Acoustic Modem Project Report

Kenta Burpee

May 2023

1 Introduction

The goal of this project was to implement an acoustic modem receiver using MATLAB. Acoustic modems are devices that transmit or receive sound waves and are often used for underwater communications. The function of the acoustic modem receiver is to decode a high-frequency signal it receives from the transmitter and extract bits of information from it. In this case, the extracted bits are later converted to a string that contains a message.

2 Procedure

2.1 Block Diagram

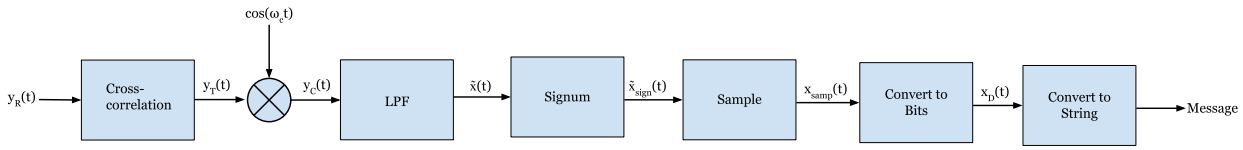

Figure 1: Block diagram for the acoustic modem receiver.

2.2 Steps

The first step in decoding the received signal is to find the start of the encoded message in the transmission. This is done by using cross-correlation with a known noise-like signal that is put before the message in the transmission. $y_R(t)$, the received signal gets converted to $y_T(t)$, which starts at the start of the message.

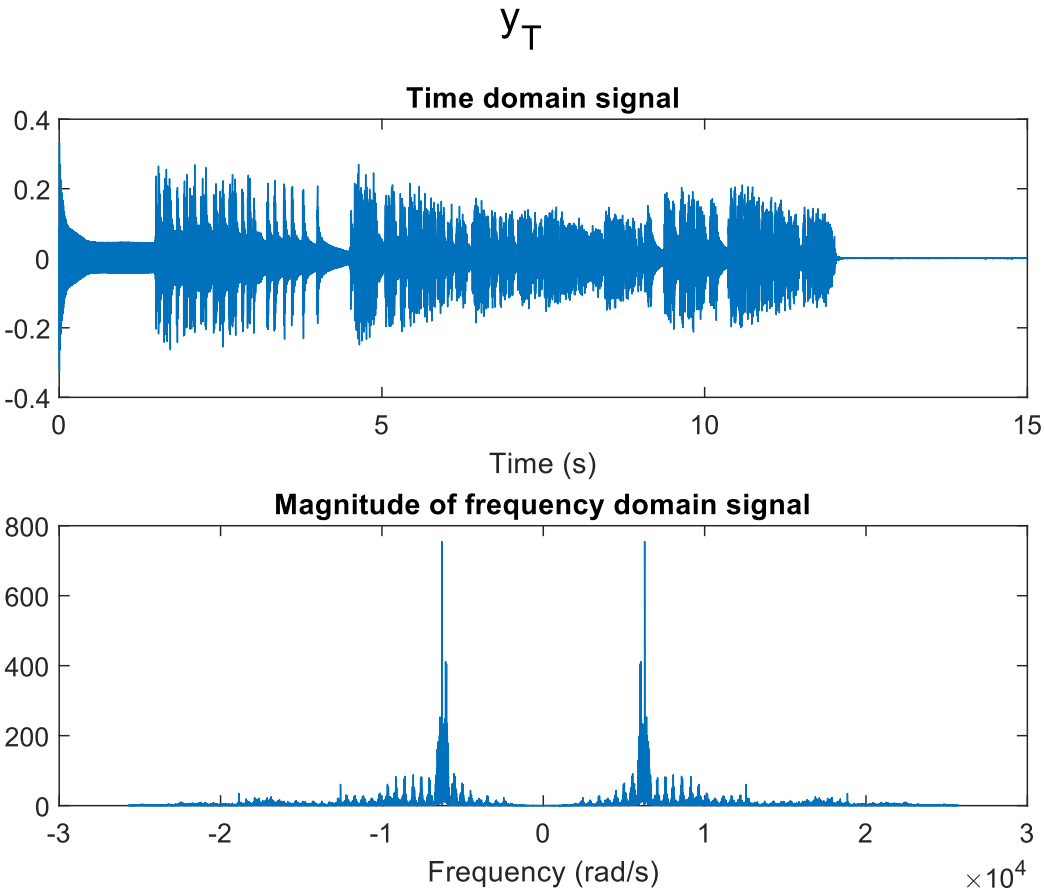

Figure 2: Time and frequency domain representations of y_T. This transmission contains a long message.

Next, in order to restore the original transmission signal before it was multiplied by a high-frequency cosine wave in the time domain, $y_T(t)$ must be once again multiplied by the same cosine wave, which is $cos(2 * \pi * f_c/F_s)$, where the cutoff frequency f_c is 1000Hz and the sampling frequency F_s is 8192Hz.

$$y_C(t) = y_T(t) * cos(2 * \pi * f_c/F_s) \tag{1}$$

The result of this multiplication in the time domain is $y_C(t)$, whose frequency response contains the original signal in the center and some high frequency noise outside the frequency range of the original signal.

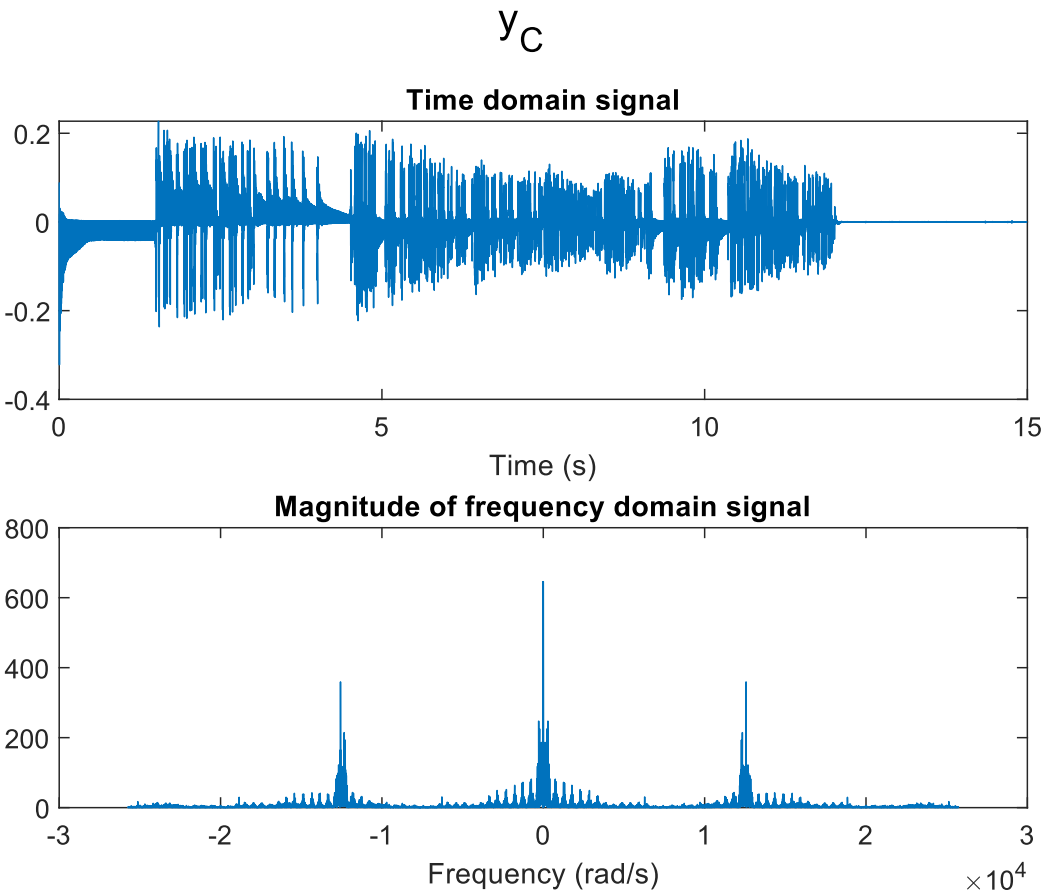

Figure 3: Time and frequency domain representations of y_C. The frequency response contains the original signal in the center.

To extract the original signal from $y_C(t)$, a low-pass filter with cutoff frequency f_c (1000Hz) is applied to $y_C(t)$. The result of this operation is $\tilde{x}(t)$. Small-amplitude noise is then filtered out in the time domain.

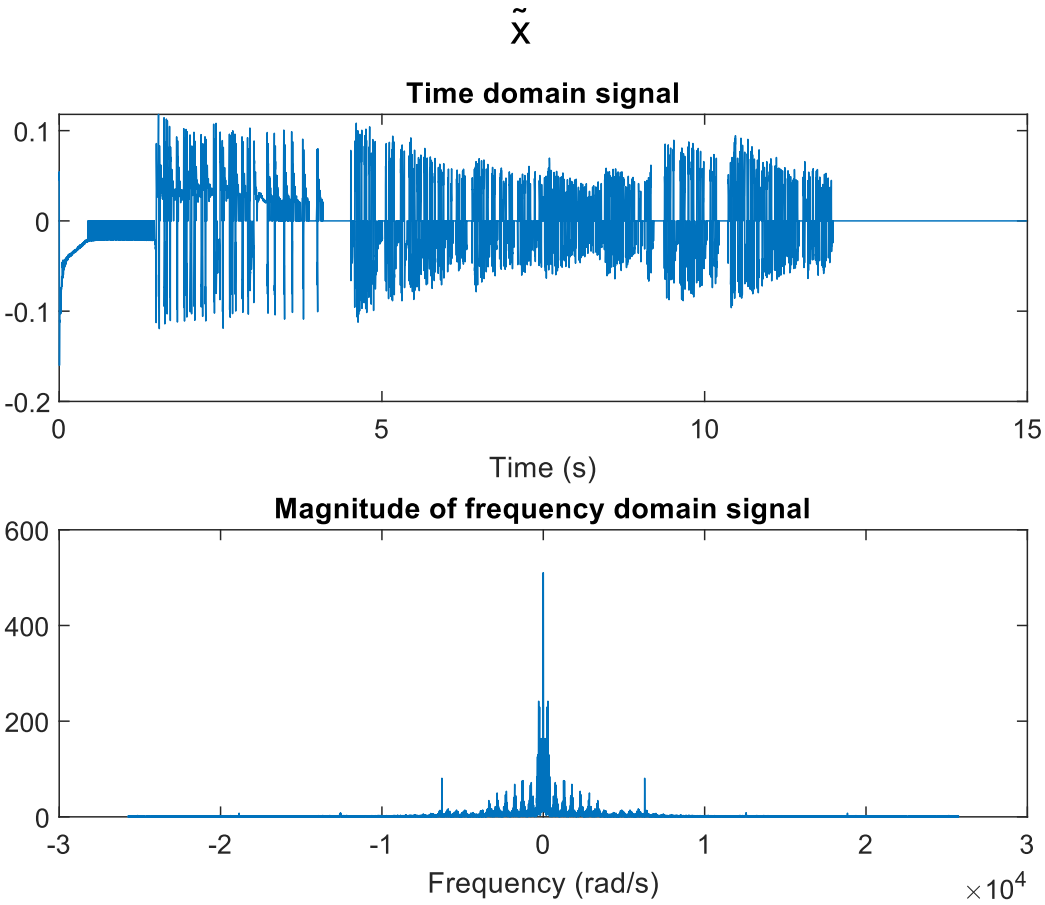

Figure 4: Time and frequency domain representations of $\tilde{x}$ after the small amplitude signals are turned into zero. The frequency response resembles that of the original signal.

Once the original frequency response is restored, a signum function is applied to $\tilde{x}$ to create $\tilde{x}_{sign}$ in order to turn all of the signals into 1, or -1 so that bits can be extracted from the signal later on.

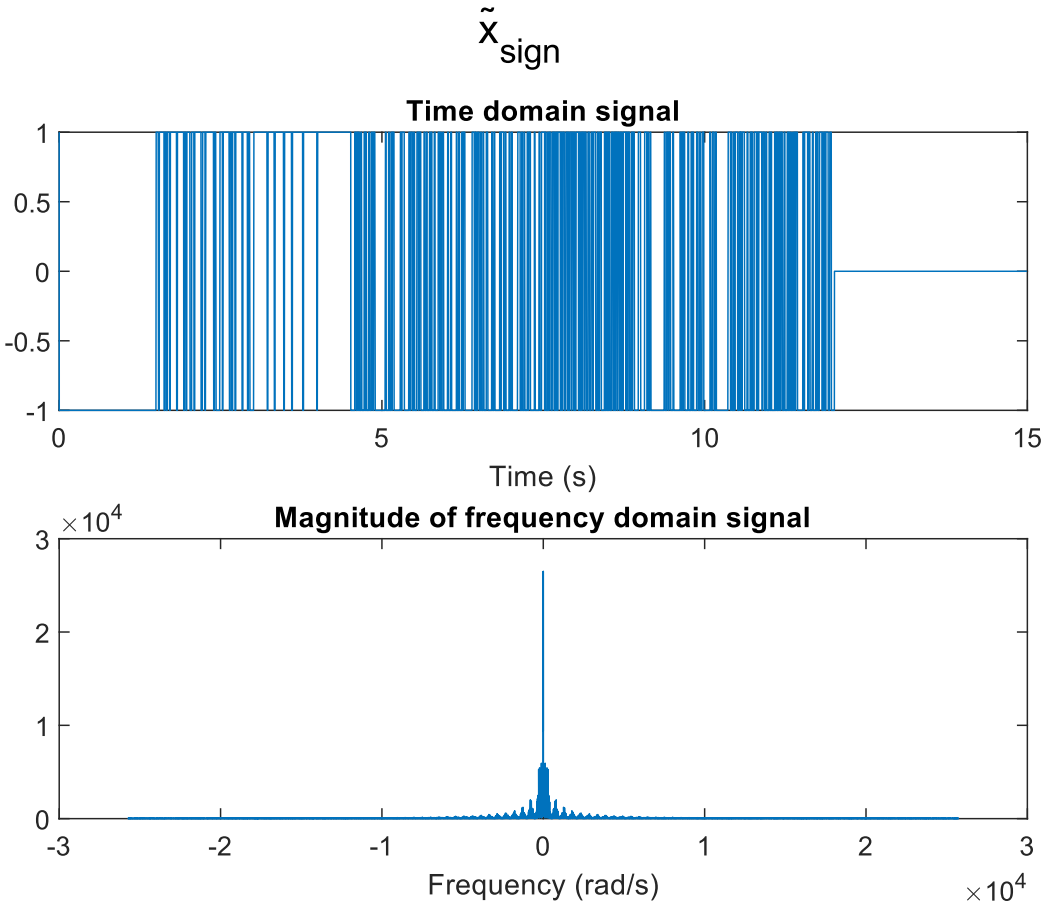

Figure 5: Time and frequency domain representations of $\tilde{x}_{sign}$

Then, every 100th signal starting at index 50 from $\tilde{x}_{sign}$ is sampled (x_{samp}) to represent the bits that were encoded in the transmission. Since the values contained in x_{samp} will be 1 or -1, the following equation is applied to convert the samples to 1 or 0 and create x_d.

$$x_d = (x_{samp} + 1)/2 \tag{2}$$

Finally, x_d is converted to a string to reveal the encoded message in the transmission.

3 Demo Video

Link to YouTube demo video

4 MATLAB Code

```
close all
load("long_modem_rx.mat")
```

```
% The received signal includes a bunch of samples from before the
% transmission started so we need discard these samples that occurred before
% the transmission started.

start_idx = find_start_of_signal(y_r,x_sync);
% start_idx now contains the location in y_r where x_sync begins
% we need to offset by the length of x_sync to only include the signal
% we are interested in
y_t = y_r(start_idx+length(x_sync):end); % y_t is the signal which starts at the beginni


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Decoder
% create a cosine wave to multiply the signal by
c = cos(2*pi*f_c/Fs*[0:length(y_t)-1]');
t = [0:length(y_t)-1]./Fs;

figure
subplot(211)
plot(t,y_t);
xlabel("Time (s)")
title('Time domain signal');
subplot(212);
plot_ft_rad(y_t,Fs);
title('Magnitude of frequency domain signal');
sgtitle("y_T")

% multiply the signal by the cosine
y_c = y_t.*c;

figure
subplot(211)
plot(t,y_c);
xlabel("Time (s)")
title('Time domain signal');
subplot(212);
plot_ft_rad(y_c,Fs);
title('Magnitude of frequency domain signal');
sgtitle("y_C")

% apply a lowpass filter to the result
x_t_tilde = lowpass(y_c, f_c, Fs);

% convert knwon noise to zeros
x_t_tilde(msg_length*8*100:end) = 0;
```

```
x_t_tilde(abs(x_t_tilde) < 0.02) = 0;

figure
subplot(211)
plot(t,x_t_tilde);
xlabel("Time (s)")
title('Time domain signal');
subplot(212);
plot_ft_rad(x_t_tilde,Fs);
title('Magnitude of frequency domain signal');
sgtitle("x")

% convert all nonzero values to 1 or -1
x_t_tilde_sign = sign(x_t_tilde);
previous_sign = 0;

% convert zeros to 1 or -1 depending on previous value
for i=1:msg_length*8*100
    if(x_t_tilde_sign(i) == -1)
        previous_sign = -1;
    elseif (x_t_tilde_sign(i) == 1)
        previous_sign = 1;
    else
        x_t_tilde_sign(i) = previous_sign;
    end
end

figure
subplot(211)
plot(t,x_t_tilde_sign);
xlabel("Time (s)")
title('Time domain signal');
subplot(212);
plot_ft_rad(x_t_tilde_sign,Fs);
title('Magnitude of frequency domain signal');
sgtitle("x_{sign}")

x_samples = zeros(msg_length*8, 1);
for i=1:size(x_samples, 1)
    x_samples(i) = x_t_tilde_sign(i*100 - 50);
end

% convert the vector to zeros and ones
x_d = (x_samples+1)./2;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% convert to a string assuming that x_d is a vector of 1s and 0s
% representing the decoded bits
BitsToString(x_d)
```