EE430 Lab 4 Writeup

Section I – Narrative

Problem 1)

i) A MATLAB function "decodeDTMFblock.m" was created with the purpose of being able to take an array of audio input, and then return a string stating which DTMF symbol is present, or to show nothing in the event that it was unrecognized. This was largely created using the previously designed "decodeDTMF" code from last lab and stripping the power analysis from it. The code is attached as part of section 4.

ii) The code from the first section was implemented into the provided "audio_loop_template.m" file. As the original file already had a loop built into it, it was decided that using the 'decodeDTMFblock.m' file should execute on each loop, with a display function afterwards that would push the symbol into the command window on each loop. This causes the symbol to refresh every block of time, and gives constant feedback as to the current and past values of the DTMF decoder. It was observed in execution that if a DTMF signal was played too loudly into a receiving microphone, the data would often be inaccurate. It is believed this is due to saturation occurring, giving false "spikes" of power in locations where they would normally be undetected. An example of how the output functions is included in the graphs section of this document.

Problem 2)

i) The file "lab4_modulated_audio.wav" was loaded into MATLAB, and then put through the spectrogram function in order to determine which frequencies were used in modulating the signal. It was observed that there were four different frequencies that had the most power when observed in this way: 500, 1500, 2500, and 3000 Hz. It was also noticed that there was around 800 Hz between distinct bands, making it the chosen frequency for the band limit.

ii) In order to demodulate the audio signals, the original signal was multiplied by a sine wave of the same frequency as one of the observed carrier frequency. This causes the desired wave to have an alias at the center frequency. The "new" center signal can then be isolated from the others by use of a low pass filter, assumed to be at 800 Hz. The spectrograms of each of the moved signals and their appearance after filtration are included in the graphs portion of this document.

iii) Each of the different signals was isolated, listened to, and transcribed. The transcriptions of each are included on the following page.

====Message 1=====

Right after school she was supposed to pick up her little cousin from kindergarten and drop her home, but she didn't show.

====Message 2=====

Coming up: the imminent threat of autonomous killer robots; they think it takes us into a world in which nobody wants to live.

====Message 3=====

But if the US and the UK don't want things to move ahead, uh, then they won't move ahead, as far I can tell.

====Message 4=====

"It would really be in the best interest of US-Soviet relations if you meant to give the ring as a present, Kraft said he was told on the White House call." [Simultaneously spoken:]"Awwwww" "What?!"

Section III – Source Code

Part 1:

```
fs=44100;
r=audiorecorder(fs,16,1);
blocktime=0.5; % length of blocks (s)
s=zeros(blocktime*fs,1);
while 1
    record(r);
    t1=clock;
    %%
    % process previously captured block of audio data
    %%
    % plot signal and FT of signal
    subplot(2,1,1);
    plot(s);
    subplot(2,1,2);
    freq=fs*((0:numel(s)-1))/numel(s);
    plot(freq,abs(fft(s)));
    % Further processing (you fill in)
    digit = decodeDTMFblock(s,fs);
    disp(digit)

    pause(blocktime-etime(clock,t1));
    pause(r);
    %% get audio data
    s=r.getaudiodata();
    stop(r);
end
```

Part 2:

```
[x,fs] = audioread('lab4_modulated_audio(1).wav');

%sound(x, fs);


% get a spectrogram of the audio file
%spectrogram(x,blackman(8000*0.02),0, [] , 8000, 'yaxis')

% Four different observed frequency carriers: 400,1400,2400, and 3400.
% Band limit appears to be 800 Hz.

%Now, apparently, the trick is to multiply the sinusoid of interest by a
%sinusoid that's identical to its carrier. I know that the carrier is of
%the form cos(2*pi*fc*t)), which means that fc should probably be 500,
%1500, 2500, and 3500. I believe that after THAT you apply the lowpass
%filter, which will cut out the other signals that aren't centered.

%% First Signal (500 Hz)

%Multiply the entire signal by the carrier
for k=1:352800
   x1(k) = x(k)*2*cos(2*pi*500*5*(k/fs));
end;

% Uncomment the following Line to See Spectrum of Shifted Signal
% spectrogram(x1,blackman(8000*0.02),0, [] , 8000, 'yaxis')

% Create a Low Pass Filter
wc = (2*pi*800)/fs;
h = fir1(200,wc,'low');
x1 = filter(h,1,x1);

% Listen to the clean audio
sound(x1, fs);

% Uncomment this line to see the end result
spectrogram(x1,blackman(8000*0.02),0, [] , 8000, 'yaxis')

%% Second Signal (1500 Hz)

%Multiply the entire signal by the carrier
for k=1:352800
   x1(k) = x(k)*2*cos(2*pi*1500*5*(k/fs));
end;

% Uncomment the following Line to See Spectrum of Shifted Signal
% spectrogram(x1,blackman(8000*0.02),0, [] , 8000, 'yaxis')

% Create a Low Pass Filter
wc = (2*pi*800)/fs;
h = fir1(200,wc,'low');
```

```
x1 = filter(h,1,x1);

% Listen to the clean audio
sound(x1, fs);

% Uncomment this line to see the end result
%spectrogram(x1,blackman(8000*0.02),0, [] , 8000, 'yaxis')

%% Third Signal (2500 Hz)

%Multiply the entire signal by the carrier
for k=1:352800
   x1(k) = x(k)*2*cos(2*pi*2500*5*(k/fs));
end;

% Uncomment the following Line to See Spectrum of Shifted Signal
% spectrogram(x1,blackman(8000*0.02),0, [] , 8000, 'yaxis')

% Create a Low Pass Filter
wc = (2*pi*800)/fs;
h = fir1(200,wc,'low');
x1 = filter(h,1,x1);

% Listen to the clean audio
sound(x1, fs);

% Uncomment this line to see the end result
%spectrogram(x1,blackman(8000*0.02),0, [] , 8000, 'yaxis')

%% Fourth Signal (3500 Hz)

%Multiply the entire signal by the carrier
for k=1:352800
   x1(k) = x(k)*2*cos(2*pi*3500*5*(k/fs));
end;

% Uncomment the following Line to See Spectrum of Shifted Signal
% spectrogram(x1,blackman(8000*0.02),0, [] , 8000, 'yaxis')

% Create a Low Pass Filter
wc = (2*pi*800)/fs;
h = fir1(200,wc,'low');
x1 = filter(h,1,x1);

% Listen to the clean audio
sound(x1, fs);

% Uncomment this line to see the end result
%spectrogram(x1,blackman(8000*0.02),0, [] , 8000, 'yaxis')
```

Section IV – Functions

DecodeDTMFblock.m:

```
function digit = decodeDTMFblock(s,fs)

% Normalize the signal
x1 = NormalizeSignal(s);

% Find the signal's Start and End Points
indexstart = FindSignalStart(x1);
indexend   = FindSignalEnd(x1);

% Create a "smaller" version of the original signal, with only the relevant
% part
subx1 = x1(indexstart:indexend);

% Let's get some power
y = fft(subx1);

% Because of the "mirroring", we need to omit frequency data above the
% halfway mark
cutoff = round(0.5*length(y),0);

test = abs(y(1:cutoff));

% Create a threshold value so that the function doesn't report back very
% small values
threshold = (max(test)/2);

% Each Signal has two relevant frequencies: A 'low' and a 'high'
% These values are found, and then rounded (because my method of cutting
% the signal introduces a little error)
low = round(find(test>threshold,1)*(fs/length(y)), -1);
high = round(find(test>threshold,1, 'last')*(fs/length(y)), -1);

% Fortunately, all of the frequencies are more than 10 apart, so there
% shouldn't be any overlap in using this method of rounding.

% Every possible combination of addition for anticipated signals results in
% a unique number. Easier to build a case structure for.

combined = low+high;

if (cutoff ~= 0)
    switch combined
        case 1910
            digit = '1';
        case 2040
            digit = '2';
        case 2180
            digit = '3';
        case 1980
```

```
            digit = '4';
        case 2110
            digit = '5';
        case 2250
            digit = '6';
        case 2060
            digit = '7';
        case 2190
            digit = '8';
        case 2330
            digit = '9';
        case 2150
            digit = '*';
        case 2280
            digit = '0';
        case 2420
            digit = '#';
        otherwise
            digit = ' ';
    end
else
    digit = ' ';
end
```

FindSignalStart.m:

```
function y = FindSignalStart(x)

threshold = (max(x)/2);
y = find(x>threshold,1);
```

FindSignalEnd.m:

```
function y = FindSignalEnd(x)

threshold = (max(x)/2);
y = find(x>threshold,1, 'last');
```