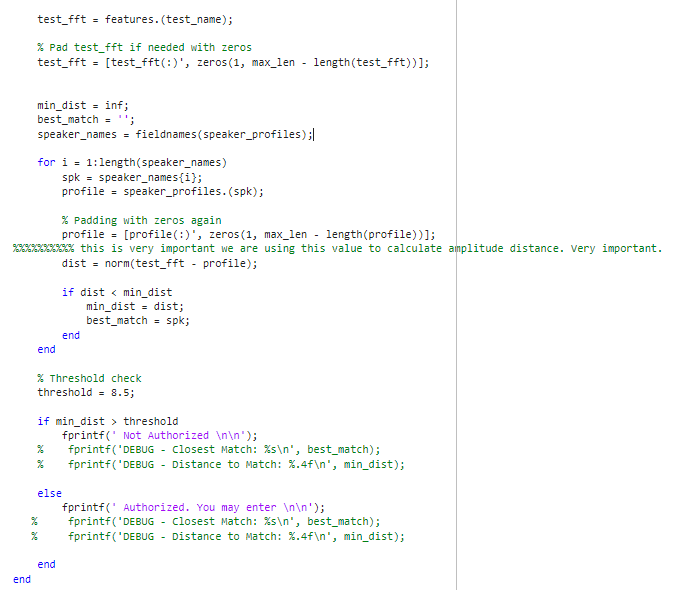


Figure ; code snapshot of the main comparison section of the algorithm.

The only other addition to the code is a debug feature that does the same action as the section described but applies it to every single sound file within the directory, laying out a large spread of results for every sound file. This was extremely important in debugging and specifically refining an appropriate threshold value.

# Results

Utilizing the algorithm’s capability to display all sound files and their detection as authorized or rejected, the following data spread is displayed:

Intended intruder voices are highlighted in **orange**, and incorrect deductions are highlighted in **red**.

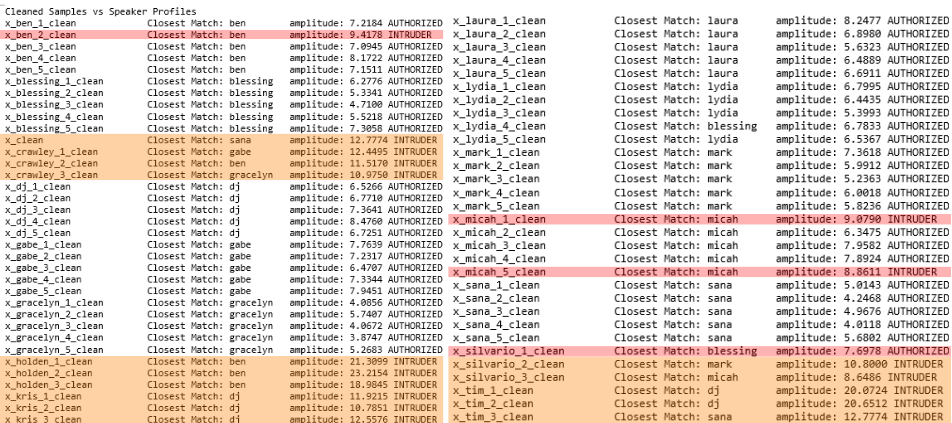


Figure ; direct console output of the algorithm, with color coded annotations.

With all 65 samples recorded, cleaned, and processed, the total percentage of success for the chosen comparison threshold is **95%** for recognizing nominal voices, and **93%** for recognizing intruders.

In retrospect as of the writing of this report, it would have been better to heed the initial directive which was to **first use 3 out of 5 of the voice recordings for the profile, and then use 2 additional recordings to determine if people were authorized or not.** This would have reinforced the concept of a ‘learning’ based model and not a ‘profile’ based model that was achieved. This could have further reduced the false negatives and false positives that were encountered with data. This would have drastically affected the validation process. Given additional time, this would have been the one significant change to implement that would have likely increased the accuracy of detection for both nominal voices and intruders.

# Conclusion

The lab served as a culmination of the foundation of signals lab over the past four months. The application of an FIR filter was significant choice in removing background noise as the majority of other groups after presentation day didn’t even factor this into their pipeline. Running FFTs across the different recordings helped in determine profiles for authorized speakers. Achieving a high but imperfect recognition rate of both nominal and intruders was a very surprising outcome for this lab.

# Appendix

**final.m**

%%%%%%%%%%%%%% " the quick brown fox jumps over the lazy dog

%%%%%%%%%%%%%%%%%% so far we have 95% accuracy rating

%%%%%%%%% first load all audio

[x\_gabe\_1, fs] = audioread('gabe\_1.wav');

[x\_gabe\_2, ~] = audioread('gabe\_2.wav');

[x\_gabe\_3, ~] = audioread('gabe\_3.wav');

[x\_gabe\_4, ~] = audioread('gabe\_4.wav');

[x\_gabe\_5, ~] = audioread('gabe\_5.wav');

[x\_gracelyn\_1, ~] = audioread('gracelyn\_1.wav');

[x\_gracelyn\_2, ~] = audioread('gracelyn\_2.wav');

[x\_gracelyn\_3, ~] = audioread('gracelyn\_3.wav');

[x\_gracelyn\_4, ~] = audioread('gracelyn\_4.wav');

[x\_gracelyn\_5, ~] = audioread('gracelyn\_5.wav');

[x\_laura\_1, ~] = audioread('laura\_1.wav');

[x\_laura\_2, ~] = audioread('laura\_2.wav');

[x\_laura\_3, ~] = audioread('laura\_3.wav');

[x\_laura\_4, ~] = audioread('laura\_4.wav');

[x\_laura\_5, ~] = audioread('laura\_5.wav');

[x\_lydia\_1, ~] = audioread('lydia\_1.wav');

[x\_lydia\_2, ~] = audioread('lydia\_2.wav');

[x\_lydia\_3, ~] = audioread('lydia\_3.wav');

[x\_lydia\_4, ~] = audioread('lydia\_4.wav');

[x\_lydia\_5, ~] = audioread('lydia\_5.wav');

[x\_mark\_1, ~] = audioread('mark\_1.wav');

[x\_mark\_2, ~] = audioread('mark\_2.wav');

[x\_mark\_3, ~] = audioread('mark\_3.wav');

[x\_mark\_4, ~] = audioread('mark\_4.wav');

[x\_mark\_5, ~] = audioread('mark\_5.wav');

[x\_micah\_1, ~] = audioread('micah\_1.wav');

[x\_micah\_2, ~] = audioread('micah\_2.wav');

[x\_micah\_3, ~] = audioread('micah\_3.wav');

[x\_micah\_4, ~] = audioread('micah\_4.wav');

[x\_micah\_5, ~] = audioread('micah\_5.wav');

[x\_sana\_1, ~] = audioread('sana\_1.wav');

[x\_sana\_2, ~] = audioread('sana\_2.wav');

[x\_sana\_3, ~] = audioread('sana\_3.wav');

[x\_sana\_4, ~] = audioread('sana\_4.wav');

[x\_sana\_5, ~] = audioread('sana\_5.wav');

[x\_ben\_1, ~] = audioread('ben\_1.wav');

[x\_ben\_2, ~] = audioread('ben\_2.wav');

[x\_ben\_3, ~] = audioread('ben\_3.wav');

[x\_ben\_4, ~] = audioread('ben\_4.wav');

[x\_ben\_5, ~] = audioread('ben\_5.wav');

[x\_blessing\_1, ~] = audioread('blessing\_1.wav');

[x\_blessing\_2, ~] = audioread('blessing\_2.wav');

[x\_blessing\_3, ~] = audioread('blessing\_3.wav');

[x\_blessing\_4, ~] = audioread('blessing\_4.wav');

[x\_blessing\_5, ~] = audioread('blessing\_5.wav');

[x\_dj\_1, ~] = audioread('dj\_1.wav');

[x\_dj\_2, ~] = audioread('dj\_2.wav');

[x\_dj\_3, ~] = audioread('dj\_3.wav');

[x\_dj\_4, ~] = audioread('dj\_4.wav');

[x\_dj\_5, ~] = audioread('dj\_5.wav');

%%%%%%%%%% intruder audio. we need to reject these.

[x\_silvario\_1, ~] = audioread('silvario\_1.wav');

[x\_silvario\_2, ~] = audioread('silvario\_2.wav');

[x\_silvario\_3, ~] = audioread('silvario\_3.wav');

[x\_crawley\_1, ~] = audioread('crawley\_1.wav');

[x\_crawley\_2, ~] = audioread('crawley\_2.wav');

[x\_crawley\_3, ~] = audioread('crawley\_3.wav');

[x\_holden\_1, ~] = audioread('holden\_1.wav');

[x\_holden\_2, ~] = audioread('holden\_2.wav');

[x\_holden\_3, ~] = audioread('holden\_3.wav');

[x\_tim\_1, ~] = audioread('tim\_1.wav');

[x\_tim\_2, ~] = audioread('tim\_2.wav');

[x\_tim\_3, ~] = audioread('tim\_3.wav');

[x\_kris\_1, ~] = audioread('kris\_1.wav');

[x\_kris\_2, ~] = audioread('kris\_2.wav');

[x\_kris\_3, ~] = audioread('kris\_3.wav');

[noise\_background, ~] = audioread('empty\_room.wav');

%%%%%%%%% batch process the matrixes from stereo to mono. all the matrices

%%%%%%%%% are in a stereo format and mono closely represents how we

%%%%%%%%% physically speak. ie, we are not speaking from two mouths

vars = whos;

for i = 1:length(vars)

name = vars(i).name;

% all initial matrixes basically will be grabbed. that's why we went

% with the naming convention

if startsWith(name, 'x\_')

data = eval(name);

if ismatrix(data) && size(data, 2) == 2

mono = mean(data, 2);

assignin('base', name, mono);

end

end

end

%%%%%%%%%% background noise to mono for our first filter

if size(noise\_background, 2) == 2

noise\_background = mean(noise\_background, 2);

end

noise\_background = noise\_background / max(abs(noise\_background));

%%%%%%%%%%% filter background noise.

N = length(noise\_background);

Y = abs(fft(noise\_background));

Y = Y(1:floor(N/2));

f = linspace(0, fs/2, length(Y));

[~, idx] = max(Y);

f\_noise = f(idx); % we have 28.37hz

%%%%%%%%%%% attempting to run an fir filter with the matlab function

%%%%%%%%%%% fir1.

%%%%%%%%%%% https://www.mathworks.com/help/signal/ref/fir1.html

bandwidth = 3; %%%%%%%%%%% parameter for a tolerance

f\_low = (f\_noise - bandwidth/2) / (fs/2);

f\_high = (f\_noise + bandwidth/2) / (fs/2);

f\_low = max(f\_low, 0);

f\_high = min(f\_high, 1);

filter\_order = 128; %%%%%%%%%% amount of coefficients is actually filter\_order+1. so 129

b = fir1(filter\_order, [f\_low f\_high], 'stop');

%%%%%%%%%% clean everything now with FIR using another loop similar to

%%%%%%%%%% earlier one.

vars = whos;

for i = 1:length(vars)

name = vars(i).name;

if startsWith(name, 'x\_') && ~contains(name, '\_clean')

x = eval(name);

if size(x, 2) == 2

x = mean(x, 2);

end

x = x(:);

x = x / max(abs(x));

x\_clean = filter(b, 1, x);

x\_clean = x\_clean / max(abs(x\_clean));

clean\_name = [name '\_clean'];

assignin('base', clean\_name, x\_clean);

end

end

%%%%%%%%%%%%%%% grabs magnitude. we can't do voice recognition on

%%%%%%%%%%%%%%% frequency alone unfortunately. I spent a significant

%%%%%%%%%%%%%%% amount of time concluding this.

features = struct();

vars = whos;

for i = 1:length(vars)

name = vars(i).name;

if startsWith(name, 'x\_') && endsWith(name, '\_clean') %%%%%% grabs the cleaned up ones and doesn't mess up golden copies

x = eval(name);

N = 2^nextpow2(length(x));

X = abs(fft(x, N));

X = X(1:floor(N/2));

X = X / max(X);

features.(name) = X;

end

end

%%%%%%%%% so next we want to try averaging the cleaned recordings together.

%%%%%%%%% This took a bit longer specifically on finding a way to traverse

%%%%%%%%% everything.

%%%%%%%%%%%%%%%%%%%% these are the people we want to take.

speakers = {'gabe', 'gracelyn', 'laura', 'lydia', 'mark', 'micah', 'sana',...

'ben', 'blessing', 'dj'};

speaker\_profiles = struct();

for i = 1:length(speakers)

spk = speakers{i};

fft\_list = [];

max\_len = 0;

temp\_ffts = {};

for j = 1:5

key = sprintf('x\_%s\_%d\_clean', spk, j);

if isfield(features, key)

temp = features.(key);

max\_len = max(max\_len, length(temp));

temp\_ffts{end+1} = temp;

end

end

%%%%%%%%%%% had to pad with zeros and the reason for this is that

%%%%%%%%%%% all the audio lengths are different from one another

for k = 1:length(temp\_ffts)

vec = temp\_ffts{k}(:)'; %%%%%%%% matrices werent matching

padded = [vec, zeros(1, max\_len - length(vec))];

fft\_list(end+1, :) = padded; % Now dimensions match

end

speaker\_profiles.(spk) = mean(fft\_list, 1);

end

%%%%%%%%%%%%%%%%%%%%%%%% master console input %%%%%%%%%%%%

fprintf('\n Password required! \n');

while true

test\_name = input('Enter cleaned test sample name: ', 's');

if strcmpi(test\_name, 'exit')

disp('exiting');

break;

end

if ~isfield(features, test\_name)

fprintf(' audio recording "%s" not found. \n\n', test\_name);

continue;

end

test\_fft = features.(test\_name);

% Pad test\_fft if needed with zeros

test\_fft = [test\_fft(:)', zeros(1, max\_len - length(test\_fft))];

min\_dist = inf;

best\_match = '';

speaker\_names = fieldnames(speaker\_profiles);

for i = 1:length(speaker\_names)

spk = speaker\_names{i};

profile = speaker\_profiles.(spk);

% Padding with zeros again

profile = [profile(:)', zeros(1, max\_len - length(profile))];

%%%%%%%%%% this is very important we are using this value to calculate amplitude distance. Very important.

dist = norm(test\_fft - profile);

if dist < min\_dist

min\_dist = dist;

best\_match = spk;

end

end

% Threshold check

threshold = 8.5;

if min\_dist > threshold

fprintf(' Not Authorized \n\n');

% fprintf('DEBUG - Closest Match: %s\n', best\_match);

% fprintf('DEBUG - Distance to Match: %.4f\n', min\_dist);

else

fprintf(' Authorized. You may enter \n\n');

% fprintf('DEBUG - Closest Match: %s\n', best\_match);

% fprintf('DEBUG - Distance to Match: %.4f\n', min\_dist);

end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% this was used for debugging and not

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% required for functionality

fprintf('\n Cleaned Samples vs Speaker Profiles\n');

sample\_names = fieldnames(features);

profile\_names = fieldnames(speaker\_profiles);

%max\_len = max([structfun(@length, speaker\_profiles), structfun(@length, features)]);

all\_lengths = [structfun(@length, speaker\_profiles); structfun(@length, features)];

max\_len = max(all\_lengths);

%max\_len = max(structfun(@length, speaker\_profiles));

threshold = 8.5; %%%%%%%%%%%%%% this was tweaked until we have our confidence range.

for i = 1:length(sample\_names)

sample = sample\_names{i};

test\_fft = features.(sample);

test\_fft = [test\_fft(:)', zeros(1, max\_len - length(test\_fft))];

min\_dist = inf;

best\_match = '';

for j = 1:length(profile\_names)

profile = speaker\_profiles.(profile\_names{j});

profile = [profile(:)', zeros(1, max\_len - length(profile))];

dist = norm(test\_fft - profile);

if dist < min\_dist

min\_dist = dist;

best\_match = profile\_names{j};

end

end

if min\_dist > threshold

result = 'INTRUDER';

else

result = 'AUTHORIZED';

end

fprintf('%-25s Closest Match: %-10s amplitude: %.4f %s\n', sample, best\_match, min\_dist, result);

end