Nathan Faber and Thomas Jagielski
May 5, 2021
Signals and Systems: Acoustic Modem

Signals and Systems: Acoustic Modem

Introduction

For this project, we created an acoustic modem receiver. An acoustic modem sends a digital signal encoded in sound waves. Specifically, the digital ‘1’ or ‘0’ is encoded in the phase offset of our sinusoidal signal. To ensure the signal we sent was large enough frequency such that it could be transported through the air, we multiplied by a cosine wave that had a carrying frequency that was sufficiently high.

On the receiving side, we must first identify the beginning of our signal. This was found through a known noise-like signal preceding the waveform that contains the message we are interested in. After this, we can multiply by the same cosine wave used to convert our binary encoded message to the sinusoidal acoustic waveform, apply a low pass filter, and minor post-processing to recover a signal close to the original.

Block Diagram

The block diagram of our entire system can be visualized below. This diagram indicates a theoretical practical implementation for the acoustic modem system, which is not the case we considered with the MVP. For example, in the MVP - as we are not transmitting a signal over the air - there will be no losses due to the H(jω) transfer function. Similarly, we did not need to multiply by a gain G1.

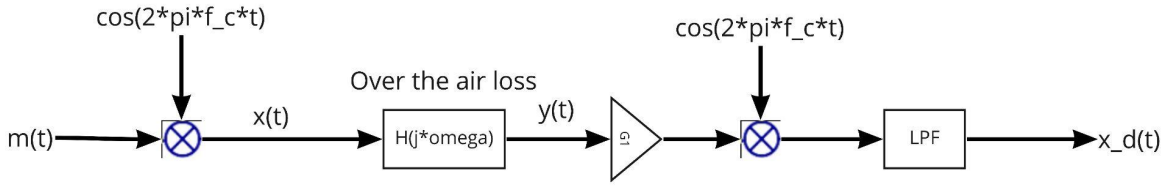

Figure 1: Block diagram for whole acoustic modem. For the MVP system, we assumed there were no losses over the air (H(jω)=1) and G1 was 1.

Implementation-- Sending and Decoding Messages

As mentioned early we will be sending and receiving messages over the air. This process can easily be broken into 2 steps: the transmission and the receiving/processing.

Transmission

While much of the transmission-related part of this project was already done, we want to show what was done as it directly impacts our decisions in processing the signal. We established earlier that the transmission would be text data that is converted into a sequence of binary data. This data needs to be converted into a signal that can be transferred over the air. To accomplish this we turn this sequence of bits into a square wave.
Below are examples with the string “Hello”:

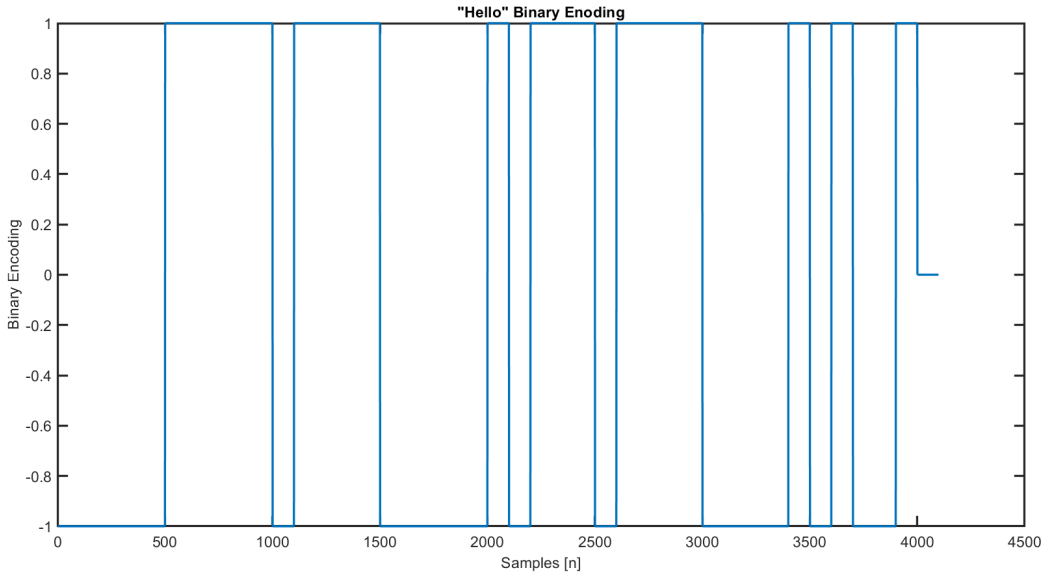

Figure 2: Binary encoding of “Hello” with a sample period of 100 samples.

The above square wave is close to what we need to transmit across the air. Unfortunately, this is too low of a frequency of signal to be transmitted through the air. This frequency is usually much higher. Our implementation uses a carrying frequency of 1000 hz. We create the final signal that is transmitted by multiplying a sinusoid (cosine) of that frequency onto our square wave signal. Specifically, the equation we used for this processing was as follows,

$$c = cos(2 * \pi * \frac{8192}{1000} * t)$$

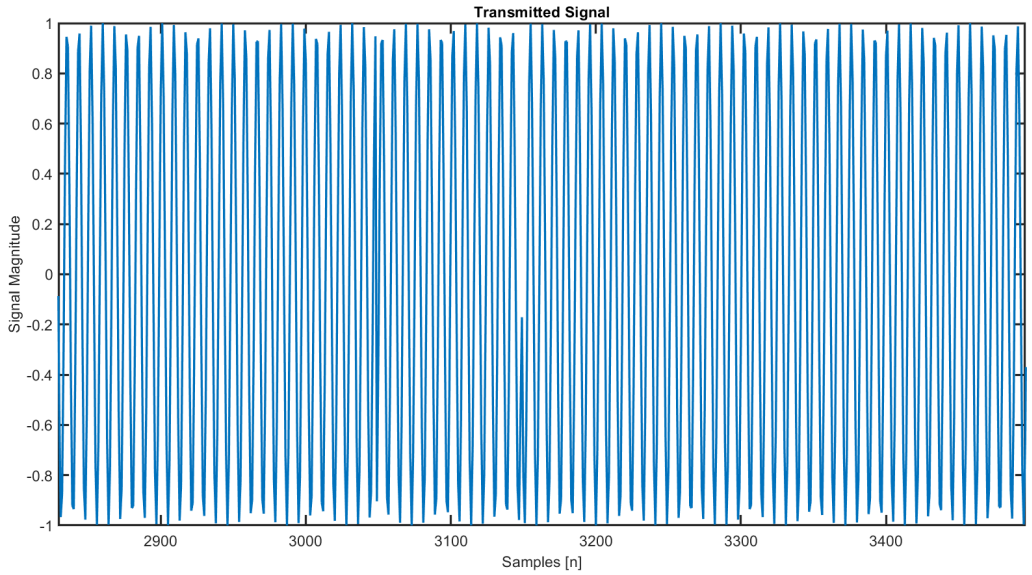

Figure 3: Zoomed in image of transmitted signal to demonstrate phase shift through scaling by binary encoding.

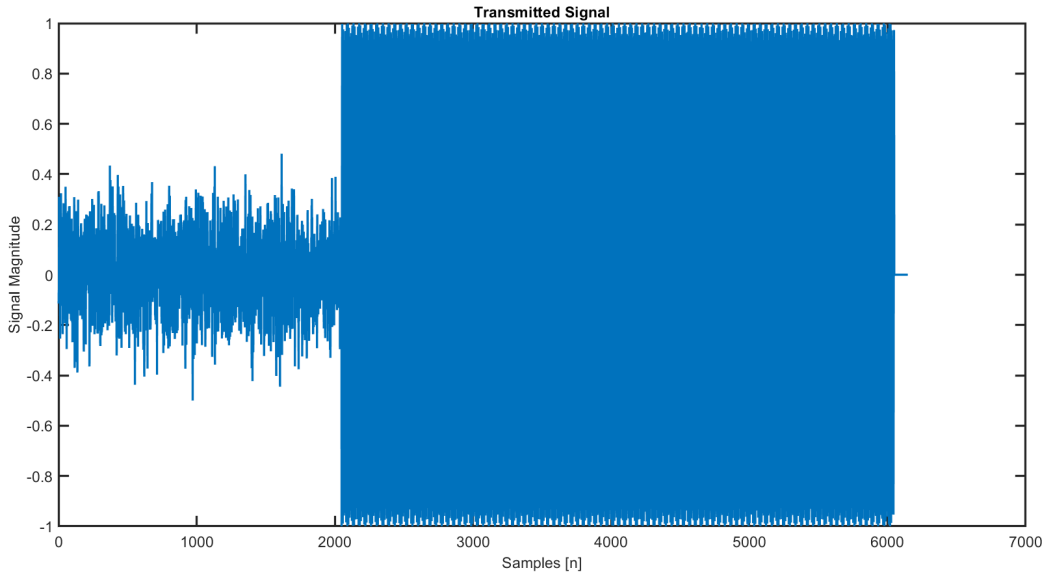

Figure 4: Transmitted signal with start noise and binary encoding for “Hello”

We also care about what the frequency content of this signal is as we are using amplitude modulation.

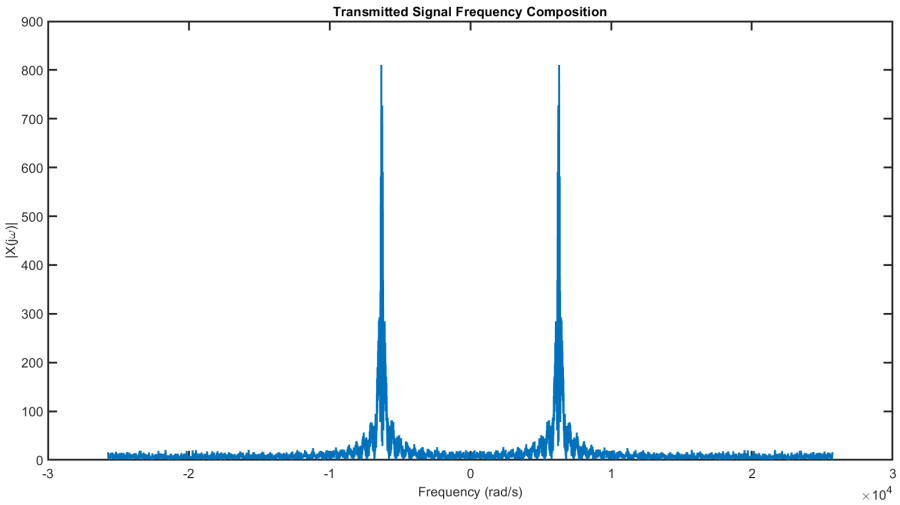

Figure 5: Frequency domain plot for transmitted signal.

There is one final addition to this signal before it is transmitted. That piece at the beginning of a signal that is very different from the informative part of our signal. It is used as noise to indicate the start of the meaningful signal. This noise is added so that we can accurately find the start of the signal when we are receiving it.

Receiving and Processing

The receiving part of this process is certainly more complicated than the transmission. The transmitted signal is read in via a microphone.

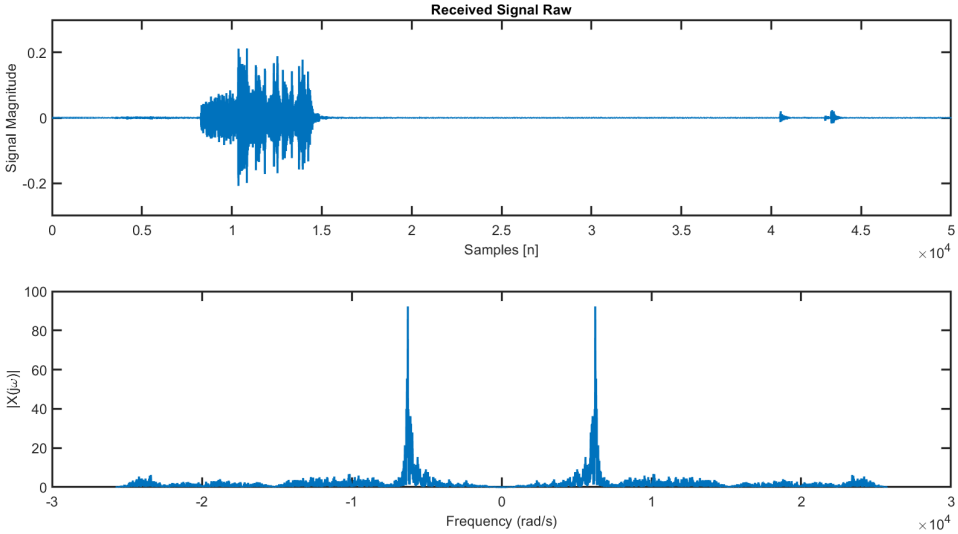

Figure 6: Received signal in the time and frequency domain.

The first step in the processing of this signal is to find the starting point of the transmission. In the transmission we intentionally added a noisy block at the beginning and by using cross-correlation we are able to find the point at which the data-containing part of the signal starts. We then trip the signal to this section.

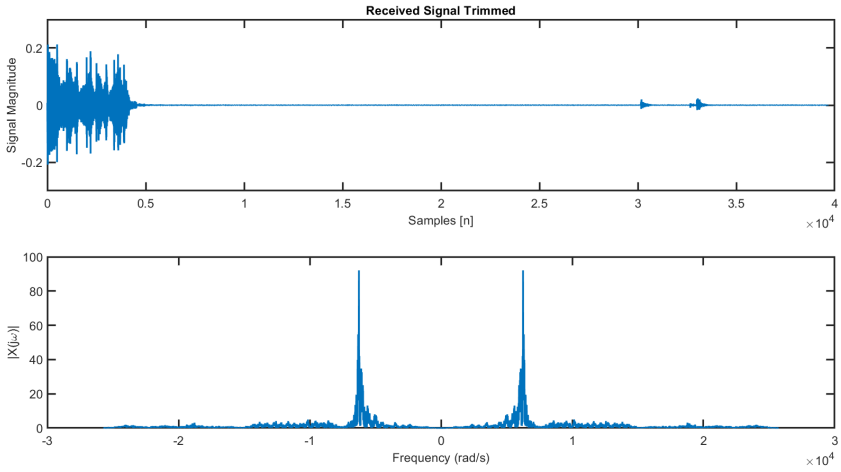

Figure 7: Signal trimmed to start of meaningful message

Following our block diagram, we next multiply the remaining signal by the same sinusoid that was done in the transmission. This allowed us to recenter the signal about the original frequency of the encoded signal. In doing this, we also expect the frequency content to be clustered around zero at this point as we have multiplied by the same cosine twice.

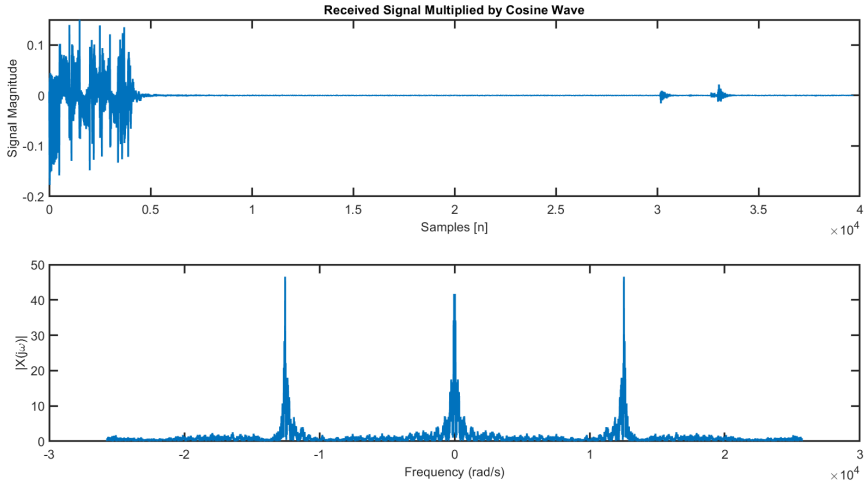

Figure 8: Time and frequency domain representation of signal processed by sinusoid.

The next step in our block diagram requires us to Low-pass filter this signal (needed because we multiplied by cosines). We experimented with multiple different filtering styles, using matlab's

butter function, however we found that convolving with a sinc function seemed to yield better results.

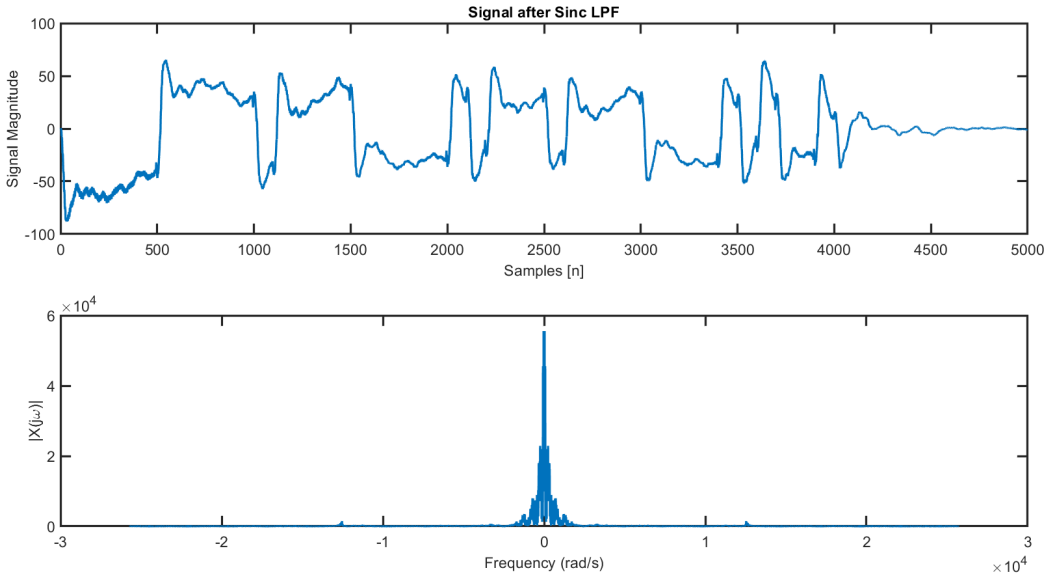

Figure 9: Signal in time and frequency domain with low pass filter applied

Through experimentation, we found a low pass filter with the following equation seemed to work best,

$$h = \frac{200}{\pi} * sinc(\frac{200}{\pi * t})$$

Thus, this forms a square wave in the frequency domain which allows us to process a signal within the range we are interested in. By doing this, we begin to have a signal that looks much more similar to what we began with. The next step of processing this signal is turning it into something that is bounded between 0 and 1 (a boxy plot) as opposed to a continuous plot. This is done by defining a signal over zero to be 1 and a signal below zero to be -1 (the binary equivalent of 0).

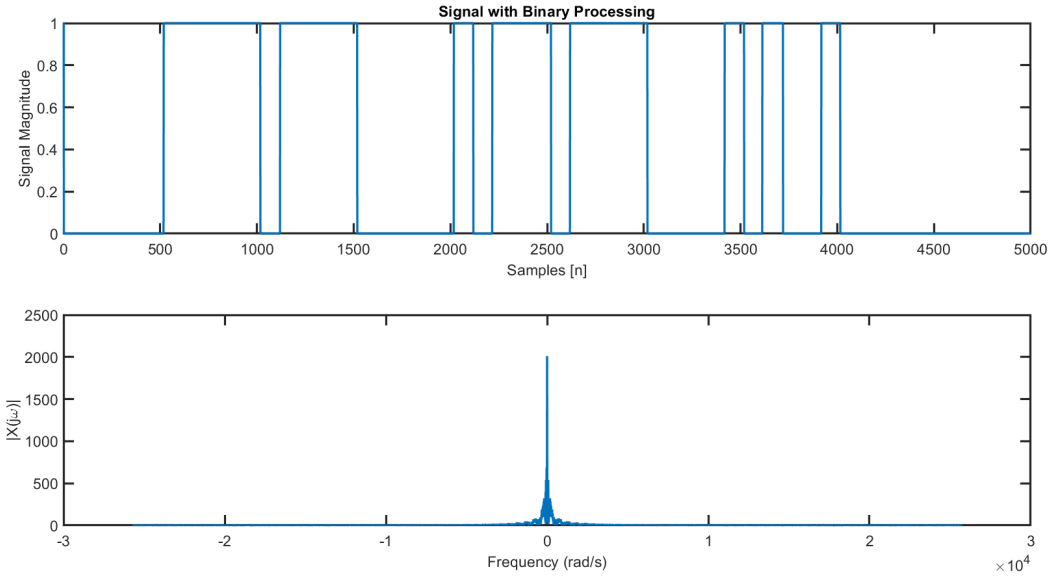

Figure 10: Processed signal to convert analog signal into discrete ones and zeros.

With any luck this should look like a slightly shifted version of the original sent box plot. This signal still needs to be down sampled to find the bits that each box corresponds to. The first part of that problem is identifying the offset of the signal. This is achieved by calculating the average peak and doing a modulus operation by the signal period. The returned value should be the offset.

Next this signal must be sampled. We choose to sample in the middle of each sampling period. In our case that means we sample the 50th point, the 150th, the 250th etc.
One track part about this is that locating the end of the signal transmission is difficult. In our attempts we have chosen to make use of knowing the length of message that we are looking for. However, we believe that an additional step could be applied to search for the end of the signal before it is discretized. As the amplitude of the signal is attenuated greatly.

This downsampling process leaves us with a list of bits. We simply make use of a conversion from bits to a string (provided) to get the final output message!

__Final Results:__
With the receiver we implemented, we decoded the short_modem_rx.mat file as “Hello”. The decoded message from the long_modem_rx.mat file was “The answer to the Ultimate Question of Life, the Universe, and Everything is 42. The question to the ultimate answer is …”

Code

Our entire github repository can be found at the following link:
https://github.com/ThomasJagielski/signals_final_project

The MATLAB code used that implements our receiver can be found at the following link: https://github.com/ThomasJagielski/signals_final_project/blob/main/modem_rx_starter.m

Demonstration Video

A video demonstration of our working acoustic modem receiver can be found that the following link: https://youtu.be/tp4ITKuJ098