ESA:Signals - Acoustic Modem Final

Nabih Estefan
Isabella Abilheira

May 2020

Contents

1 Introduction

In this project we were able to implement a receiver for an acoustic modem. We created an acoustic modem that sends a digital signal using sound waves that use amplitude modulation to transmit them. After some reconstruction, filtering, and interpretation of the signal, we are able to decode any message contained in an acoustic signal.

2 Block Diagram

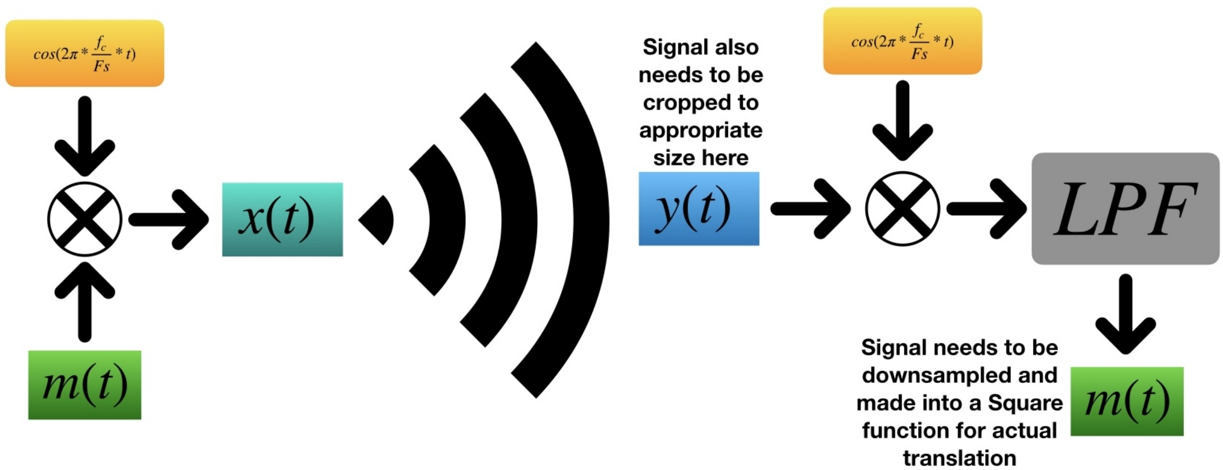

Figure 1: Block Diagram of Message Path

3 Step-by-Step with graphs

To begin, a continuous-time waveform $m(t)$ is used to represent our encoded message as a bit sequence shown in Figure 2 below.

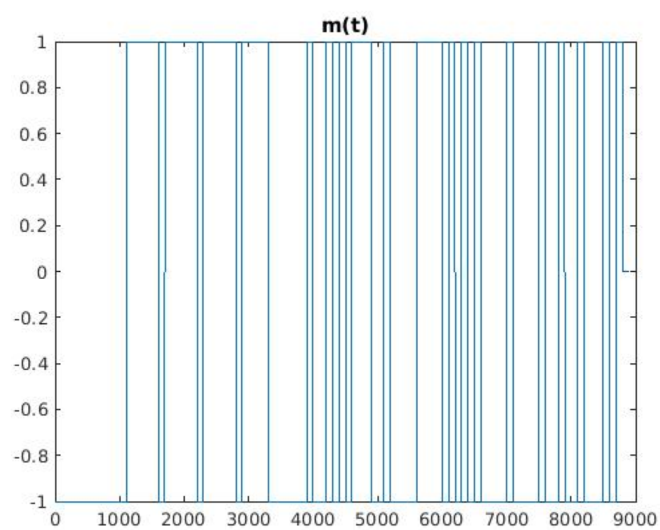

Figure 2: Original $m(t)$

Since digital information can be transmitted by using a sequence of pulses; the positive pulses represent ones in the bit sequence and negative pulses represent zeros. This message uses a pulse width, also know as the Symbol Period, of 100 time units. Since this message, $m(t)$ cannot be transmitted through the air we translate it to higher frequencies by multiplying it with a high frequency cosine. This then gives us our transmitted signal $x(t)$ represented by the equation

$$x(t) = m(t)cos(2\pi f_c t) \tag{1}$$

where f_c is the carrier frequency.

In Figure 3 below you can see plots for this signal in the time and frequency domains.

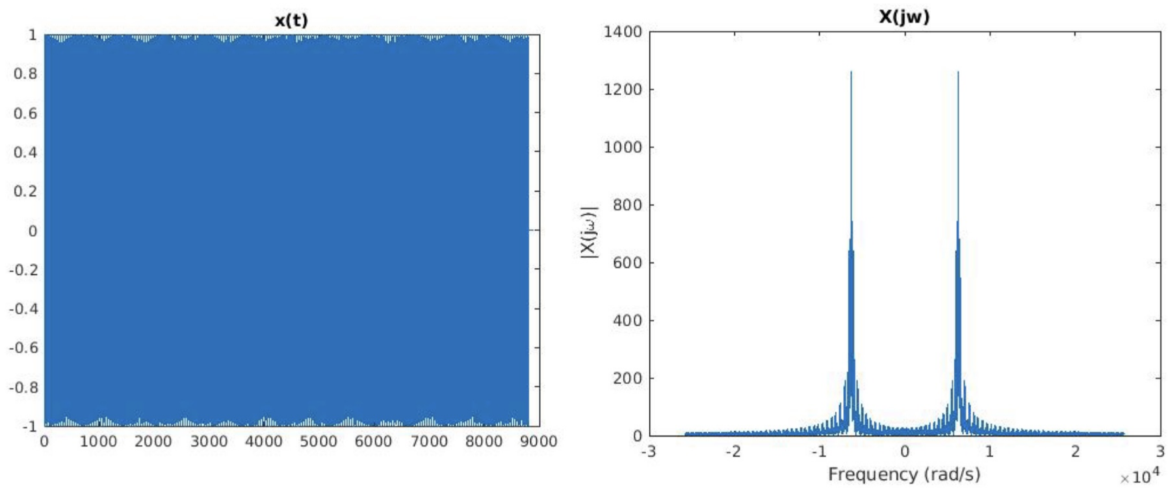

Figure 3: Transmitted Signal $x(t)$ & Frequency Response $X(jw)$

Once the signal has been sent to the receiver, it listens and saves the audio waves it records to $y(t)$. This means that our message is nested inside of $y(t)$ as you can see in Figure 4 below.

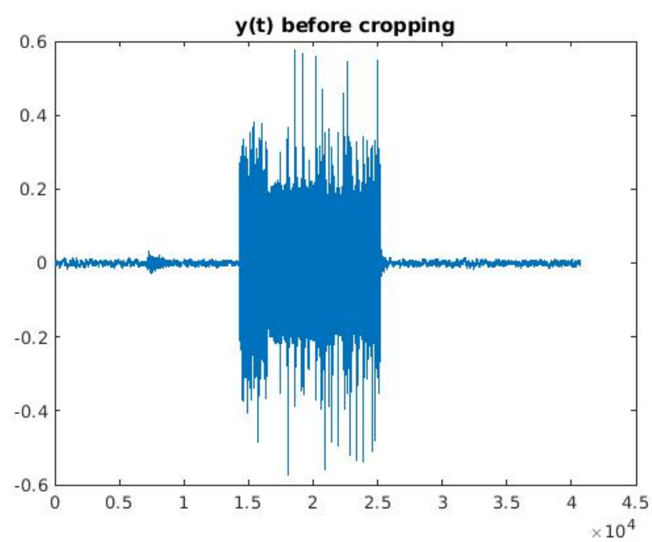

Figure 4: Received Signal $y(t)$ before cropping

Knowing this, the first step we take in synthesizing the received signal is to use our sync and message length data and crop the signal so that it is only the length of the message transmitted. The plots for this cropped $y(t)$ in the time and frequency domains can be seen below in Figure 5.

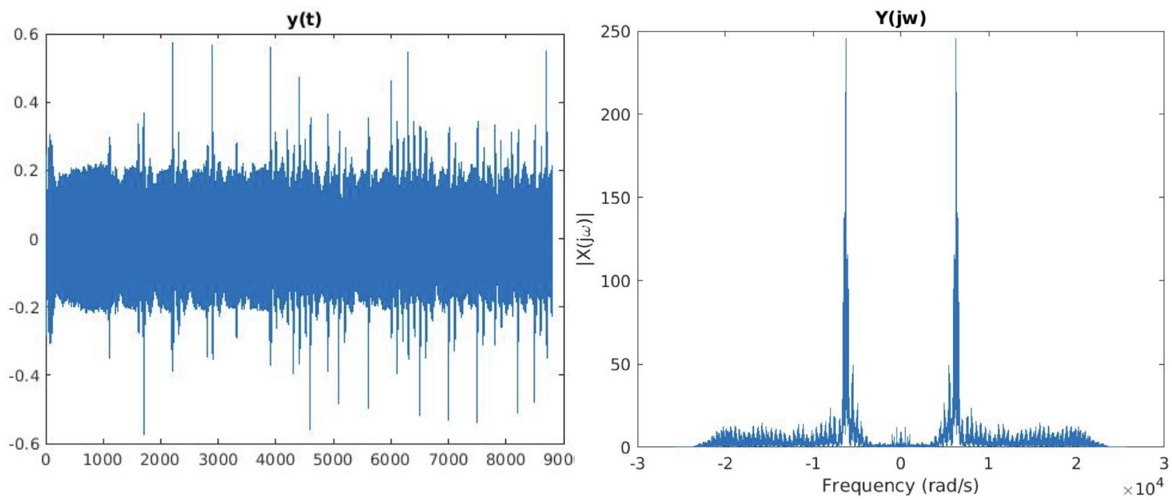

Figure 5: Received Signal $y(t)$ & Frequency Response $Y(jw)$ after cropping

After having cropped our recording to the message signal, we multiply this signal once again by the cosine wave we mentioned above. This leads to the time and frequency domain plots that can be seen in Figure 6.

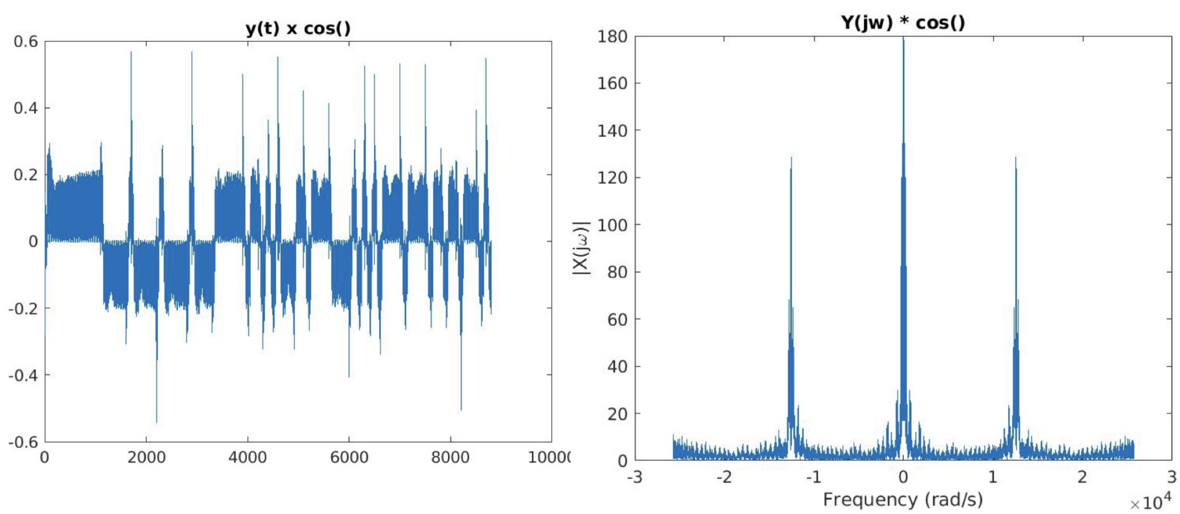

Figure 6: $y(t)cos(2\pi f_c t)$ & $Y(jw) * cos(2\pi f_c t)$

After getting this new $y(t)$ we need to implement a Low-Pass Filter,

$$h(t) = \frac{W}{\pi} sinc(\frac{W}{\pi} t) \tag{2}$$

to crop the signal to only the frequencies we want. Below, you can see the plots for $h(t)$ and its transform pair $H(jw)$ in Figure 7.

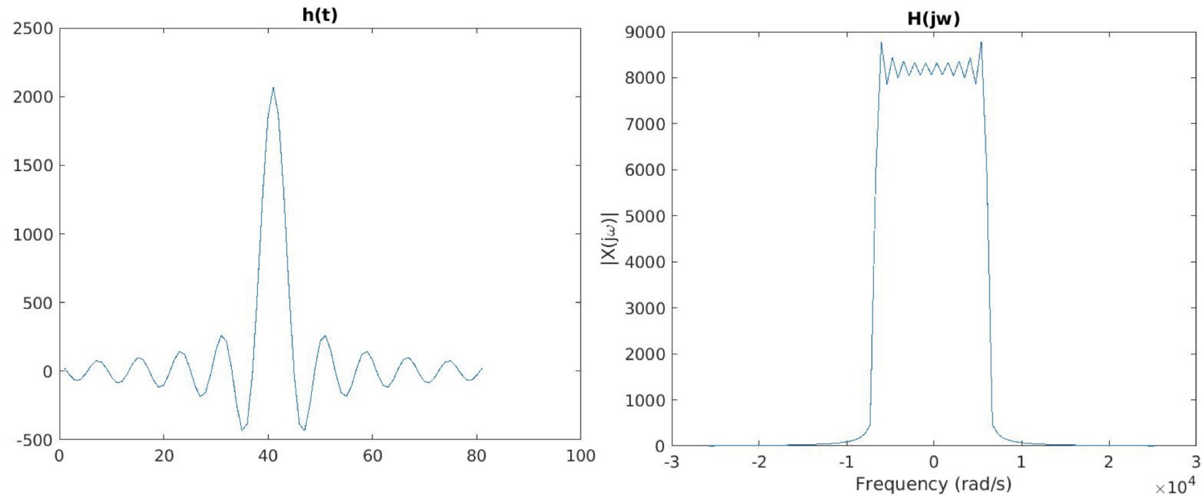

Figure 7: Low-Pass Filter $h(t)$ & Frequency Response $H(jw)$

Since we now have a Low-Pass Filter $h(t)$ and a signal $y(t)$, we can convolve them, in the time domain, to reconstruct our signal $m(t)$. When we do this operation, we get the signal seen in Figure 8 below. As you can see, its not a perfectly square signal so it requires some cleaning up.

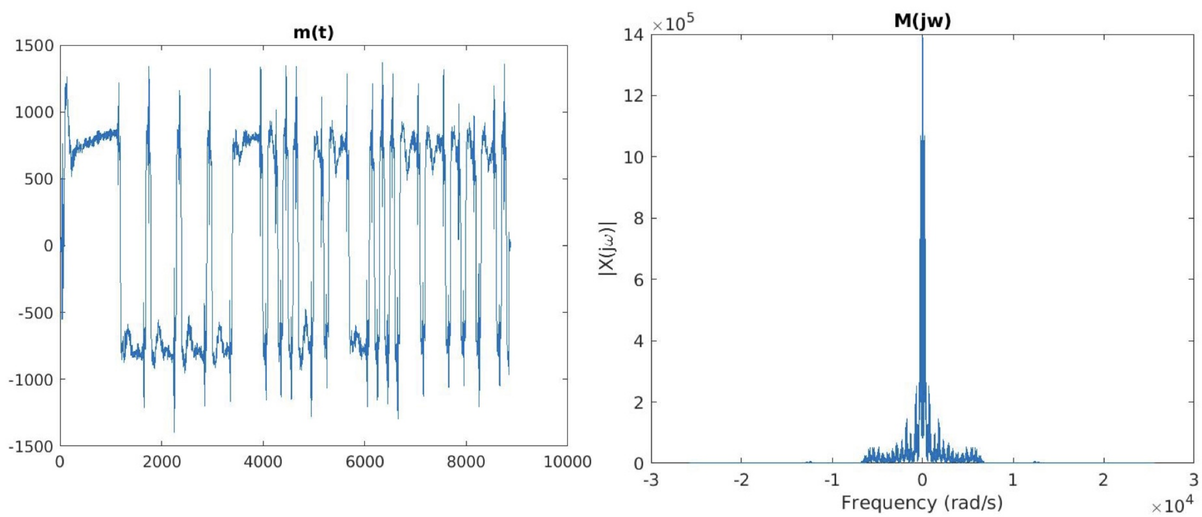

Figure 8: Received Signal $m(t)$ & Frequency Response $M(jw)$

Finally, we take the signal $m(t)$ above and clean it up by downsampling and making it a square signal. Once we have done this we get the wave in Figure 9 which you can see below. You will also see that its the same signal as the one in Figure 2, now with a Symbol Period of 1 time unit, meaning we recovered the message completely.

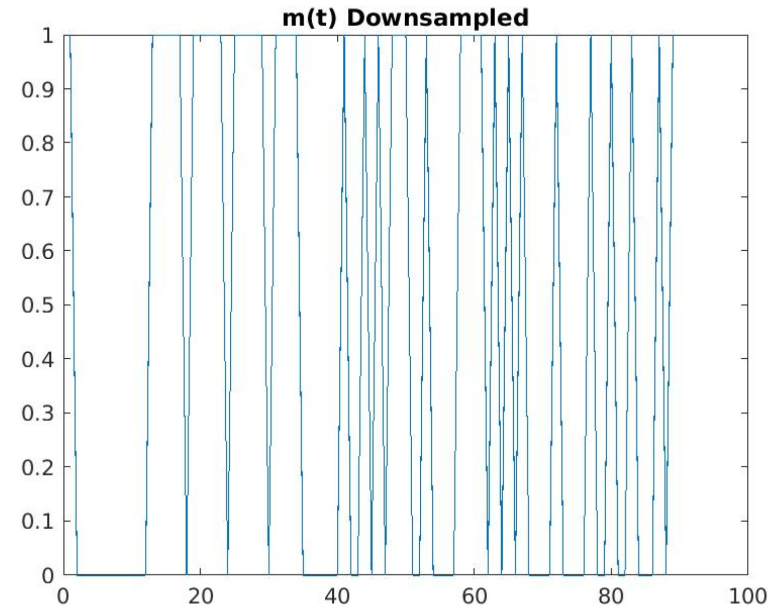

Figure 9: Cleaned up message $m(t)$

4 Code

Reciever:

```
clear all;close all;clc;
load data/mcdonalds.mat

recorder = audiorecorder(Fs, 16, 1);
recorder.StartFcn = 'disp(''Start speaking.'')';
recorder.StopFcn = 'disp(''End of recording.'')';

record(recorder, 10);
pause(10)
y_r = getaudiodata(recorder);

% Uncomment to read directly from wav file
% [y_r, Fs] = audioread('data/mcdonalds.wav');

f_c = 1000;
figure()
plot(y_r)
title("y(t) before cropping")

start_idx = find_start_of_signal(y_r,x_sync);
y_x = y_r(start_idx+length(x_sync):end);
y_t = y_x(1:msg_length*8*100);
figure()
plot(y_t)
title("y(t)")
figure()
plot_ft_rad(y_t, Fs);
title("Y(jw)")
```

```
%Multiply Singal with Cosine function
c = cos(2*pi*f_c/Fs*[0:length(y_t)-1]');
y_d = y_t .* c;
figure()
plot(y_d)
title("y(t) x cos()")
figure()
plot_ft_rad(y_d, Fs);
title("Y(jw) * cos()")

%Create Low-Pass Filter
W = 6500;
t = [-40:1:40]*(1/Fs);
h = W/pi*sinc(W/pi.*t);
figure()
plot(h)
title("h(t)")
figure()
plot_ft_rad(h, Fs);
title("H(jw)")

%Convolve with LPF
x_t = conv(y_d,h);
figure()
plot_ft_rad(x_t, Fs);
title("M(jw)")
figure()
plot(x_t)
title("m(t)")

%Normalize Singal to Interpret it
x_d = downsample(((square(x_t/400) + 1)./2),100);

% If signal needs to be flipped uncomment
x_d = mod(x_d+1, 2);

figure()
plot(x_d)
title("m(t) Downsampled")
Message = BitsToString(x_d(2:end))
```

Transmitter:

```
Fs = 8192;
f_c = 1000;
bits_to_send = StringToBits('   Heres    how I try to look at it, and this is
    just me: this guy being the president,  its  like theres a horse loose
     in a hospital,    Mulaney says.   I   think eventually everythings
    going to be okay, but I have no idea whats going to happen next. And
    neither do any of you, and neither do your parents, because theres a
    horse loose in the hospital!');
msg_length = length(bits_to_send)/8;
SymbolPeriod = 100;

```

```
% convert the vector of 1s and 0s to 1s and -1s
m = 2*bits_to_send -1;
% create a waveform that has a positive box to represent a 1
% and a negative box to represent a zero
m_us = upsample(m, SymbolPeriod);
m_boxy = conv(m_us, ones(SymbolPeriod, 1));
% figure()
% plot(m_boxy); % visualize the boxy signal
% title('m(t)')

% create a cosine with analog frequency f_c
c = cos(2*pi*f_c/Fs*[0:length(m_boxy)-1]');
% create the transmitted signal
x_tx = m_boxy.*c;
% figure()
% plot(x_tx)  % visualize the transmitted signal
% title('x(t)')
% figure()
% plot_ft_rad(x_tx, Fs)
% title('X(jw)')

% create  noise-like signal
% to synchronize the transmission
% this same noise sequence will be used at
% the receiver to line up the received signal
% This approach is standard practice is real communications
% systems.
randn('seed', 1234);
x_sync = randn(Fs/4,1);
x_sync = x_sync/max(abs(x_sync))*0.5;
% stick it at the beginning of the transmission
x_tx = [x_sync;x_tx];
save data/horse.mat x_sync Fs msg_length
% write the data to a file
audiowrite('data/horse.wav', x_tx, Fs);
```