EE 430 PROJECT FINAL REPORT

Sabri Bolkar (1949874) - Selim Borekci (1813831)
Section 1 - Section 3

25 January 2015

Contents

List of Figures

Listings

1 Introduction

In Digital Signal Processing term project, we are assigned to create a comprehensive audio processing system consisting of data acquisition from audio file or directly from microphone, spectrogram analyzer, equalizer, sound affect mixer and remover systems. In the first phase, data acquisition and spectrogram parts are implemented, and in the second phase remaining tasks are completed. In addition to main programs, graphical user interfaces are also designed, even in the second phase all the main code is functionalized and embedded to the GUI, therefore no additional program .m file is written. In the fallowing parts all the work done including coding, GUI design and technical analysis is given with proper figures and listings.

2 Phase-1

2.1 Task 1-Data Acquisition

The acquisition part is readily implemented by MATLAB built-in functions, *audioread* for MP3 reading and *audiorecorder* for microphone reading. Also, *play* and *sound* commands easily enabled us to play the audio data.

```
Fs= 80000; % A/D sampling rate in Hz ----------CONFIGURABLE
T=5; % Record Time in seconds ----------CONFIGURABLE

recObj = audiorecorder(Fs,16,1); % 8 bit - 1 channel sound object
recordblocking(recObj, T);  %record T seconds
data = getaudiodata(recObj); %store sound data to "data"
play(recObj, Fs) %Play the sound

% MP3 data reading

name= 'feel.mp3' ; %% file name to be read
[data2, Fs]= audioread('feel.mp3'); % Get the data
sound('feel.mp3', Fs); %% PLay the sound
```

Listing 1: Data Acquisition Part

2.2 Task 2-Spectrogram

For spectrogram implementation we are restricted not to use built-in MATLAB function *spectrogram*; hence, we designed our own algorithm and program to implement spectrogram processing.

Firstly some research was done on spectrogram and short time Fourier transform. The process is not a cumbersome one but, there were some problems including how to implement the moving window function (Figure 1), which we choose as two kinds (i.e., Hamming and Rectangular). Also, parameters are required to be configurable which created another challenge. The chosen method is to move the window according to entered window size. We would like to design a 50% overlapping system to make sure not to miss any data, therefore half of the window size as hop size is chosen. In this way, we were able to solve the challenge to make sure no data is skipped. In subsequent parts, system multiplies the data points with window function and calculates DFT with MATLAB's built-in *fft* command (using entered DFT points) in a straightforward manner.

```
% window1 Hamming
win1 = hamming(wind_length, 'periodic');% ----------CONFIGURABLE
% window2 Rect
win2 = ones(wind_length,1); %----------CONFIGURABLE
% Spect will be the spectrogram
rown = ceil((1+ndft)/2);
coln = 1+fix((data_length-wind_length)/hop);
spect = zeros(rown, coln);
indx = 0;
col = 1;
%%%% DFT calculation
while indx + wind_length <= data_length
    % windowing
    windowed_data = data(indx+1:indx+wind_length).*win1; %%%%% CHANGE WINDOW FUNC

    % FFT
    X = fft(windowed_data, ndft);

     spect(:, col) = X(1:rown);

    indx = indx + hop;
    col = col + 1;
end

t = (wind_length/2:hop:wind_length/2+(coln-1)*hop)/Fs;
f = (0:rown-1)*Fs/ndft;
spect= 20*log10(abs(spect));  %%% TAKING DB of Spectrogram data
```

Listing 2: Spectrogram Part

2.3 Graphical User Interface-1

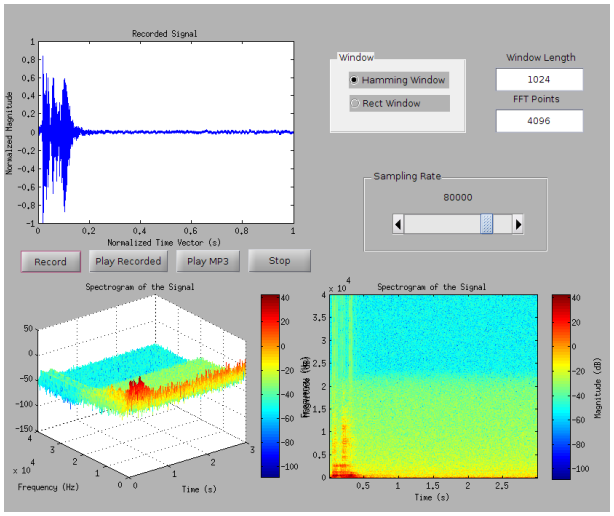

Figure 1: Designed Graphical User Interface (Linux Ubuntu Version)

3 Phase-2

In this part of the project, we have completed more serious and practical tasks namely a implementation of real-time spectrum, audio equalizer, sound effect, audio noise interference. All the details including MATLAB codes and explanations can be found in below sections.

3.1 Task 3-Real-Time Spectrum

It is aimed to create a system taking input sound within the determined period of time, then executing DFT for each part and piloting these results with *drawnow* command. Sampling of the sound signal, and capture time is designed to be configurable. An example output for Sabri's coughing and code snippet can be viewed below in figure-1 and listing-1.

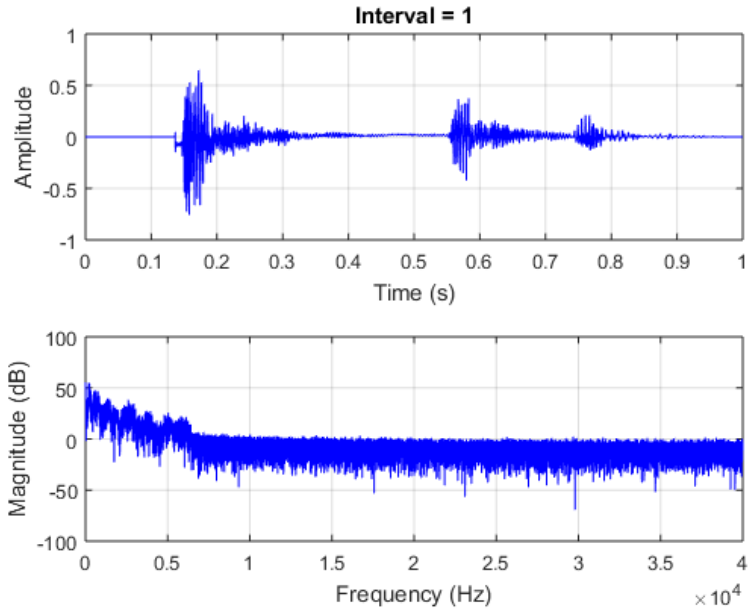

Figure 2: Real-Time Audio Processing

```
Fs = 80000;                         % Sampling frequency ------- CONFIGURABLE
T = 1;                              % length of an interval ------- CONFIGURABLE
N = 5;                              % Number of interval ------- CONFIGURABLE
record_time=N*T;
t = 0:1/Fs:T-1/Fs;                  % time values
nfft = 2^nextpow2(Fs);              % n-point DFT
halved = ceil((nfft+1)/2);          % half poimt
f = (0:halved-1)'*Fs/nfft;          % Frequency values

%%% PLOTS
figure
hAx(1) = subplot(211);
```

```
hLine(1) = line('XData',t, 'YData',nan(size(t)), 'Color','b', 'Parent',hAx(1));
xlabel('Time (s)'), ylabel('Amplitude')
hAx(2) = subplot(212);
hLine(2) = line('XData',f, 'YData',nan(size(f)), 'Color','b', 'Parent',hAx(2));
xlabel('Frequency (Hz)'), ylabel('Magnitude (dB)')
set(hAx, 'Box','on', 'XGrid','on', 'YGrid','on')

%%%%% FFT calculations

recObj = audiorecorder(Fs,8,1);  %%%%% Creating Audio Object

% N*T seconds recording
disp('Start speaking NOW...')
for i=1:N
    recordblocking(recObj, T);      %% Time to Record (1 second record)

    sig = getaudiodata(recObj);   %%%% Transfer auido DATA to MATRIX data
    fftMag = 20*log10( abs(fft(sig,nfft)) );    %%% FFT TIME in db

    set(hLine(1), 'YData',sig)      %% Plot modifications
    set(hLine(2), 'YData',fftMag(1:halved))
    title(hAx(1), num2str(i,'Interval = %d'))
    drawnow           %%% UPDATE THE PLOT
end

disp('You shall STOP now.')
record_time                     % displays the recorded time
```

Listing 3: Real-time spectrum

3.2 Task 4-Audio Equalizer

In this part, we are asked to devise an audio equaliser system in software environment. It is also stated that 10 different frequency bands in the audio range should be suppressed or amplified according to graphical user interface configuration. Our system satisfies this task for both re cored signal, which record-time can be configured via GUI, and MP3 signal.

Bandpass filters are designed to be linear phase filters so that distortion can be minimized and if required signal can be recovered by just time shift. The method implemented is quite straightforward, after FFT process all the data is multiplied with an array of coefficients having no phase component. In that array, the bands to be amplified is multiplied by A, whose value depends on GUI, and other parts are not changed. Also to create a linear phase additional phase vector is multiplied with the filtered data. Desired edge frequencies were as fallowing $32, 64, 125, 250, 500, 1000, 2000, 4000, 8000, 16000Hz$. Code for filter can be viewed in the listing-2, also whole GUI code can be found in the attached file.

```
function myFilter = createFilter(dB, L)  %%%% LINEAR PHASE FILTER FUNCTION

w_array = [0,32,64,125,250,500,1000,2000,4000,8000,16000]; %%% Desired edge
    frequencies

w = 32000*pi*(0:(L-1))/L; % L samples from 0 to 2pi, w is the frequency values
Ad = ones(1,L); %%% DEFAULT COEFFICIENT ARRAY

for x = 1:10    %% for 10 different Band
w1 = 2*pi*w_array(x);   % First Cut-off frequency of the selected band
```

```
w2 = 2*pi*w_array(x+1); % Second Cut-off frequency
A = 10^(dB(x)/20);  %% Input is DB hence convert them first

%%%% FILTERING

    for m = 1:L
        if (w(m) <= w2) && (w(m) >= w1) % If frequency is less than cutoff frequency
            Ad(m) = A;
        end
    end
end

k = 0:L-1;
p = exp(-i*w*1000*(L-1)/L); % Multiplies Phase vector
myFilter = (Ad .* p)'; % Sampled frequency responce
```

Listing 4: Linear Phase Filter Snippet

3.3 Task 5-Special Effects

3.3.1 Reverberation

Reverberation effect is a sound effect related with the echoes of the original audio signal. It occurs when original signal reflects from different objects, it's result is delayed versions of the signals lesser in amplitude which decays in time (1). A sample block diagram can be seen in figure-2. This effect can be created with a feedback system where delayed signal is fed back to the original signal; hence, because of feedback system function will have poles (or a single pole) which results in IIR filter characteristics. The code snipped for the reverberating can be observed in the below listing.

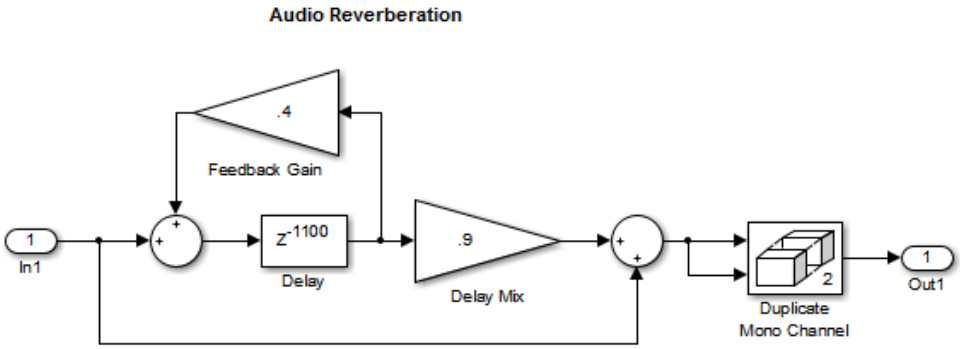

Figure 3: Reverberation effect block diagram (1)

```
function out_data = reverb(in_data, Fs, decay_rate, fir_delay, iir_delay)

delay1 = round(Fs*fir_delay); % FIR Delay
delay2 = round(Fs*iir_delay); % IIR Delay

%%% Filtering using FILTER command
out_data = filter([1 zeros(1,delay1) decay_rate],[1 zeros(1,delay2) -decay_rate],
    in_data);

%%% Nominator [1 zeros(1,delay1) decay_rate]
%%% Denominator [1 zeros(1,delay2) -decay_rate]
```

Listing 5: Reverb-Effect

3.3.2 Synthetic Stereo

Stereophonic sound is a famous technique used to create two-dimensional artificial audio perspective. To create such effect mono audio signal is delayed and sent to the second channel audio amplifier (1). Stereo devices are two-output (audio) devices where each output is reserved for mono signals. Code part for the Stereo effect can be seen in listing-4.

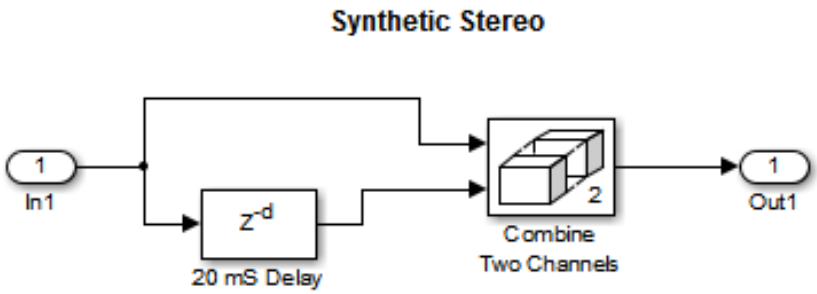

Figure 4: Synthetic Stereo Code (1)

```
function out_data = synthetic_stereo(in_data, Fs, stereo_delay)

delay = round(Fs*stereo_delay); %% Delay is the multiplative value*Fs

delayed_data = filter([zeros(1,delay) 1],[1],in_data); %% Using FILTER command
%%% Note that it's an FIR filter hence denominator is unity

out_data = [in_data delayed_data]; %%% Two channel data is created for the
    AudioObject
```

Listing 6: Stereo Effect Code

3.3.3 Changing Speed of the Audio

```
set(handles.stop,'Enable','off');
data = guidata(hObject);

%%% CHANGIND THE SOUND PACE
Fs = data.Fs*data.speed;    %% Calculate the new Fs
data.out_data = data.in_data;

%%% REVERBERATION
if data.reverb_check == 1
    data.out_data = reverb(data.out_data, Fs, data.decay_rate, data.fir_delay, data.
        iir_delay);
end

%%% STEREO EFFECT
if data.stereo_check == 1
    data.out_data = synthetic_stereo(data.out_data, Fs, data.stereo_delay);
end
soundsc(data.out_data, Fs);
set(handles.stop,'Enable','on');
```

Listing 7: Playing with Audio Pace

3.4 Task 6-Noise Removal

For pure noise addition, a single frequency cosine having same data points is added to the sound signal. In noise removal, we have utilized the same filter that we used in equalizer, but this time it is easier. Assuming high frequency noise interference (i.e., $f_n > 20kHz$) signal can be recovered with a simple low pass filter with cut-off frequency 20 kHz.

3.5 Graphical User Interfaces

In total, two graphical user interfaces designed for the second phase (i.e., Equalizer GUI and Special effects GUI). Both of the GUIs have large code blocks; hence, they are not included in the appendix, but can be found in the attachment where all the work done is stated. Sample figures for both GUIs can be seen above figure-4 and below figure-5.

It was the hardest part to create the GUIs for we have no experience. But, we were able to handle the problems by the help of sample MATLAB guides and videos.

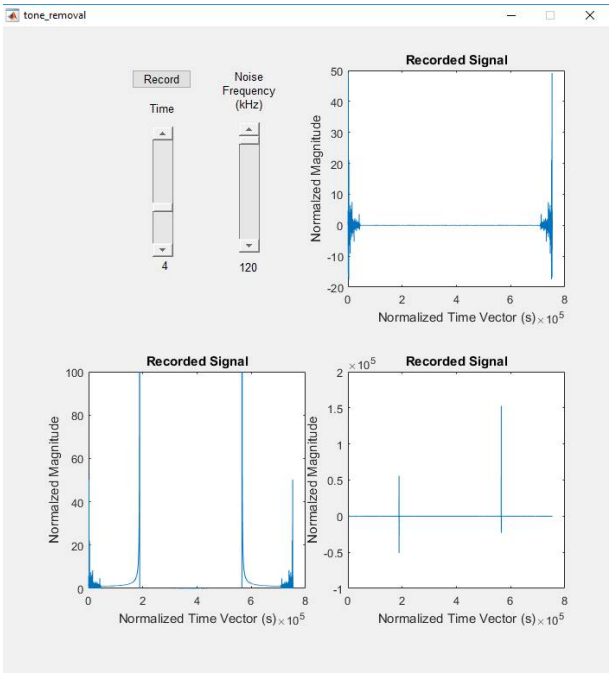

Figure 5: Noise added, pure and removed signals with designed GUI

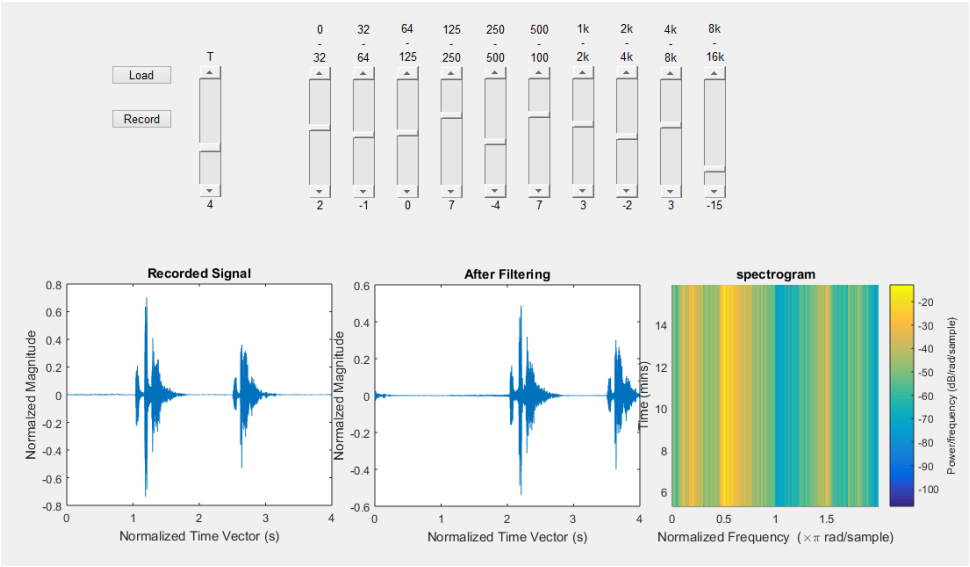

Figure 6: Equalizer GUI

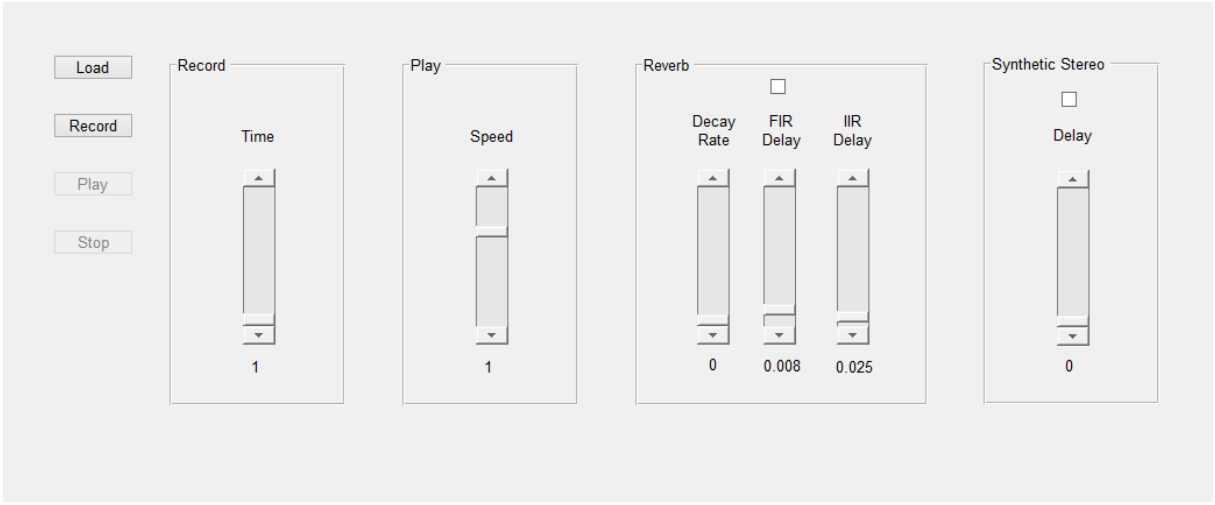

Figure 7: Special Effects GUI

4 Conclusion

During project design and implementation, we had the chance to combine all our DSP knowledge for practical purposes. Audio processing is still quite popular and modern tools are emerging for SOC and FPGA systems. Our design is not an optimized one, but still it assisted us as a starting point in Digital signal processing research field.

It was a great experience for us to implement DFT for a real-world process. In the first phase, after we compared our program with built-in *spectrogram* command, results amazed us with their almost same outcome for same sampling rate. In below figure, the similarity can be observed. We also observed that, hamming window gives better results for the analysis by smoothing the edges which window is applied to. DFT points and window length have also effects on spectrogram results. But, it can be seen as a trade-off. If a longer window length is chosen, although transform is more accurate the response time is longer which is not desired for some applications where quick analysis is needed.

In the second phase, we have designed various filters in particular for equalizing purposes. Also, we practiced GUI design with two applicable graphical user interfaces. But, the GUI for the noise effect did not work in all computers because of its inherent sound card dependent structure. When it is designed in the computer lab, it was operating just fine, but in my computer it does not work. Later, we realized that my computers hardware was not powerful enough to handle sound processing.

5 References

[1] MATHWORKS, MATLAB (2015), Retrieved from http://www.mathworks.com/help/dsp/examples/audio-special-effects.html

6 Appendix

```
%%%%%% PHASE-1 CODES

%% STEP 1

%%%%% Microphone reading

Fs= 80000; % A/D sampling rate in Hz ----------CONFIGURABLE
T=5; % Record Time in seconds ----------CONFIGURABLE

recObj = audiorecorder(Fs,16,1); % 8 bit - 1 channel sound object
recordblocking(recObj, T);  %record T seconds
data = getaudiodata(recObj); %store sound data to "data"

play(recObj, Fs) %Play the sound

%%
%%%%%%%%%%%%%%%%%%%%%
% MP3 data reading

%  name= 'feel.mp3' ; %% file name to be read

%  [data2, Fs]= audioread('feel.mp3'); % Get the data

%  sound('feel.mp3', Fs); %% PLay the sound

%%
%%%%%%%%%%%%%%%%%%%%%
% SPECTROGRAM

% Sample audio data FOR CHECKING
%[data, Fs] = audioread('Applause.mp3');

%%%% CHANGE data with data2 for mp3 processing

data_max = max(abs(data));
data = data/data_max;   %% Normalization

% Default input parameters
data_length = length(data);
wind_length = 1024;    %%% ----------CONFIGURABLE
hop = wind_length/2;     %% hopsize; take windows as 50% overlap  ----------
    CONFIGURABLE
ndft = 4096;          %% fft points ----------CONFIGURABLE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% DFT

% window1 Hamming
win1 = hamming(wind_length, 'periodic');% ----------CONFIGURABLE

% window2 Rect
win2 = ones(wind_length,1); %----------CONFIGURABLE

% Spect will be the spectrogram
rown = ceil((1+ndft)/2);
coln = 1+fix((data_length-wind_length)/hop);
```

```
spect = zeros(rown, coln);

indx = 0;
col = 1;

%%%% DFT calculation
while indx + wind_length <= data_length
    % windowing
    windowed_data = data(indx+1:indx+wind_length).*win1; %%%%% CHANGE WINDOW FUNC

    % FFT
    X = fft(windowed_data, ndft);


    spect(:, col) = X(1:rown);

    indx = indx + hop;
    col = col + 1;
end


t = (wind_length/2:hop:wind_length/2+(coln-1)*hop)/Fs;
f = (0:rown-1)*Fs/ndft;

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

spect= 20*log10(abs(spect));  %%% TAKING DB of Spectrogram data

%Play the sound
% sound(data)

%%
% PLOTING %%%%%%%%%%%%%%%%%%%%%%%%%

%plot the sound data

subplot(3,1,1)
Tx=linspace(0, T, length(data));
plot(Tx, data);
set(gca,'YDir','normal')
xlabel('Time (s)')
ylabel('Sample Magnitude')
title('Recorded Signal')

%%% plot the spectrogram

%%%% 3D
subplot(3,1,2)
h=surf(t, f, spect); colorbar;
set(h,'LineStyle','none') %% turn linestyle off to see the graph
set(gca,'YDir','normal')
xlabel('Time (s)')
ylabel('Frequency (Hz)')
title('Spectrogram of the Signal')

handle = colorbar;
ylabel(handle, 'Magnitude (dB)')
```

```
%%%% 2D
subplot(3,1,3)
imagesc(t, f, spect) %%% Plot it with color info

set(gca,'YDir','normal')
xlabel('Time (s)')
ylabel('Frequency (Hz)')
title('Spectrogram of the Signal')

handle = colorbar;
ylabel(handle, 'Magnitude (dB)')

%%
```

Listing 8: Phase-1 All Codes