Sam Wisnoski, Oscar Bao
Spring 2025
ESA Signals (ENGR2410) with Professor Xuan Kong

Acoustic Modem Project

I. Project Overview

In this project, our goal is to implement a receiver for an acoustic modem, which is a device that sends (and receives!) digital signals using sound waves. This approach was used over the internet for many years, and is still used in underwater communications today.

The basic idea here is to use amplitude modulation to transmit signals through an acoustic channel. This is completed through a process known as binary-phase-shift keying. This name comes from "binary" because there are one of 2 possible phase offsets to the cosine and "phase" because the information is carried in the phase of the cosine. This signal is then transmitted over the air, and when the signal is received, we can use the principles we learned in Engineering Systems Analysis: Signals to decode it.

In our project, we will be working with pre-generated over the air signals, and we just need to demodulate, filter, and decode them. Our first goal is to decode the word "Hello", and our second code is a secret mystery to be discovered.

In our report, we will cover the methodology and our project results. All code can be found at the end of our document, in the appendix.

II. Methodology

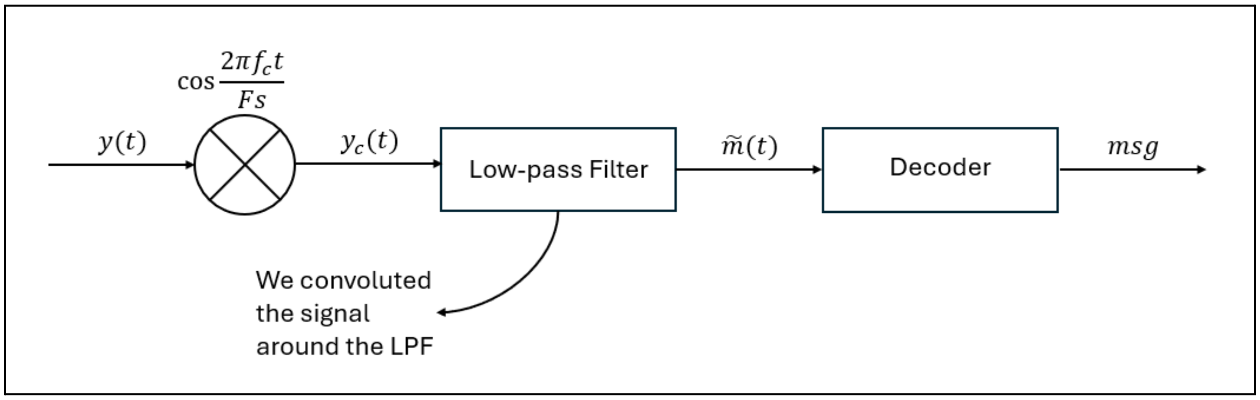
$\cos\frac{2\pi f_c t}{Fs}$
$y(t)$
$y_c(t)$
Low-pass Filter
$\tilde{m}(t)$
Decoder
msg
We convoluted the signal around the LPF

Figure 1. Block Diagram of Our Acoustic Modem Receiver. y(t) is the signal we receive from the transceiver, and yc(t) is our demodulated signal, $\tilde{m}(t)$ is the filtered signal, and msg is the resultant transmitted message.

We want to begin our methodology with an analysis of our transmitted signal, which will give us helpful insight into how to decode it back to the original. The original symbol, $m(t)$, is a bit signal, where each pulse represents a 1, and each non-pulse represents a 0. In the transmitter, the original signal is multiplied by $cos(\frac{2\pi f_c t}{F_s})$ before being transmitted through the air. We will call this transmitted signal $x(t)$.

$$x(t) = m(t) \cdot cos(\frac{2\pi f_c t}{F_s}) = y(t)$$

This signal is transmitted through the air and will eventually reach our receiver. We will call this received signal $y(t)$. This is where we start in the project—we begin with the pre-generated signal $y(t)$. We began the process of decoding the signal by first multiplying the received signal with a cosine function as shown in the block diagram. The equation for this is:

$$y_c(t) = y(t) \cdot cos\frac{2\pi f_c t}{F_s} = m(t) \cdot cos(\frac{2\pi f_c t}{F_s}) \cdot cos(\frac{2\pi f_c t}{F_s}) = m(t) \cdot cos^2(\frac{2\pi f_c t}{F_s})$$

After the multiplication, we can use trigonometric properties to break our new signal, $y_c(t)$, into two components: a baseband component containing the original signal information, and a high frequency trigonometric component.

To get rid of this high frequency component and reduce our signal back to the original signal, we can then convolve the signal around a low-pass filter. Specifically, using the frequency convolution theorem, we know that for two signals in time domain, $x_1(t)$ and $x_2(t)$, and their Fourier transformation $X_1(\omega)$ and $X_2(\omega)$, $x_1(t) \cdot x_2(t) = \frac{1}{2\pi}[X_1(\omega) * X_2(\omega)]$. This forms the basis of our low-pass filter.

After the signal is filtered, we have a baseband output $\tilde{m}(t)$ that roughly resembles the original signal, which we can then decode to the original bitstring by extrapolating if it was supposed to be a one or zero. Lastly, we can run the BitstoString function to convert our binary signal into our original component.

Next, we will walk through the results of our project.

III. Results

Note: Our full code can be found in Section V, the Appendix.

Short Message

To test our acoustic filter, we began with the short message, since we already knew what to expect for our output: “Hello”. With each step in the process, we generated graphs to track our progress. In our report, we will walk through each step and show results.

First, we plotted the incoming signal in both the frequency and time domains:

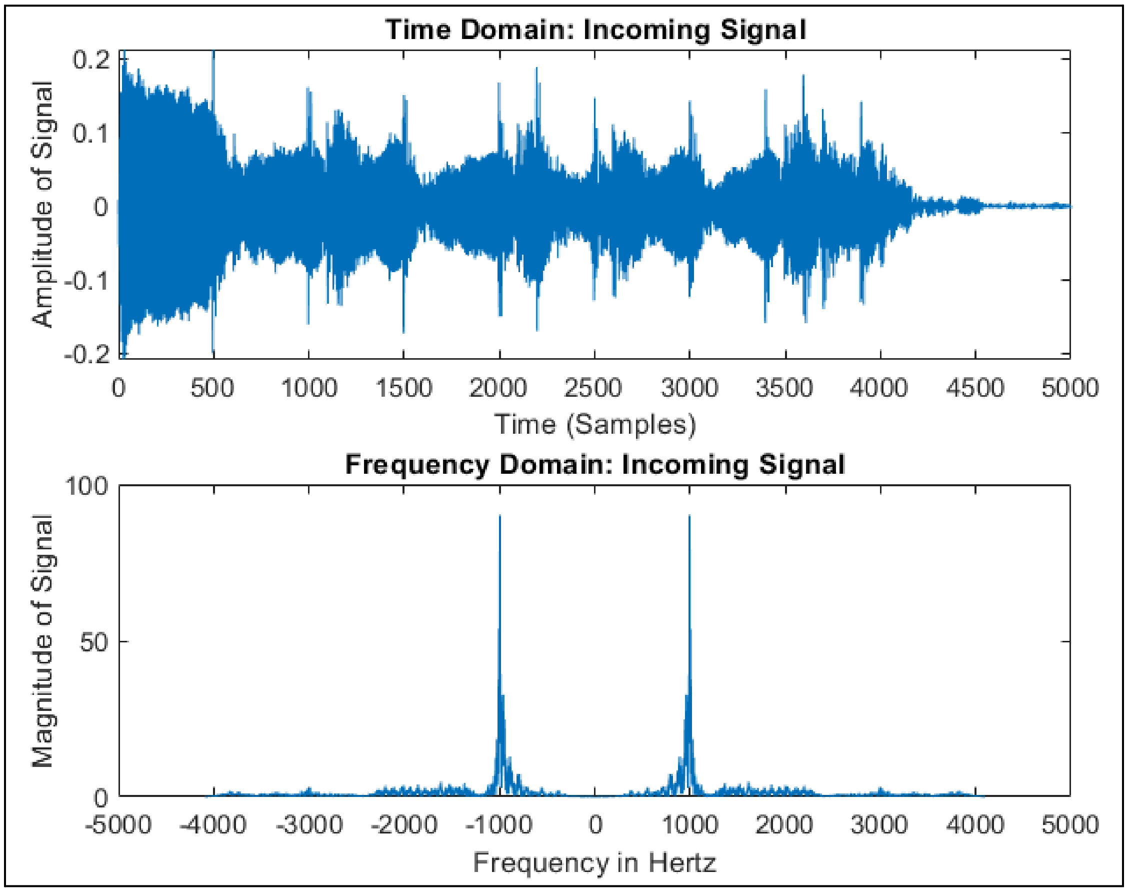

Figure 2. Incoming Signal in Time Domain and Frequency Domain (Short Message)

Clearly, we see the time domain signal is a mess, but we can clearly see the cosine in the transmitted signal. Next, we demodulated the signal by multiplying it by cos $(\frac{2\pi f_c}{F_s} * t)$.

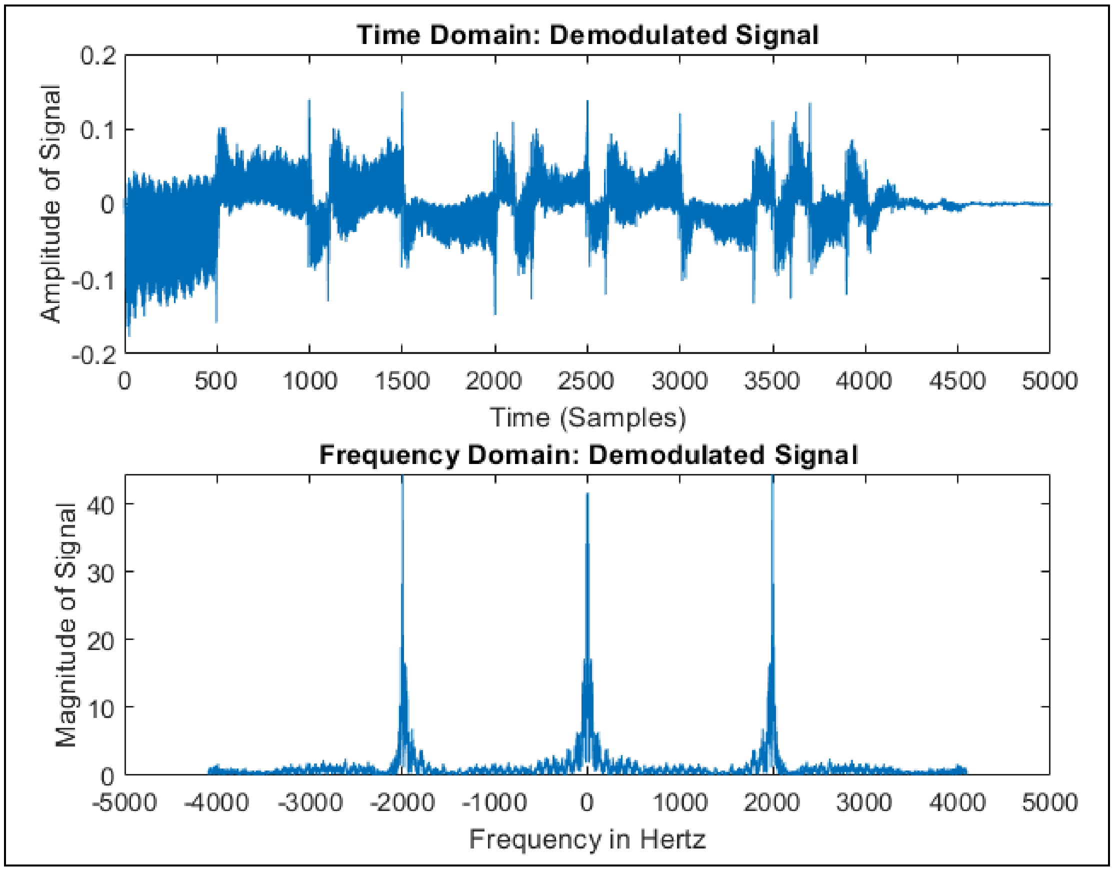

Figure 3. Demodulated Signal in Time Domain and Frequency Domain (Short Message)

As seen above, the signal is less distorted in the time domain, and the binary-phase-shift keying is much more obvious. Additionally, we now have a spike in magnitude of our frequency in the time domain around Hz = 0, which is our original signal, while the two spikes on the side are due to the cos squared. This means we can use a low-pass filter to filter out the high frequency trigonometric component. We create a low-pass filter using the sinc function, and we can see the low-pass filter visually below by taking the fourier transform and displaying it in the time domain. We choose a low-pass frequency of 1000 in order to properly filter out the signal.

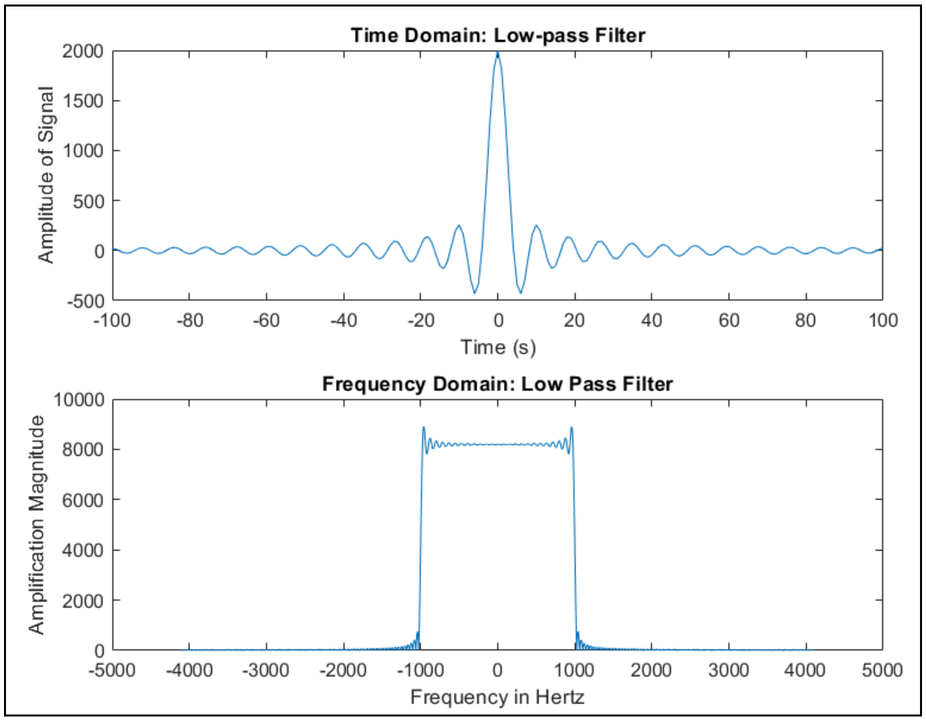

Figure 4. Our low-pass filter in the time and frequency domains (Short Message)

Next, we can colvolve our signal with the low-pass filter in order to filter out the high frequency component. This leaves us with a recreation of our original signal, seen below.

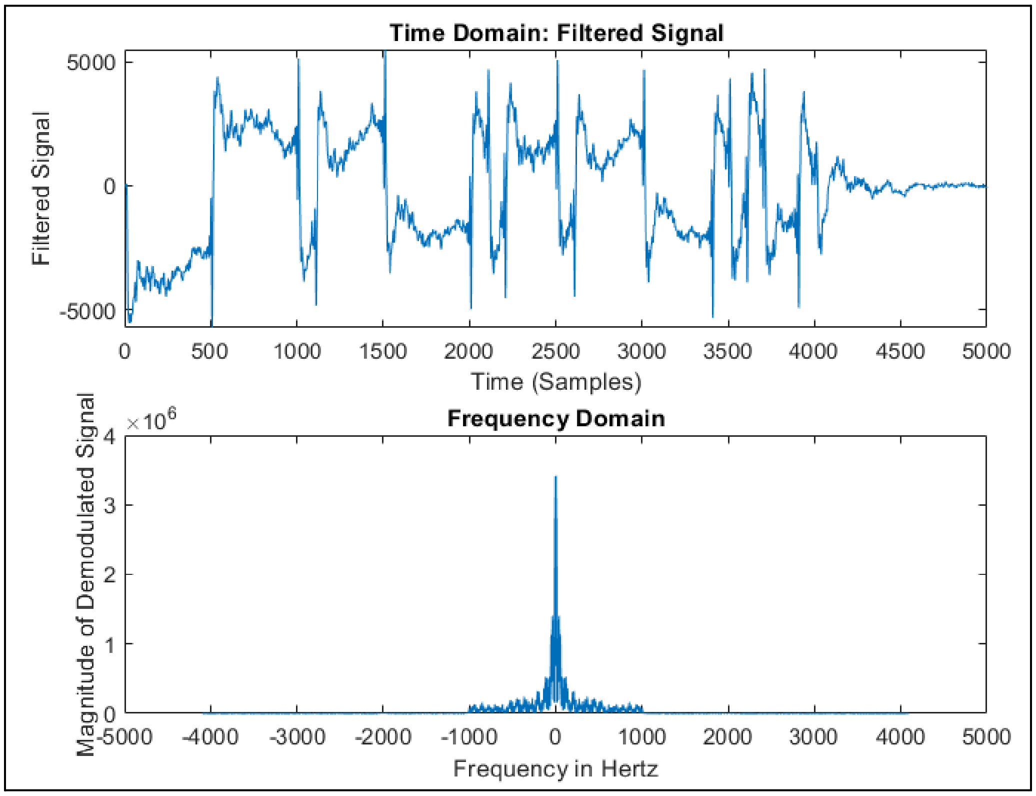

Figure 5. Our Filtered Signal in the Time and Frequency Domains (Short Message)

After filtering our signal, we have a rough estimation of the original signal. At each phase shift, we can estimate if the signal is a one or a zero and convert back to the original bitstring. After running our result through the BittoString function, we get our result:

```
message = 'Hello'
```

Mission success!

Long Message

Next, we will follow the same process for our longer message.

First, we display our incoming signal:

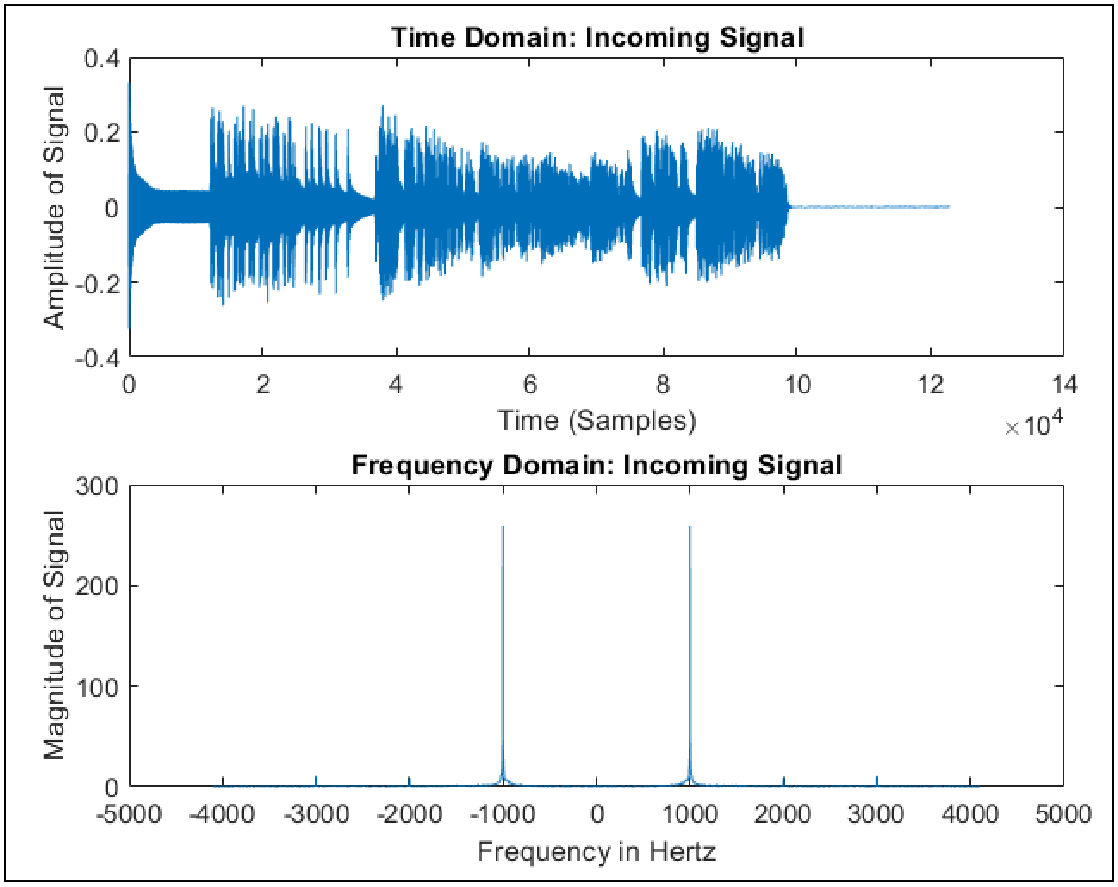

Figure 6. Incoming Signal in Time Domain and Frequency Domain (Long Message)

Then, we demodule by multiplying by cosine:

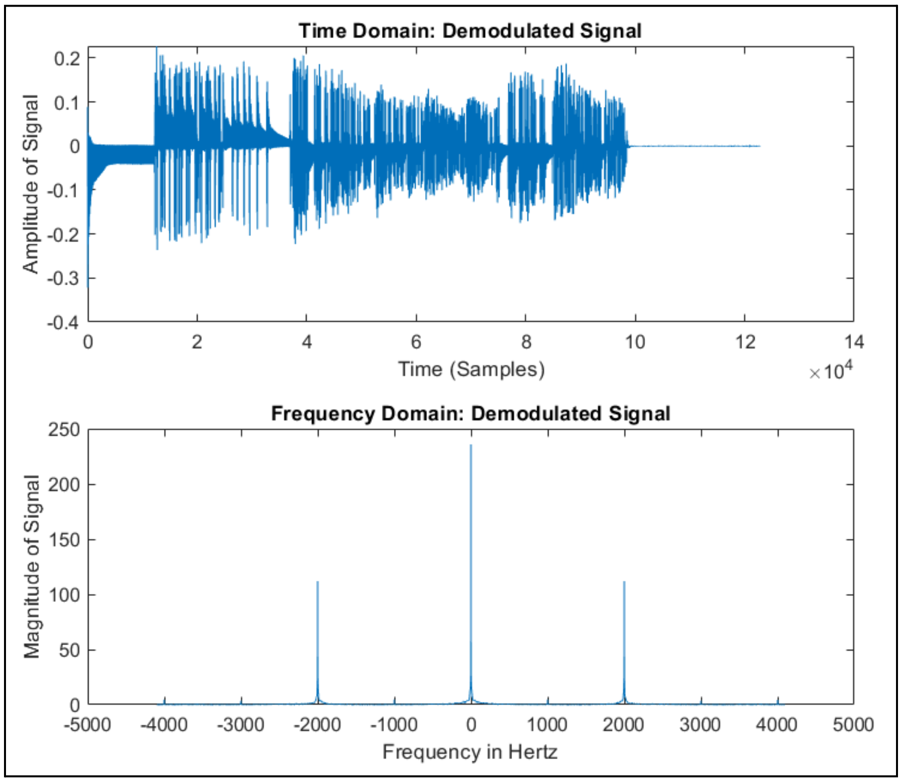

Figure 7. Demodulated Signal in Time and Frequency Domain (Long Message)

Lastly, using the same low-pass filter as defined above, we filter our signal for the final result:

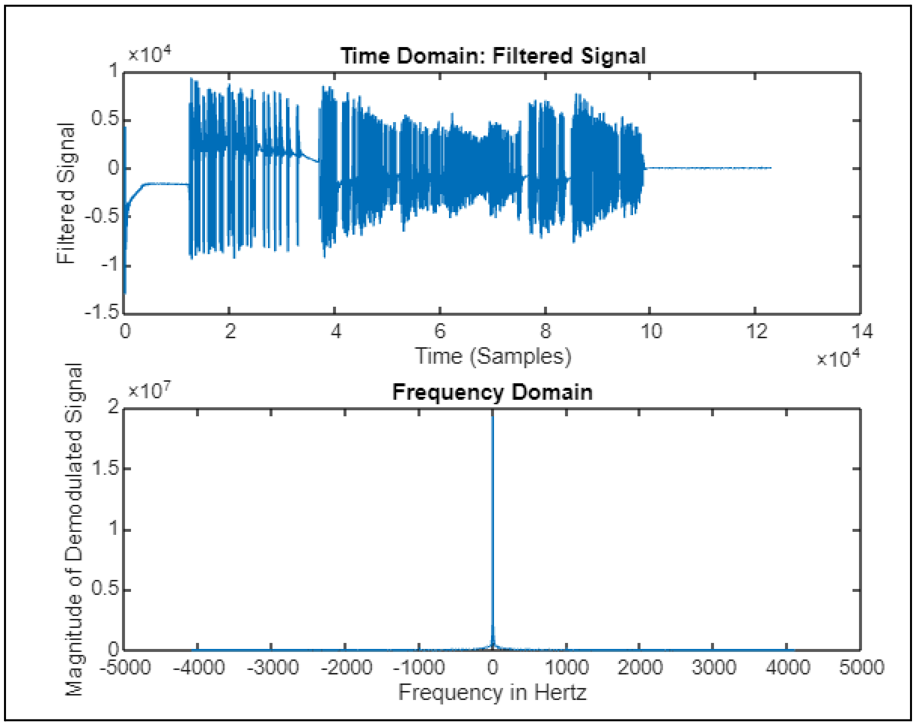

Figure 7. Filtered Signal in Time and Frequency Domain (Long Message)

Lastly, we decode and get our long message:

```
message = 'The answer to the Ultimate Question of Life, the Universe, and Everything is 42.

The question to the ultimate answer is ...'
```

Looks like another success!

To conclude our results, we have created a brief overview of our running code. The video can be watched here.

IV. Conclusion

Through this project, we were able to understand the concept of CTFT and frequency modulation better. It truly has been fascinating learning how deeply ingrained Fourier transformation is in our lives.

V. Appendix

Here is our entire code for our Acoustic Modem Reciever:

```
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 beginning of the transmission;

y_t_omega = fftshift(fft(y_t, Fs+1));


clf

subplot(2,1,1)
```

```
plot(y_t)

title("Time Domain: Incoming Signal")

xlabel("Time (Samples)")

ylabel("Amplitude of Signal")

% xlim([0,5000]) %If plotting short signal


subplot(2,1,2)

plot((-Fs/2:Fs/2), abs(y_t_omega))

title("Frequency Domain: Incoming Signal")

xlabel("Frequency in Hertz")

ylabel("Magnitude of Signal")


symbol = 100;

omega = 2*pi*f_c/Fs;

cos_down_shifting = cos(omega*[0:length(y_t)-1]') ;

y_demodded = y_t .* cos_down_shifting;

y_t_demodded_omega = fftshift(fft(y_demodded, Fs+1));


clf

subplot(2,1,1)

plot(y_demodded)

title("Time Domain: Demodulated Signal")

xlabel("Time (Samples)")

ylabel("Amplitude of Signal")

% xlim([0,5000]) %If plotting short signal


subplot(2,1,2)

plot((-Fs/2:Fs/2), abs(y_t_demodded_omega))
```

```
title("Frequency Domain: Demodulated Signal")
xlabel("Frequency in Hertz")
ylabel("Magnitude of Signal")
%% Low Pass Filter
omega_w = 2*pi*1000;
time = (-100:1:100)*(1/Fs);
low_pass_filter_t = omega_w/pi*sinc(omega_w/pi*time);
low_pass_filter_omega = fftshift(fft(low_pass_filter_t, Fs+1));

clf
subplot(2,1,1)
plot((-100:100), low_pass_filter_t)
title("Time Domain: Low-pass Filter")
xlabel("Time (s)")
ylabel("Amplitude of Signal")

subplot(2,1,2)
plot((-Fs/2:Fs/2), abs(low_pass_filter_omega))
title("Frequency Domain: Low Pass Filter")
xlabel("Frequency in Hertz")
ylabel("Amplification Magnitude")

x_d = 10*conv(y_demodded, low_pass_filter_t);
x_d = x_d(90:end);
x_omega = fftshift(fft(x_d, Fs+1));

figure
subplot(2,1,1)
```

```
plot(x_d)
title("Time Domain: Filtered Signal")
xlabel("Time (Samples)")
ylabel("Filtered Signal")
% xlim([0,5000]) %If plotting short signal

subplot(2,1,2)
plot((-Fs/2:Fs/2), abs(x_omega))
title("Frequency Domain")
xlabel("Frequency in Hertz")
ylabel("Magnitude of Demodulated Signal")

num_symbols = floor(length(x_d)/symbol);
final_message = [];

for i = 1:num_symbols
   start_id = (i-1)*symbol + 1;
   end_id = i*symbol;
   final_message(i) = sum(x_d(start_id:end_id))>0;
End

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% convert to a string assuming that x_d is a vector of 1s and 0s
% representing the decoded bits
final_message = final_message(1:msg_length*8);
message = BitsToString(final_message)
```