NORWICH UNIVERSITY
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

EE 487 DIGITAL SIGNAL PROCESSING

Spring 2014

Laboratory Report

Laboratory Experiment 8
Software Radio

Submitted by:
Daniel Alexander Rodriguez

Email:
drodrigu@stu.norwich.edu

Date:
16 April, 2014

Abstract

This laboratory exercise involved creating a software radio platform using MATLAB. The tasks for modulation were split between two different MATLAB scripts, **`transmitter.m`** and **`receiver.m`**. The **`transmitter.m`** script read in two .wav sound files, low-pass filtered them, AM modulated them, and then saved the data into a new .wav sound file. The **`receiver.m`** script took input the .wav file from the output of the script **`transmitter.m`** and demodulated it by first band-pass filtering the data twice for each signal individually, removed the carrier signal using a local oscillator, and then removed the DC offset. Plots of the signals are specified in the report to provide a visual understanding of the process taken for this software radio.

Introduction

The purpose of this laboratory exercise was to create two MATLAB scripts, one to transmit an AM signal composed of two individual input sound files, and the other to demodulate the transmitted AM signal into the original separated sound files. These MATLAB scripts, called **`transmitter.m`** and **`receiver.m`**, had to transmit the AM signal and demodulate it, respectively.

Apparatus and Procedure

The mathematics-modeling program MATLAB was used to model this laboratory exercise. This procedure will be narrated along with the corresponding MATLAB code.

The script **`transmitter.m`** had to read in two 48-kHz .wav files, one from "station 1" and the other from "station 2". This was accomplished using the **`wavread.m`** script, where the file data was read into the programs memory, along with the sampling rate and number of bits.

```
[station1,Fs,nbits] =  wavread('Soundtrack-Theme_12.wav');
[station2,Fs,nbits] =  wavread('Bloody_Tears.wav');
```

Next, the song lengths had to be equated for proper operation of the transmitter program. Note that the sound file with the shortest length was used as the limiting factor. In effect, the sound file with the longer length was reduced to the length of the sound file with the shorter length. The variable **`station1_fix`** is the result of adjusting the length of the sound file with greater length.

```
N2 = length(station2);
N1 = length(station1);
station1_fix = station1(1:N2);
```

Following this adjustment, both signals were low-pass filtered to a cutoff frequency of 4,000 Hz to shrink their frequency domain and consequently allow both signals to fit within the same 48-kHz frequency band. Note that `wn` $= 2\pi \times 4{,}000$ Hz. The order of the filter was carefully selected to avoid signal distortion.

```
[Hzn_LP, Hzd_LP] = butter(10, wn);

mt1 = filter(Hzn_LP,Hzd_LP,station1_fix);
mt2 = filter(Hzn_LP,Hzd_LP,station2);
```

A carrier frequency was added to both signals, as well as a DC offset. The amplitude was reduced to avoid clipping in the signal. Note that both signals were given two different frequencies. These were carefully selected to avoid overlapping signals. Since the signals were low-pass filtered to a cutoff frequency of 4,000 Hz, their bandwidth reached twice that cutoff point (e.g. 8,000 Hz) due to folding.

```
fc1 = 9000;
fc2 = 19000;

st1 = Ac*[1 + mt1].*cos(2*pi*fc1*t');
st2 = Ac*[1 + mt2].*cos(2*pi*fc2*t');
```

After filtering and modulating both signals, they were combined into one data file, and then stored to the computer's hard drive as a .wav sound file using the **`wavwrite.m`** script. Note that the **`transmitter.m`** can be found in **APPENDIX A** in its entirety.

```
st = st1 + st2;
wavwrite(st,Fs,nbits,'modulate.WAV')
```

The **`receiver.m`** script then reads in the .wav file `modulate.WAV` stored to the computer's hard drive by the **`transmitter.m`** script.

```
[st,Fs,nbits] =  wavread('modulate.wav');
```

The two signals contained in the data file were separated through band-pass filtering. Note that **`butter.m`** was used to generate a transfer function for the band-pass filters, and **`filter.m`** was used to filter the signals using the supplied transfer function. The order for **`butter.m`** was carefully selected to avoid signal distortion.

```
[Hzn_BP1, Hzd_BP1] = butter(2, [5000 13000]/(Fs/2),'bandpass');
demod1 = filter(Hzn_BP1,Hzd_BP1,st);

[Hzn_BP2, Hzd_BP2] = butter(2, [15000 23000]/(Fs/2),'bandpass');
demod2 = filter(Hzn_BP2,Hzd_BP2,st);
```

Next, a local oscillator signal was created to apply a frequency offset to the signal. The following code found the phase shift angle and then created the local oscillator signal with it to synchronize with both sound signals separately. Note that this code is specifically for the signal with the band centered around 9,000 Hz. The other signal uses the same code for its respective variables.

```
for i=2:Fs % about 1 sec of s(t)
   if (demod1(i-1) < demod1(i) & demod1(i) >= ...
           demod1(i+1) & demod1(i) > threshold)

         k = k + 1;
         peaks(k) = i;
   end;
end;
phi1  = 2*pi*fc1*t(peaks(1));
demod1_LO = cos(2*pi*fc1*t - phi1);
```

With the local oscillator signals produced, the sound files could be filtered by them. Both signals were simply multiplied by their respective oscillator signals to remove their carrier frequencies and to move them to the origin of their corresponding frequency spectra.

```
st1 = demod1.*demod1_LO';
st2 = demod2.*demod2_LO';
```

With both signals moved to the origin, they could then be low-pass filtered and restored. The MATLAB scripts **`butter.m`** and **`filter.m`** were again used for low-pass filtering.

```
[Hzn_LP, Hzd_LP] = butter(10, wn);

st1_LP = filter(Hzn_LP,Hzd_LP,st1);
st2_LP = filter(Hzn_LP,Hzd_LP,st2);
```

Although the signals were restored, the DC offset applied earlier was still present. This offset was removed to finally reproduce the signals in their original form.

```
mt1 = st1_LP - Ac/2;
mt2 = st2_LP - Ac/2;
```

Finally, the signals were written to the computer's hard drive as .wav sound files using the **`wavwrite.m`** script. Note that the **`receiver.m`** can be found in **APPENDIX B** in its entirety.

Analysis and Results

The two sound signals input into the **`transmitter.m`** program were composed into a single signal. Both signals were combined into the same spectrum so that no overlap or distortion would occur, and then were low-pass filtered to span a maximum of 8 kHz of frequency along the same spectrum

The two signals were modulated using the technique of AM modulation. See **Equation (1)** for the AM modulation equation.

$$V_{OUT}(t) \approx \frac{A_c}{2}[1 + m(t)]\cos(\theta_0) \tag{1}$$

This modulation technique introduces a DC offset and a sinusoid of a specified carrier frequency to modulate the data. The carrier signals were carefully chosen for both respective signals to ensure no overlap would occur between both signals' bands. The carrier frequencies were selected to be 10 kHz apart, allowing proper signal band clearance. See **Figure 1** for the resulting spectrum.

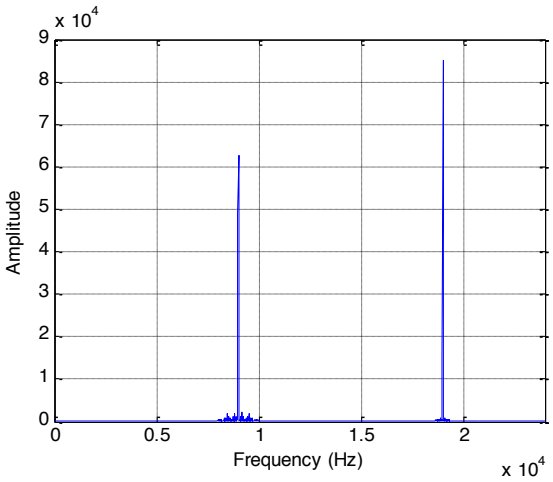

Figure 1. This plot contains the spectrum of the two sound signal combined into the same data stream. Notice that both carrier frequencies are about 10 kHz apart due to the carrier frequencies, giving room for the 8-kHz band of both signals to avoid overlapping.

Using **`transmitter.m`**, each sound signal was then separated using band-pass filtering both signals individually. See **Figure 2** for the plotted result of the filtered spectra data for both sound signals.

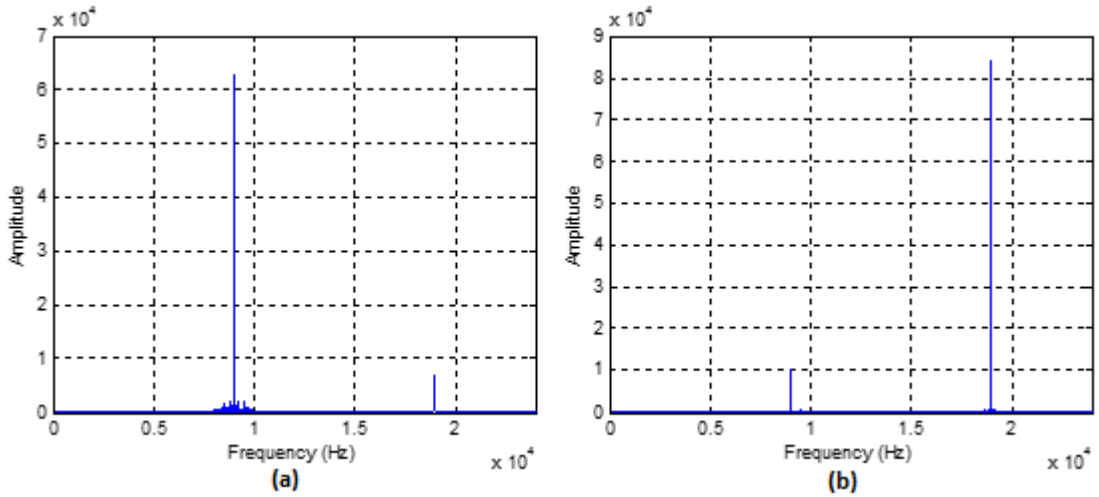

Figure 2. On the left, **Figure 2(a)** shows the result of band-pass filtering the signal at the 9-kHz carrier frequency, whereas on the right, **Figure 2(b)** shows the result of band-pass filtering the signal at the 19-kHz carrier frequency. Notice that compared to the original spectrum in **Figure 1**, each respective filter greatly depletes one of the carrier frequencies as well as the data around them.

Upon playing back both recovered sound signals in their current state, it would be discovered that the song would contain a powerful high-frequency noise, along with the data in the higher frequency range. This was more noticeable for the data centered at the 19-kHz carrier frequency peak. This issue was fixed by introducing a local oscillator signal that would offset the carrier signal so that the sound data would center about the origin. See **Equation (2)** for the equation of a local oscillator.

$$V_{LO}(t) = \cos\left(2\pi f_c t + \theta_0\right) \tag{2}$$

The oscillator shares the same carrier frequency of the corresponding signal, and finds the offset angle θ_0 to center the signal at the origin. MATLAB code was used in the procedure of this laboratory exercise to automatically find the theta value for both signals. After filtering the AM modulated signal with the local oscillator, the result should appear as it does in **Equation (3)**.

$$V_{OUT}(t) \approx \frac{A_c}{2}[1 + m(t)] \tag{3}$$

See **Figure 3** for spectra of both sound signals after being filtered by the local oscillator.

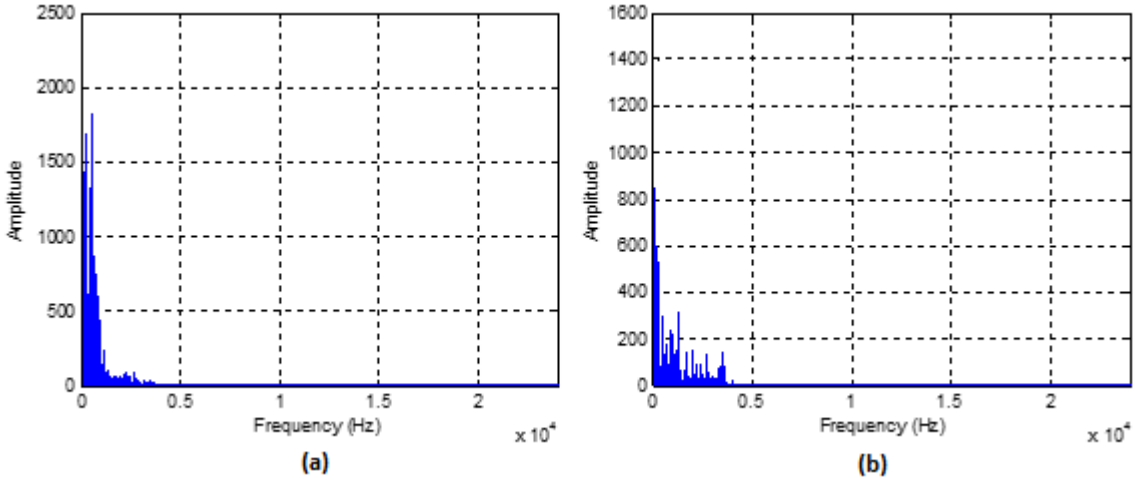

Figure 3. The spectrum on the left, **Figure 3(a)** contains the signal with the 9-kHz carrier frequency after it was filtered by the local oscillator, and the spectrum on the right, **Figure 3(b)**, contains the signal with the 19-kHz carrier frequency after it was filtered by the local oscillator. Notice the signals of both spectra are now centered about the origin.

By inspection, both signals appear to be restored completely; however, the DC offset still remains in both signals, which throws off the gain of the signals. When examining the signals in the time domain, the DC offset is easy to identify. See **Figure 4** for the time domain plot of the signals with the DC offset.

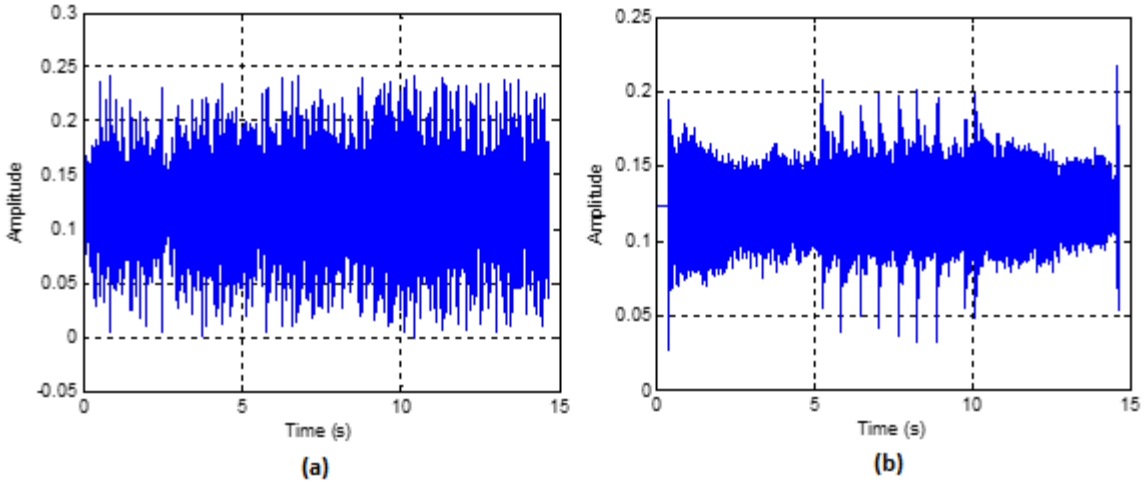

Figure 4. These are the time domain signals of the sound data with the DC offset still in the signal. **Figure 4(a)** contains the signal that previously contained the 9-kHz carrier frequency, while **Figure 4(b)** contains the signal that previously contained the 19-kHz carrier frequency.

The DC offset can be removed by simply subtracting the same offset from the entire signal. See **Equation (4)** for the equation to remove the DC offset from the signals.

$$V_{OUT}(t) \approx \frac{A_c}{2}[1 + m(t)] - \frac{Ac}{2} = \frac{A_c}{2}m(t) \qquad (4)$$

See **Figure 5** for the plotted result of removing the DC offset from both of the signals.

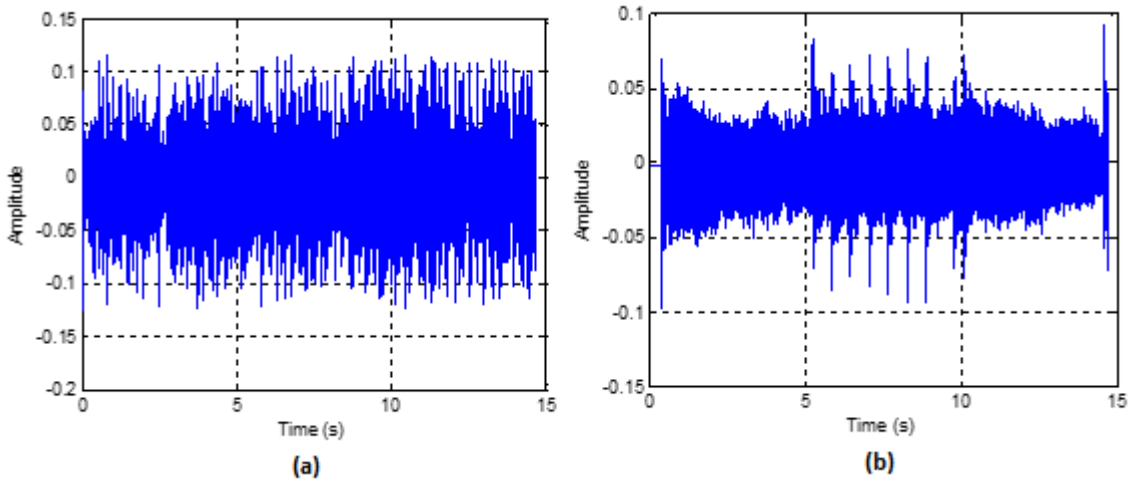

Figure 5. These are the time domain signals of the sound data with the DC offset removed. **Figure 5(a)** contains the signal that previously contained the 9-kHz carrier frequency, while **Figure 5(b)** contains the signal that previously contained the 19-kHz carrier frequency.

The result contains the fully restored sound signals that were input into **`transmitter.m`**. Upon playback, the sound signals played back exactly as they had before being modulated, aside from an obvious loss in quality from low-pass filtering both signals down to 4 kHz and lowering the gain.

Conclusion

Ultimately, this laboratory exercise has proven to be a success. The signals input to the AM modulating system were demodulated back into their original form. The frequency bands and carrier frequencies were used to fit both sound signals into the same spectrum to allow the transfer between the MATLAB scripts **`transmitter.m`** and **`receiver.m`**. The data were demodulated successfully by removing the carrier signal with a local oscillator to negate it, and the DC offset was removed. The gain of the signal was still decreased and the smaller sampling rate decreased the quality of the sound data. Regardless, this lab was intended to prove that the songs could be recreated, not to recreate the sound fidelity.

Appendices

APPENDIX A

This is the MATLAB code for the **`transmitter.m`** script, which was used to read in and prepare two sound files for modulation.

```
clc;
clear;
close all;

%%-- File Read --%%

[station1,Fs,nbits] =  wavread('Soundtrack-Theme_12.wav');
[station2,Fs,nbits] =  wavread('Bloody_Tears.wav');

%%-- File Vector Length Fix --%%

N2 = length(station2);
N1 = length(station1);
station1_fix = station1(1:N2);

%%-- Low-Pass Filter --%%

f0 = 4000;
df = Fs/N2;
Ts = 1/Fs;
wn = f0/(Fs/2);
t = Ts*[0:1:N2-1];
f = (Fs/N2)*[0:1:N2-1];

[Hzn_LP, Hzd_LP] = butter(10, wn);

mt1 = filter(Hzn_LP,Hzd_LP,station1_fix);
mt2 = filter(Hzn_LP,Hzd_LP,station2);

%%-- AM Modulation --%%

Ac = 0.25;
fc1 = 9000;
fc2 = 19000;

st1 = Ac*[1 + mt1].*cos(2*pi*fc1*t');
st2 = Ac*[1 + mt2].*cos(2*pi*fc2*t');

%%-- AM Signal Combination --%%

st = st1 + st2;
st_fft = fft(st);

figure(1)
plot(mt1)
grid on;
```

```
figure(2)
plot(st1)
grid on;

figure(3)
plot(f, abs(st_fft))
xlim([0 (Fs/2)])
grid on;

%%-- File Write --%%

wavwrite(st,Fs,nbits,'modulate.WAV')
```

APPENDIX B

This is the MATLAB code for the **`receiver.m`** script, which was used to take the output modulated file from **`transmitter.m`** and demodulate the two original sound files contained in it.

```
clc;
clear all;
close all;

%%-- File Read --%%

[st,Fs,nbits] =  wavread('modulate.wav');

N = length(st);
f = (Fs/N)*[0:1:N-1];
df = Fs/N;
Ts = 1/Fs;
t = Ts*[0:1:N-1];

%%-- Bandpass Demodulation --%%

[Hzn_BP1, Hzd_BP1] = butter(2, [5000 13000]/(Fs/2),'bandpass');
demod1 = filter(Hzn_BP1,Hzd_BP1,st);

[Hzn_BP2, Hzd_BP2] = butter(2, [15000 23000]/(Fs/2),'bandpass');
demod2 = filter(Hzn_BP2,Hzd_BP2,st);

demod1_fft = abs(fft(demod1));
demod2_fft = abs(fft(demod2));

figure(1)
plot(f, demod1_fft)
xlim([0 Fs/2])
grid on

figure(2)
plot(f, demod2_fft)
xlim([0 Fs/2])
grid on

%%-- Local Oscillator --%%

fc1 = 9000;
fc2 = 19000;

% Synchronization of the demod1's Local Oscillator
k = 0;
threshold = 0.5*max(st);
for i=2:Fs % about 1 sec of s(t)
   if (demod1(i-1) < demod1(i) & demod1(i) >= ...
           demod1(i+1) & demod1(i) > threshold)
         k = k + 1;
         peaks(k) = i;
```

```
   end;
end;
phi1  = 2*pi*fc1*t(peaks(1));
demod1_LO = cos(2*pi*fc1*t - phi1);
% ------------------------------------------------------------

% Synchronization of the demod2's Local Oscillator
k = 0;
threshold = 0.5*max(st);
for i=2:Fs % about 1 sec of s(t)
   if (demod2(i-1) < demod2(i) & demod2(i) >= ...
            demod2(i+1) & demod2(i) > threshold)
         k = k + 1;
         peaks(k) = i;
   end;
end;
phi2  = 2*pi*fc2*t(peaks(1));
demod2_LO = cos(2*pi*fc2*t - phi2);
% ------------------------------------------------------------

st1 = demod1.*demod1_LO';
st2 = demod2.*demod2_LO';

%%-- Low-Pass Filter --%%

f0 = 4000;
wn = f0/(Fs/2);

[Hzn_LP, Hzd_LP] = butter(10, wn);

st1_LP = filter(Hzn_LP,Hzd_LP,st1);
st2_LP = filter(Hzn_LP,Hzd_LP,st2);

figure(3)
plot(t, st1_LP)
grid on

figure(4)
plot(t, st2_LP)
grid on

%%-- DC Offset Removal --%%

Ac = 0.25;

mt1 = st1_LP - Ac/2;
mt2 = st2_LP - Ac/2;

figure(5)
plot(t, mt1)
grid on

figure(6)
plot(t, mt2)
```

```
grid on

figure(7)
plot(f, abs(fft(mt1)))
xlim([0 Fs/2])
grid on

figure(8)
plot(f, abs(fft(mt2)))
xlim([0 Fs/2])
grid on

%%-- File Write --%%

wavwrite(mt1,Fs,nbits,'mt1.WAV')
wavwrite(mt2,Fs,nbits,'mt2.WAV')
```