
Principles of Communication Systems Lab

Project Report

Date of Submission : November 30, 2019

Mili Goyal (IMT2017513)

Vaishnavi Dhulipalla (IMT2017514)

Purpose

To design FM Modulator/Demodulator that resists jamming.

Introduction

Jamming :

The idea of jamming is to prevent a receiver from receiving an intended signal, or to alter a signal or to inject a false signal.

Preventing a receiver from receiving an intended signal is often just a matter of power.

A very loud jamming signal can “desensitize” the receiver to the point that it can’t hear the intended signal, even if they are on different (but nearby) frequencies. This is particularly noticeable on FM radio, where the receiver will lock on to the loudest signal and pretty much ignore a weaker signal.

In this project : The major jamming techniques that we had to resist were :

- 1) Capture Effect : Adding another FM signal to the modulated waveform.
- 2) Noise : White Gaussian noise as per SNR given by the instructor

Anti-Jamming Techniques :

The major anti jamming techniques that we researched to deal with the capture effect was :

Spread Spectrum : This is a technique in which a telecommunication signal is transmitted on a bandwidth considerably larger than the frequency content of the original information.

Spread-spectrum telecommunications is a signal structuring technique that employs direct-sequence, frequency hopping, or a hybrid of these, which can be used for multiple access and/or multiple functions. This technique decreases the potential interference to other receivers while achieving privacy. Spread spectrum generally makes use of a sequential noise-like signal structure to spread the normally narrowband information signal over a relatively wideband of frequencies. The receiver correlates the received signals to retrieve the original information signal.

The two major techniques in spread-spectrum are :

- 1) **Frequency Hopping Spread Spectrum (FHSS)** : Frequency-hopping spread spectrum (FHSS) is a method of transmitting radio signals by rapidly switching a carrier among many frequency channels, using a pseudorandom sequence known to both transmitter and receiver.

This is the technique that we have used in this project.

- 2) **Direct Sequence Spread Spectrum (DSSS)** : Direct Sequence Spread Spectrum (DSSS) is a spread spectrum technique whereby the original data signal is multiplied with a pseudo random noise spreading code. This spreading code has a higher chip rate (this the bitrate of the code), which results in a wideband time continuous scrambled signal.

For the reduction of the noise, the technique that we used was '**wdenoise**' in matlab.

This is basically Wavelet threshold denoising technique.

The basic idea behind wavelet denoising, or wavelet thresholding, is that the wavelet transform leads to a sparse representation for many real-world signals

What this means is that the wavelet transform concentrates signal in a few large-magnitude wavelet coefficients. Wavelet coefficients which are small in value are typically noise and you can "shrink" those coefficients or remove them without affecting the signal or image quality. After you threshold the coefficients, you reconstruct the data using the inverse wavelet transform.

wdenoise(X) : denoises the data in X using an empirical Bayesian method with a Cauchy prior. By default, the sym4 wavelet is used with a posterior median threshold rule. Denoising is down to the minimum of $\text{floor}(\log_2 N)$ and $\text{wmaxlev}(N, 'sym4')$ where N is the number of samples in the data. (For more information, see `wmaxlev`.) X is a real-valued vector, matrix, or timetable.

Approach

So the basic idea is to build a transmitter, channel and receiver in matlab such that it can resist jamming.

For modulation : We divide the signal into various segments with overlapping and then create a carrier array with randomized frequencies and modulate each segment with a different carrier frequency from this array.

We ensure that there is sufficient gap between each carrier frequency to avoid interference. So, we get various channels instead of a single channel. And our signal practically hops between these.

For demodulation : we use the same carrier array to demodulate the modulated signal by dividing into same segments considering overlapping.

So, this way, we increase the bandwidth and if a channel is jammed, whole signal will not be affected, only a smaller segment of it will be affected, thereby enabling us to receive the near original signal at the receiver.

Difficulties Faced

- 1) **Upsampling and Downsampling** : One problem that we had to deal with was that our f_s for sound signal was around 41000Hz and for modulating, the carrier given to us were in Mhz. So to deal with this we did upsampling and downsampling of the signal so that we get an increased f_s and thus were successfully able to modulate the signal with the given frequencies.
- 2) **Spikes** : Due to repeated changing of the carrier frequencies very frequently (after every 1 seconds as each segment was nearly 1 second long) , there were spikes in the final received signal, which added some 'tuck' sounds in the final signal. So to deal with this problem, we tried various filters like median filters but that did not work that effectively, so we tried removing a very few samples at the starting and the ending of each segment.

- 3) **Unusual behaviour in the demodulated plot** : There was some unusual behaviour in the demodulated plot which showed addition of some amplitudes in the final signal with no clue of where it was coming from. This was a very unexpected result and we were unable to figure out the main cause of it. We could guess that it probably was a result of demodulation happening in 'fmdemod' function but the exact cause is known.

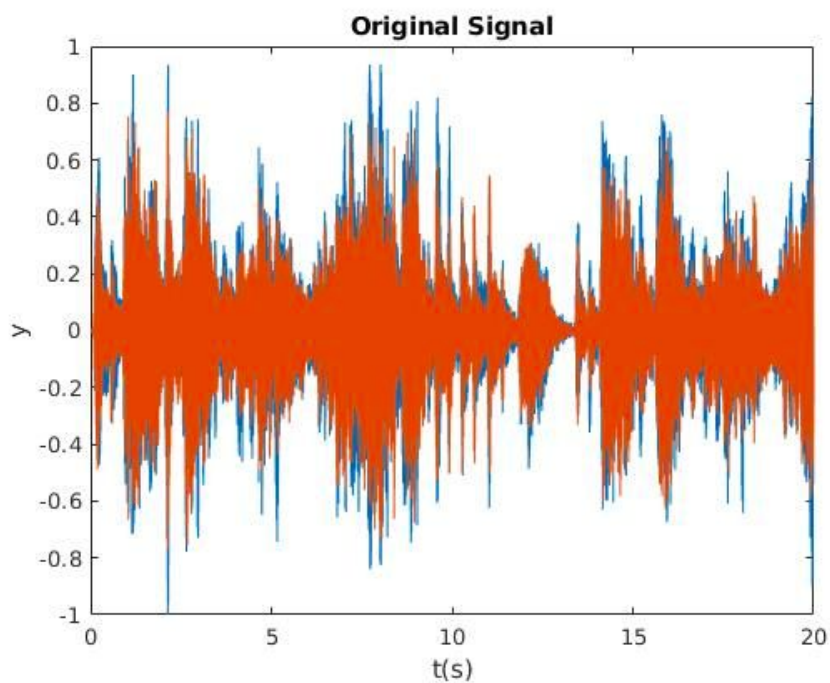
The plot is shown in the observations section below.

Observations

The sound file for the below plots is the 'Music.wav', that was given to us in the lab.

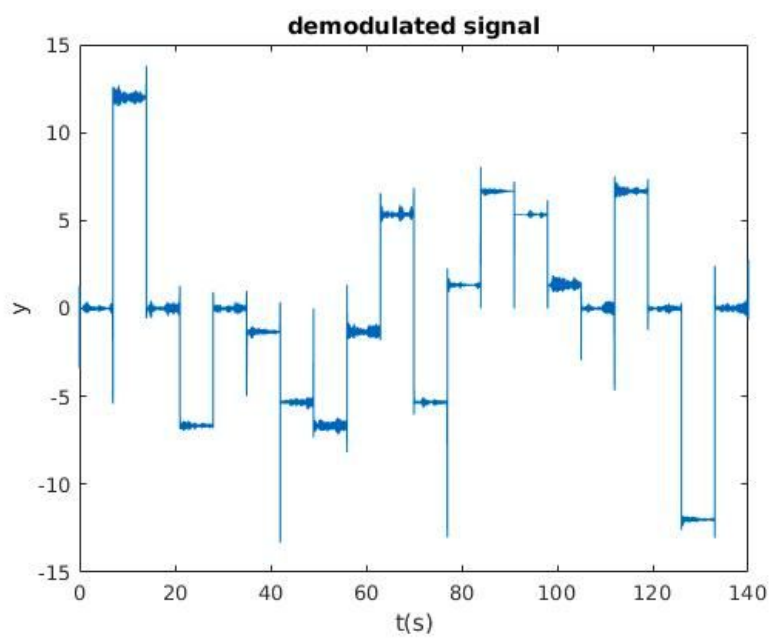
Plot 1

Original sound signal



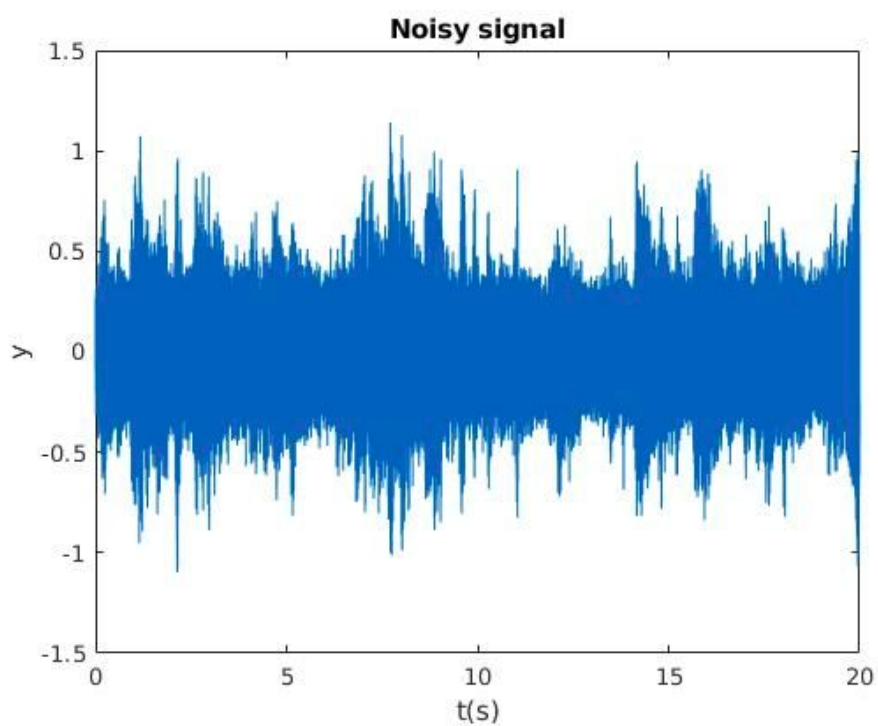
Plot 2

Plot after demodulation with capture effect



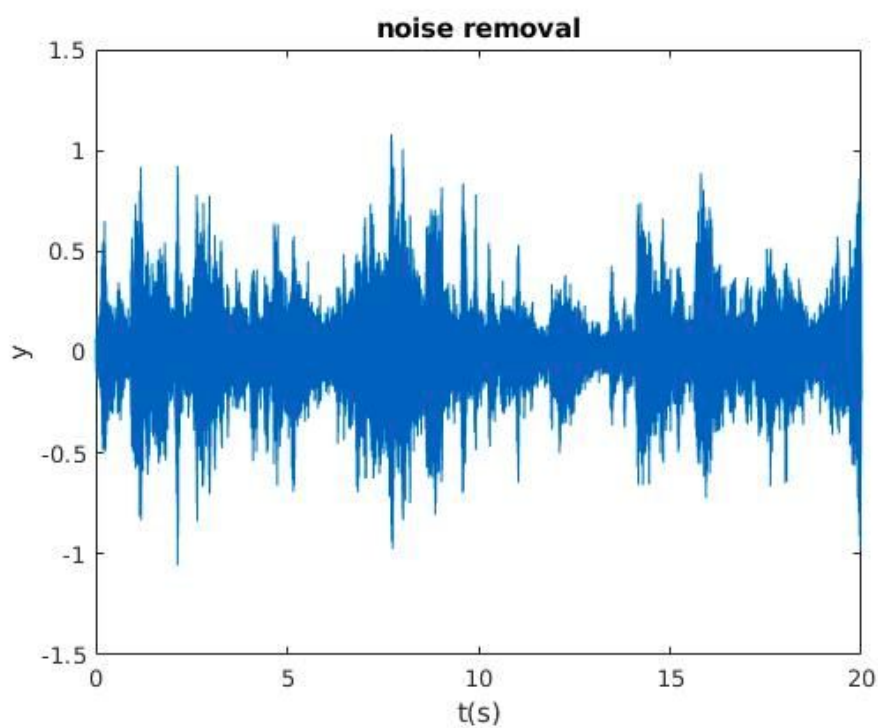
Plot 3

Plot after adding AWGN noise of SNR 20 to the signal



Plot 4

Plot of the signal received after removal of the noise



Appendix

Matlab Code

```
[y, fs] = audioread("Music.wav");
dt = 1/fs;
t = 0:dt:(length(y)*dt)-dt;

grid on;
plot(t, y)
title('Original Signal');
xlabel('t(s)');
ylabel('y');
figure(1)


y_noise = awgn(y(:, 1), 20);    % addition of white noise
y_fft = fft(y(:, 1));
freq = 1./t;

% upsampling
y = upsample(y, 7);
fs2 = 7*fs;
```

% this gives us the plot for the original sound signal

```
figure(2)
grid on;
plot(t, y_noise);
title('Noisy signal');
xlabel('t(s)');
ylabel('y');
```

```
% reading another audio file
[y5, fs5] = audioread("no-thats-not-gonna-do-it.wav");
y5 = y5(:, 1);
y5 = upsample(y5, 7);
y_mod = fmmmod(y5, 1e4, fs2, 7500);
y_mod = y_mod(:, 1);
y_mod = [y_mod; y_mod; y_mod; y_mod; y_mod; y_mod; y_mod; y_mod; y_mod];
y_mod = [y_mod; y_mod];
y_mod = [y_mod; y_mod];
% Method to remove white noise from the signal
% Wp = 1000/(fs/2);
% Ws = 1010/(fs/2);
% Rp = 20; % Passband Ripple (dB)
% Rs = 30; % Stopband Ripple (dB)
% [n, Ws] = cheb2ord(Wp, Ws, Rp, Rs); % Filter Order
% [z, p, k] = cheby2(n, Rs, Ws, 'low'); % Filter Design
% [soslp, glp] = zp2sos(z, p, k); % Convert To Second-Order-Section For
    Stability
```





```
% filtered_sound = filtfilt(soslp, glp, y_noise);
% soundsc(filtered_sound, fs)

% Use of wavelet threshold for noise removal
y_filtered = wdenoise(y_noise, 4);      % posterior median threshold rule.

figure(3)
grid on;
plot(t,y_filtered);
title('noise removal');
xlabel('t(s)');
ylabel('y');

N = 20;    % This is the number of segments of the signal
carrier_array = [];
f1 = 1e4;
f2 = 2e4;
f3 = 3e4;
f4 = 4e4;
f5 = 5e4;
f6 = 6e4;
f7 = 7e4;
f8 = 8e4;
f9 = 9e4;
f10 = 10e4;
f11 = 12e4;
f12 = 13e4;
for n = 1:N
    c = randi(12, 1);
```



```
switch(c)
case(1)
    carrier_array = [carrier_array f4];
case(2)
    carrier_array = [carrier_array f4];
case(3)
    carrier_array = [carrier_array f4];
case(4)
    carrier_array = [carrier_array f2];
case(5)
    carrier_array = [carrier_array f2];
case(6)
    carrier_array = [carrier_array f6];
case(7)
    carrier_array = [carrier_array f7];
case(8)
    carrier_array = [carrier_array f8];
case(9)
    carrier_array = [carrier_array f9];
case(10)
    carrier_array = [carrier_array f10];
case(11)
    carrier_array = [carrier_array f11];
case(12)
    carrier_array = [carrier_array f12];
end
end

% dividing the sound signal into N segments:
y2 = length(y)/N;
```

```
y2 = ceil(y2);
t1 = 1;
t2 = ceil(y2);
count = 1;
modulated_signal = [];
y = y(:,1);
% y = y+y_mod;
y_t = transpose(y);
% fs2 = 2.2*f12;

% this is to create the overlapping window
z = 50;

% modulation
while t1 < length(y)
    if count == 1
        y3 = y(t1:min(t2, length(y)));
    else
        y3 = y(t1:min(t2, length(y)));
    end
    modu = fmmod(y3,carrier_array(count),fs2,7500);
    modulated_signal = vertcat(modu, modulated_signal);
    t1 = t1+y2;
    t2 = t2+y2;
    count = count+1;
end

modulated_signal = modulated_signal;% + y_mod(1:4410000);
fft_modulated_capture = fft(modulated_signal);
```

```

% demodulation
td1 = 1;
td2 = ceil(y2);
countd = 1;
demodulated_signal = [];
while td1 < ceil(length(modulated_signal))

    if countd == 1
        y4 = modulated_signal(td1:min(td2, ceil(length(modulated_signal))));
    else
        y4 = modulated_signal(td1:min(td2, ceil(length(modulated_signal))));
    end

    demodu = fmdemod(y4,carrier_array(countd),fs2,7500);
    demodulated_signal = vertcat(demodu, demodulated_signal);
    td1 = td1+y2;
    td2 = td2+y2;
    countd = countd+1;
end
modulated_fft = fft(modulated_signal);
demodulated_fft = fft(demodulated_signal);

% medfiltLoopVoltage = medfilt1(demodulated_signal,10);

% This is used to remove the spikes from the demodulated_signal

% y_demod = fmdemod( demodulated_signal, 100000, 2.2*100000, 7500);
% soundsc(medfiltLoopVoltage, fs);
% rm_outliers = rmoutliers(demodulated_signal);

```

% hearing the received sound signal

figure(4)

t2 = 0:dt:(length(demodulated_signal)*dt)-dt;

grid on;

plot(t2, demodulated_signal);

title('demodulated signal');

xlabel('t(s)');

ylabel('y');

demodulated_signal = downsample(demodulated_signal, 5);

soundsc(demodulated_signal, fs);

figure(5)

plot(abs(modulated_fft));

figure(6)

demodulated_fft = fft(demodulated_signal);

plot(abs(demodulated_fft));

figure(7)

plot(abs(fft_modulated_capture));

figure(8)

plot(modulated_signal);

