# 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.

```
%%%%%%%%% 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
```

Figure 1; 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.

```
%%%%%%%%%%% 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');
```

Figure 2; 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.

```
%%%%%%%%%% 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
```

Figure 3; 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.

```
%%%%%%%%%%%%%%%  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
```

Figure 4; 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.

```
%%%%%%%%% 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
```

Figure 5; 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.

```
    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
```

Figure 6; 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**.

```
Cleaned Samples vs Speaker Profiles
x_ben_1_clean          Closest Match: ben        amplitude: 7.2184 AUTHORIZED
x_ben_2_clean          Closest Match: ben        amplitude: 9.4178 INTRUDER
x_ben_3_clean          Closest Match: ben        amplitude: 7.0945 AUTHORIZED
x_ben_4_clean          Closest Match: ben        amplitude: 8.1722 AUTHORIZED
x_ben_5_clean          Closest Match: ben        amplitude: 7.1511 AUTHORIZED
x_blessing_1_clean     Closest Match: blessing   amplitude: 6.2776 AUTHORIZED
x_blessing_2_clean     Closest Match: blessing   amplitude: 5.3341 AUTHORIZED
x_blessing_3_clean     Closest Match: blessing   amplitude: 4.7100 AUTHORIZED
x_blessing_4_clean     Closest Match: blessing   amplitude: 5.5218 AUTHORIZED
x_blessing_5_clean     Closest Match: blessing   amplitude: 7.3058 AUTHORIZED
x_clean                Closest Match: sana       amplitude: 12.7774 INTRUDER
x_crawley_1_clean      Closest Match: gabe       amplitude: 12.4495 INTRUDER
x_crawley_2_clean      Closest Match: ben        amplitude: 11.5170 INTRUDER
x_crawley_3_clean      Closest Match: gracelyn   amplitude: 10.9750 INTRUDER
x_dj_1_clean           Closest Match: dj         amplitude: 6.5266 AUTHORIZED
x_dj_2_clean           Closest Match: dj         amplitude: 6.7710 AUTHORIZED
x_dj_3_clean           Closest Match: dj         amplitude: 7.3641 AUTHORIZED
x_dj_4_clean           Closest Match: dj         amplitude: 8.4760 AUTHORIZED
x_dj_5_clean           Closest Match: dj         amplitude: 6.7251 AUTHORIZED
x_gabe_1_clean         Closest Match: gabe       amplitude: 7.7639 AUTHORIZED
x_gabe_2_clean         Closest Match: gabe       amplitude: 7.2317 AUTHORIZED
x_gabe_3_clean         Closest Match: gabe       amplitude: 6.4707 AUTHORIZED
x_gabe_4_clean         Closest Match: gabe       amplitude: 7.3344 AUTHORIZED
x_gabe_5_clean         Closest Match: gabe       amplitude: 7.9451 AUTHORIZED
x_gracelyn_1_clean     Closest Match: gracelyn   amplitude: 4.0856 AUTHORIZED
x_gracelyn_2_clean     Closest Match: gracelyn   amplitude: 5.7407 AUTHORIZED
x_gracelyn_3_clean     Closest Match: gracelyn   amplitude: 4.0672 AUTHORIZED
x_gracelyn_4_clean     Closest Match: gracelyn   amplitude: 3.8747 AUTHORIZED
x_gracelyn_5_clean     Closest Match: gracelyn   amplitude: 5.2683 AUTHORIZED
x_holden_1_clean       Closest Match: ben        amplitude: 21.3099 INTRUDER
x_holden_2_clean       Closest Match: ben        amplitude: 23.2154 INTRUDER
x_holden_3_clean       Closest Match: ben        amplitude: 18.9845 INTRUDER
x_kris_1_clean         Closest Match: dj         amplitude: 11.9215 INTRUDER
x_kris_2_clean         Closest Match: dj         amplitude: 10.7851 INTRUDER
x_kris_3_clean         Closest Match: dj         amplitude: 12.5576 INTRUDER
x_laura_1_clean        Closest Match: laura      amplitude: 8.2477 AUTHORIZED
x_laura_2_clean        Closest Match: laura      amplitude: 6.8980 AUTHORIZED
x_laura_3_clean        Closest Match: laura      amplitude: 5.6323 AUTHORIZED
x_laura_4_clean        Closest Match: laura      amplitude: 6.4889 AUTHORIZED
x_laura_5_clean        Closest Match: laura      amplitude: 6.6911 AUTHORIZED
x_lydia_1_clean        Closest Match: lydia      amplitude: 6.7995 AUTHORIZED
x_lydia_2_clean        Closest Match: lydia      amplitude: 6.4435 AUTHORIZED
x_lydia_3_clean        Closest Match: lydia      amplitude: 5.3993 AUTHORIZED
x_lydia_4_clean        Closest Match: blessing   amplitude: 6.7833 AUTHORIZED
x_lydia_5_clean        Closest Match: lydia      amplitude: 6.5367 AUTHORIZED
x_mark_1_clean         Closest Match: mark       amplitude: 7.3618 AUTHORIZED
x_mark_2_clean         Closest Match: mark       amplitude: 5.9912 AUTHORIZED
x_mark_3_clean         Closest Match: mark       amplitude: 5.2363 AUTHORIZED
x_mark_4_clean         Closest Match: mark       amplitude: 6.0018 AUTHORIZED
x_mark_5_clean         Closest Match: mark       amplitude: 5.8236 AUTHORIZED
x_micah_1_clean        Closest Match: micah      amplitude: 9.0790 INTRUDER
x_micah_2_clean        Closest Match: micah      amplitude: 6.3475 AUTHORIZED
x_micah_3_clean        Closest Match: micah      amplitude: 7.9582 AUTHORIZED
x_micah_4_clean        Closest Match: micah      amplitude: 7.8924 AUTHORIZED
x_micah_5_clean        Closest Match: micah      amplitude: 8.8611 INTRUDER
x_sana_1_clean         Closest Match: sana       amplitude: 5.0143 AUTHORIZED
x_sana_2_clean         Closest Match: sana       amplitude: 4.2468 AUTHORIZED
x_sana_3_clean         Closest Match: sana       amplitude: 4.9676 AUTHORIZED
x_sana_4_clean         Closest Match: sana       amplitude: 4.0118 AUTHORIZED
x_sana_5_clean         Closest Match: sana       amplitude: 5.6802 AUTHORIZED
x_silvario_1_clean     Closest Match: blessing   amplitude: 7.6978 AUTHORIZED
x_silvario_2_clean     Closest Match: mark       amplitude: 10.8000 INTRUDER
x_silvario_3_clean     Closest Match: micah      amplitude: 8.6486 INTRUDER
x_tim_1_clean          Closest Match: dj         amplitude: 20.0724 INTRUDER
x_tim_2_clean          Closest Match: dj         amplitude: 20.6512 INTRUDER
x_tim_3_clean          Closest Match: sana       amplitude: 12.7774 INTRUDER
```

Figure 7; 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
```