ADComm Final Report:
Audio Modem and Steganography

Claire Barnes, Victoria Coleman, Kevin O'Toole, Matt Wismer

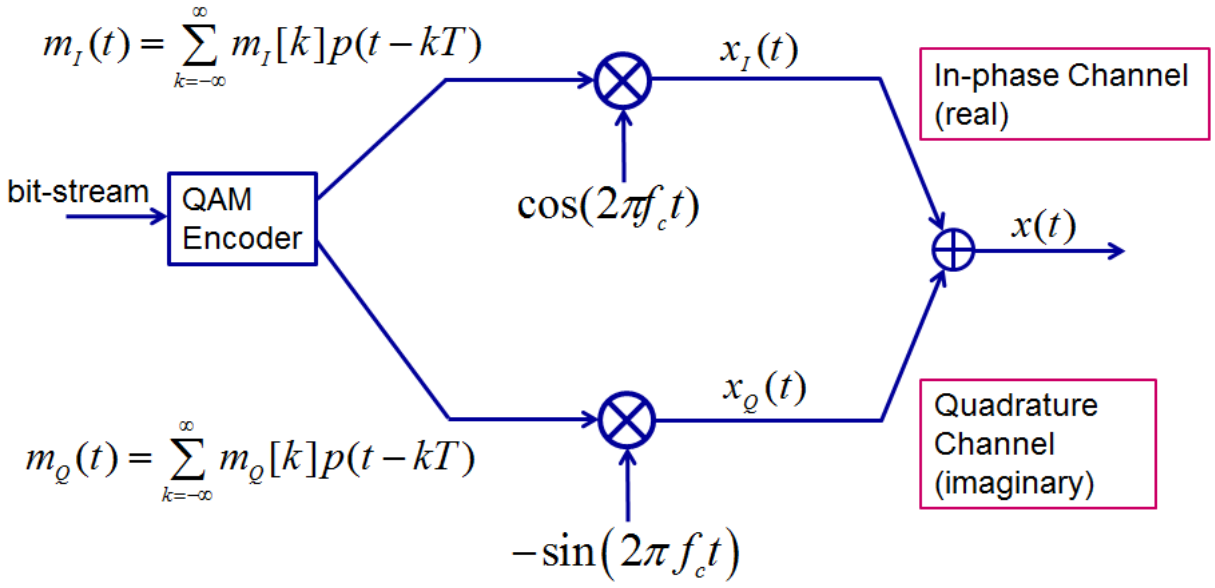
Intro

We created a 4-QAM audio modem with python that transfers information in frequencies in the audible spectrum from one laptop to another with native microphones, speakers, and sound cards. Our stretch goal was to implement this system steganographically. Humans naturally filter out a certain amount of noise when listening to a sound that makes it seem closer to a pure tone. This is why when we sing or play an instrument, we don't have to produce perfect, pure tones in order for the music to sound good. This is called a masking threshold, which is how much noise can be present with a tone that will be naturally filtered out by the human ear. This threshold is a function of frequency and is much lower at low frequencies and relatively high at higher frequency. This characteristic is why humans can differentiate between lower frequencies much more easily than higher ones. A steganographic audio modem would transmit audible data that is played with a song that masks the data so a human can't perceive the difference between the song with or without the data added to it.

We were successfully able to transmit data (lyrics to a song encoded in 8bit binary) in the audible spectrum with a data rate of 100 bits/second with an error of about 0.9%. Independently, we were also able to create a system that disguises encoded signals in the audible spectrum such that the listener would not be able to discern a difference in music due to the signal being added. The original intent of this was to make a fully steganographic modem; however, due to the limited time we had, we never successfully implemented decoding of messages that were encoded and disguised with the music.

System Descriptions

Transmitter



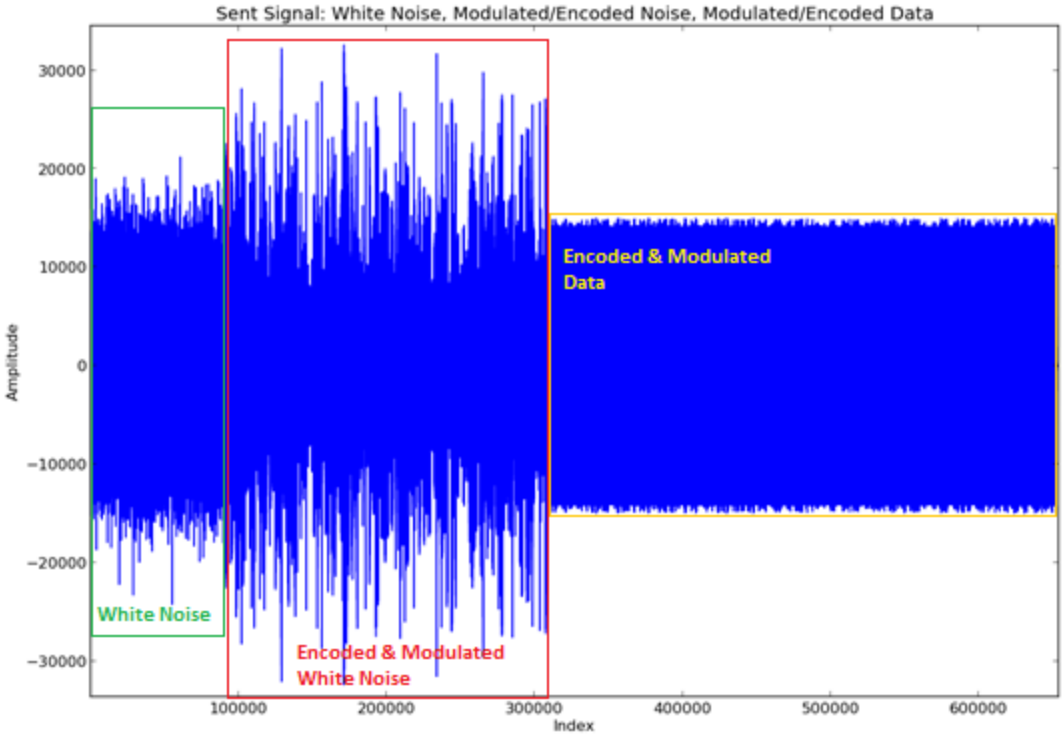
To transmit the data, we used a 4-QAM system. The transmitter takes in a bit-stream, which for this project is a binary string representing lyrics to the song being played (8bit format). Since this project does not do real-time processing, we did QAM encoding by putting the first half of the data bits through the In-Phase Channel, and the second half in the Quadrature Channel. We used a carrier frequency of 1100 Hz to modulate the noise. Not only did this allow us to implement the 4-QAM system, but we are restricted to the physical limitations of the speakers that can only convey essentially amplitude differences. The pulse that we used was a raised cosine to help limit the noise and effectively implement a no-ISI system:

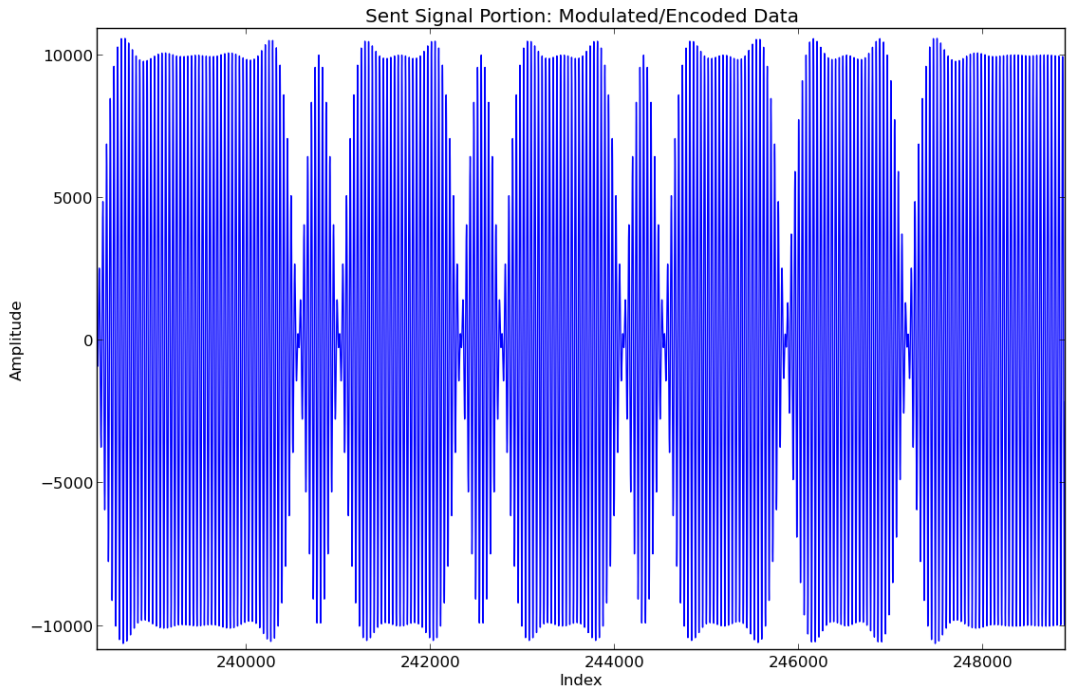
$$p(t) = sinc(\tfrac{t}{T})\frac{cos(\frac{\pi\beta t}{T})}{1-\frac{4\beta^2 t^2}{T^2}} \quad where\ \beta = 1$$

As a note, at one point, this equation divides by zero, so that exception had to be accounted for. The width of the pulse was proportional to the data rate so the peak is over the zero of the previous pulse to prevent ISI (the pulses add to a constant).

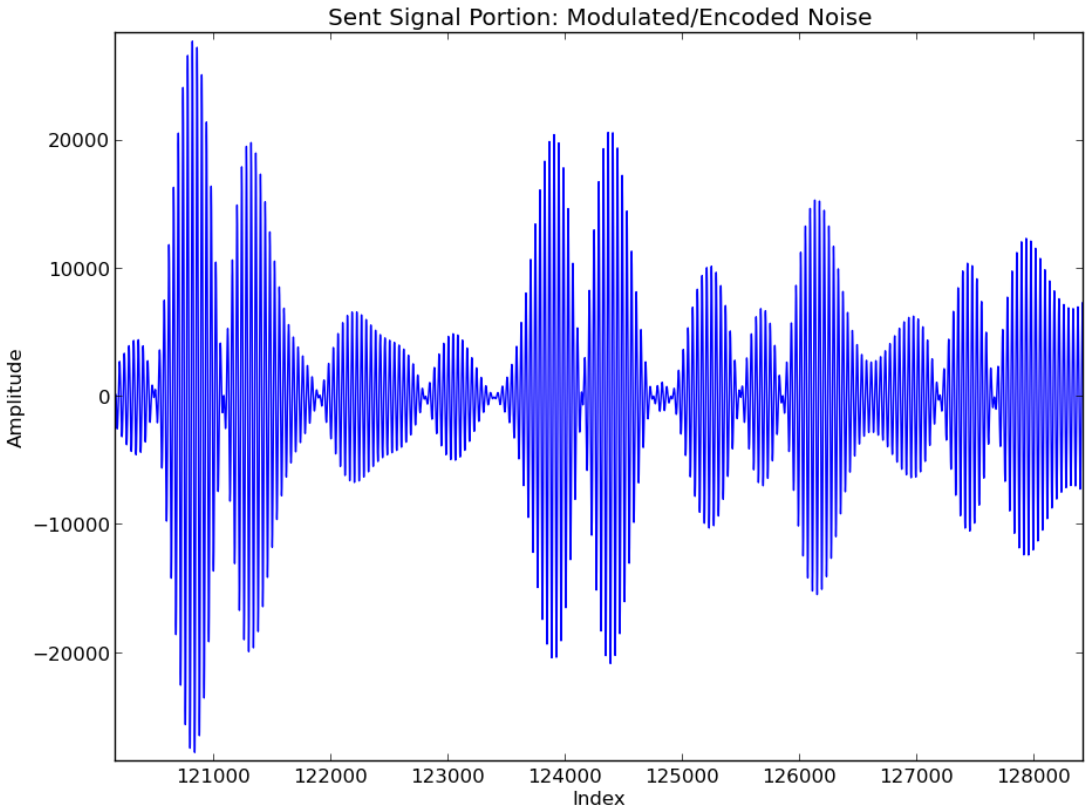
To characterize the channel and get the timing correct, we had to send a signal with three

portions: pure white noise to determine the delay of the channel, modulated and encoded white noise to characterize the channel, and the data in the form of -1 and 1. The white noise was generated by seeding the random number generator with a known value so the receiver can correlate it with the same values. Two seconds of the white noise was initially sent, and since the audio card samples at 44100 Hz, it sent 88200 points of data. For the rooms that we tested in, we found that using much less noise resulted in a poorer performance in determining the delay. Next, 500 values of white noise was sent through the entire system for channel characterization, followed by the data.





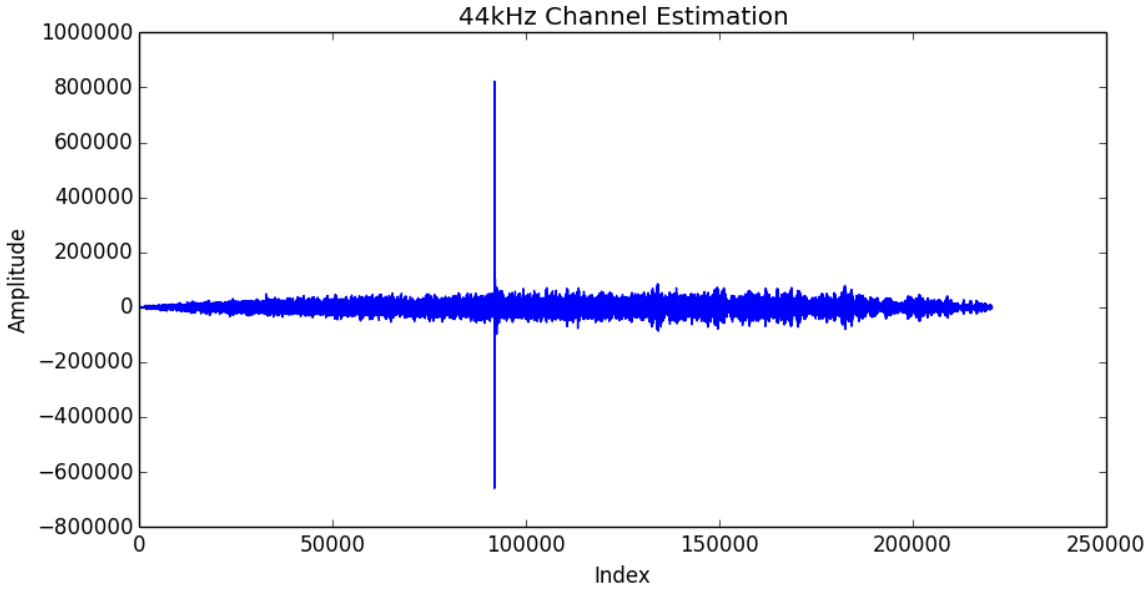
This figure is a zoom of the modulated and encoded data bits. This part of the signal is the addition of the sine and cosine modulated signals. Since the first half of the bits are modulated with cosine and the second half with sine, the first bit is the first bit and the length of the data-divided-by-two's bit.



This portion is white noise passed through the entire transmission system, so it is 4-QAM encoded like the data and modulated by sine and cosine.

44kHz Channel Estimate

At the beginning of sending data, we sent 2 seconds (88200 samples) of pure white noise. We cross-correlated the received data for the first 3 seconds with the known noise values, which we got by seeding the random number generator in numpy the same as the transmitter. The maximum of the output of this cross-correlation was used to align the received samples since that was when the received data best matched the expected data. This portion of the receiver was the most computationally time-consuming. We then discarded the noise samples, so that we could only decode the samples that had been encoded.

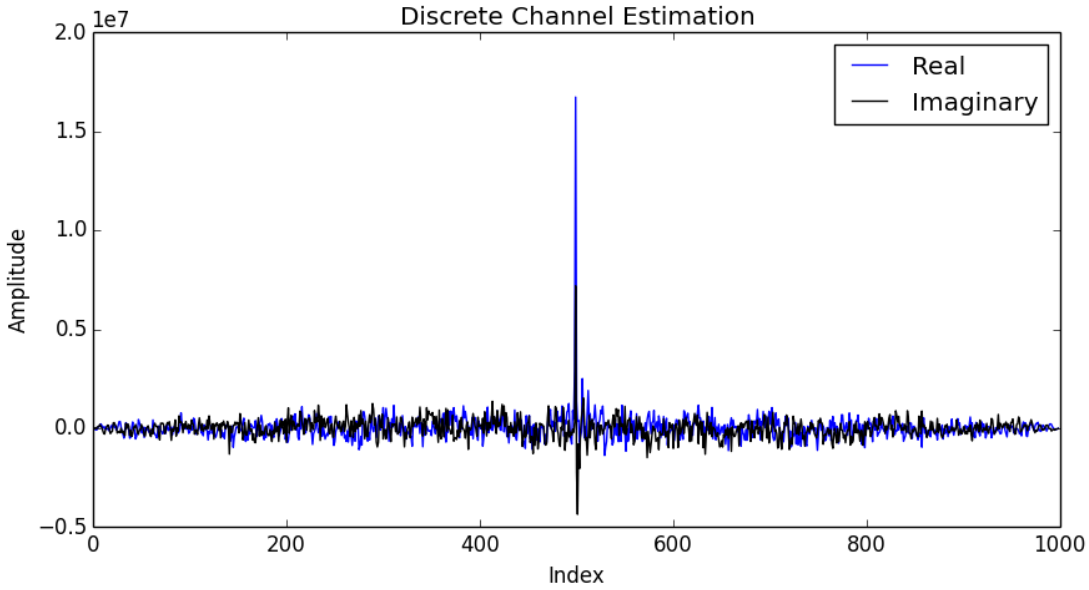


QAM decoding

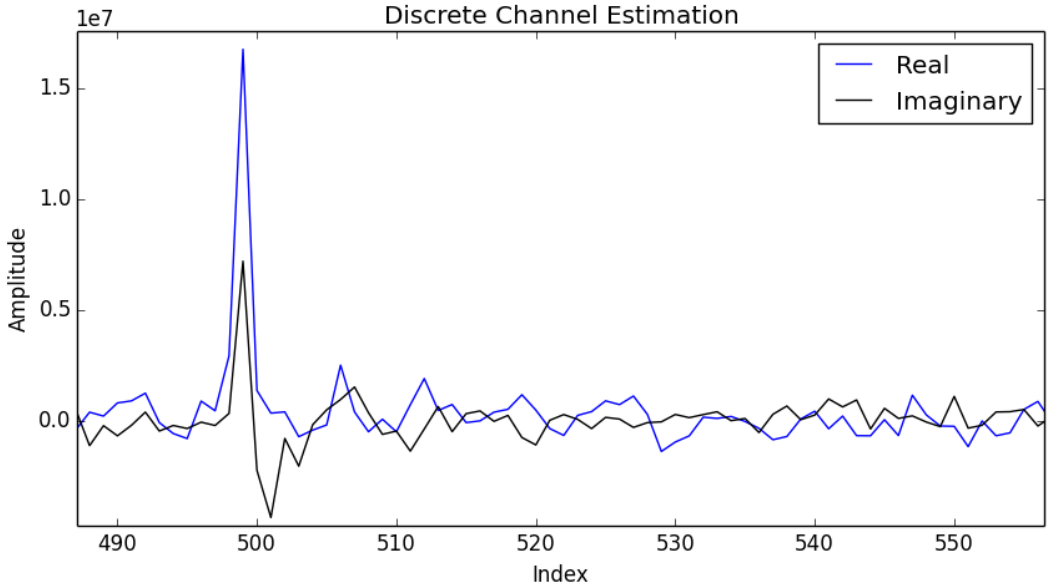
With the aligned samples, it is fairly trivial to decode the samples, multiplying by both a cosine and a minus sine wave, which we are now confident are aligned in phase. We can easily multiply the sine channel by j, and return the complex numbers to our discrete channel. We did, however, have to multiply by -1 sometimes to get the correct data since the bits somehow got reversed.

Discrete Channel Estimation

After the pseudo-random noise used to align the signal, we encoded more noise, so that we could estimate the digital channel. In our final system, we encoded 500 symbols of noise, which each included a real and imaginary component (the first half was multiplied by cosine and the second by sine). We then cross-correlated the first 500 samples with the known noise, as the random noise was again seeded, as complex numbers. We then saw a channel impulse like the one below.

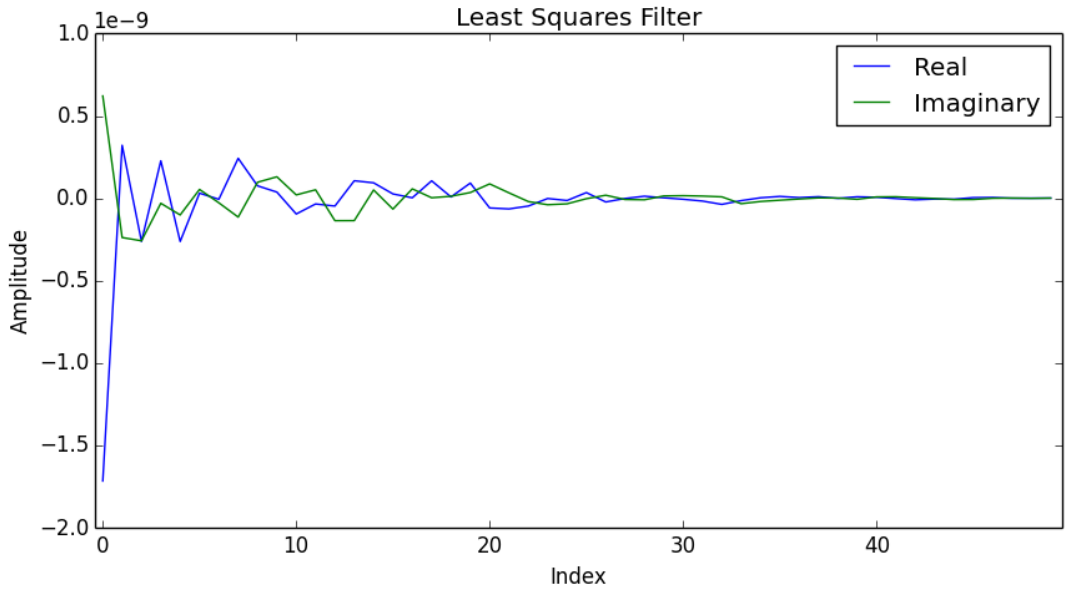


However, we didn't want to use all of the channel to equalize, since most of the channel appears to be just noise and we know the significant chanel influence occurs near the impulse (both before and afterwards, although more so afterwards). What we instead did was look at the channel near the peak. The best results were obtained with a channel that extended 5 samples backwards and 20 samples forwards in time. This could be tuned based on what the impulse response looked like, specifically how long it seemed to last. Optimizations were only approximately made, since it seemed like cheating to tune the system based on it's output, without changing the input.

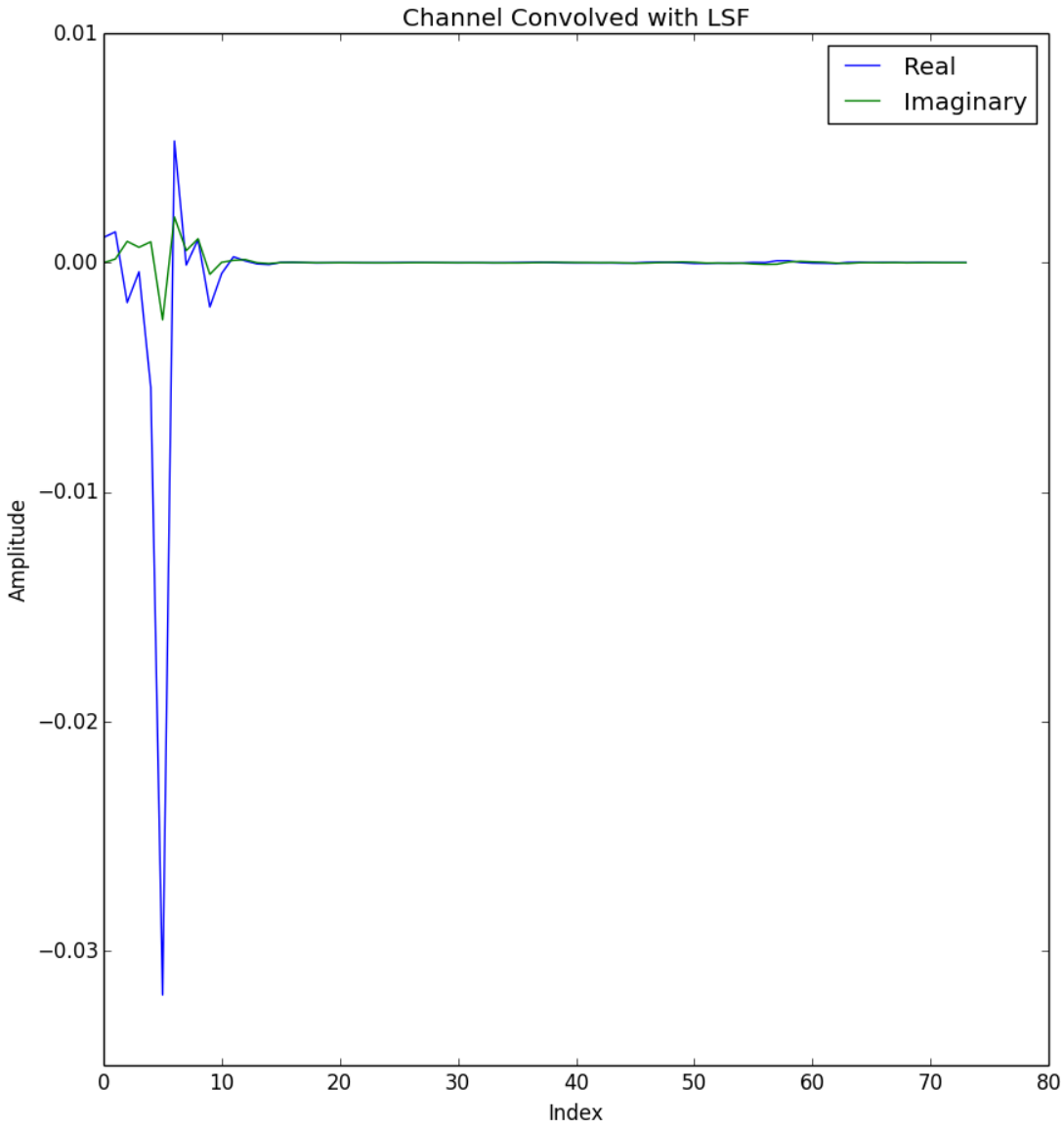


Channel Equalization

We performed channel equalization using a least squares method described in class. For the R matrix we used an impulse instead of a raised cosine because we didn't have any issues with instability. We attempted to make the result of the channel convolved with the filter equal to a delta function centered at 0. The resultant filter appears to have the correct behavior to cancel echos; it flips polarity and decreases in magnitude as time progresses.

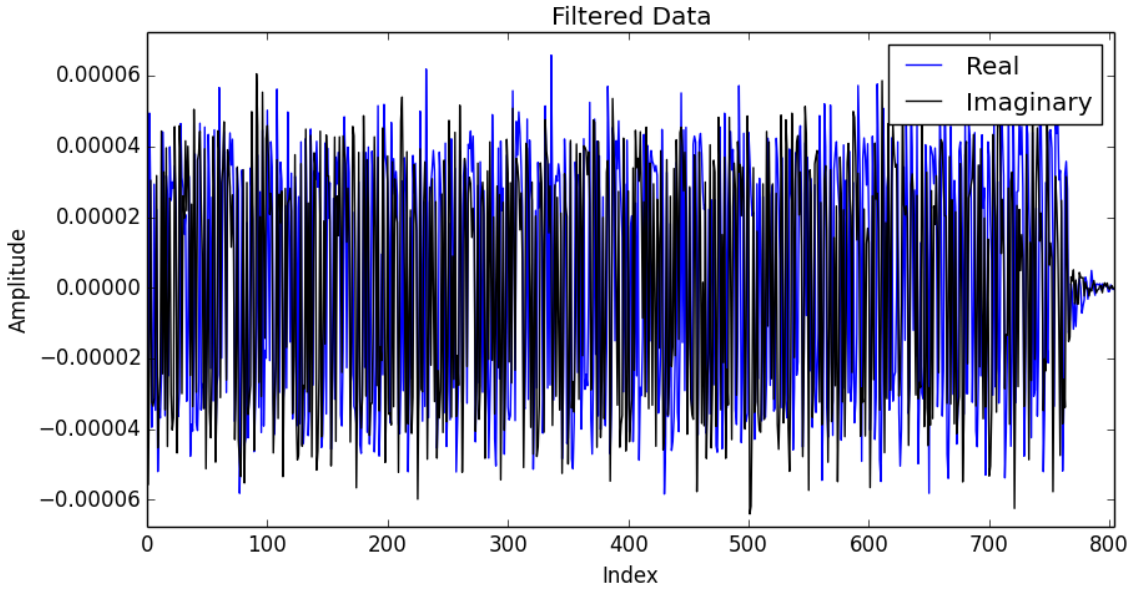


To verify that our channel equalization is performing properly, we convolved our filter with our estimated channel. This would ideally give us an exact delta. What actually happens is that there is still some ISI between consecutive symbols, and some between the real and imaginary parts of the same symbol. This is all small in magnitude relative to the peak from the symbol itself.

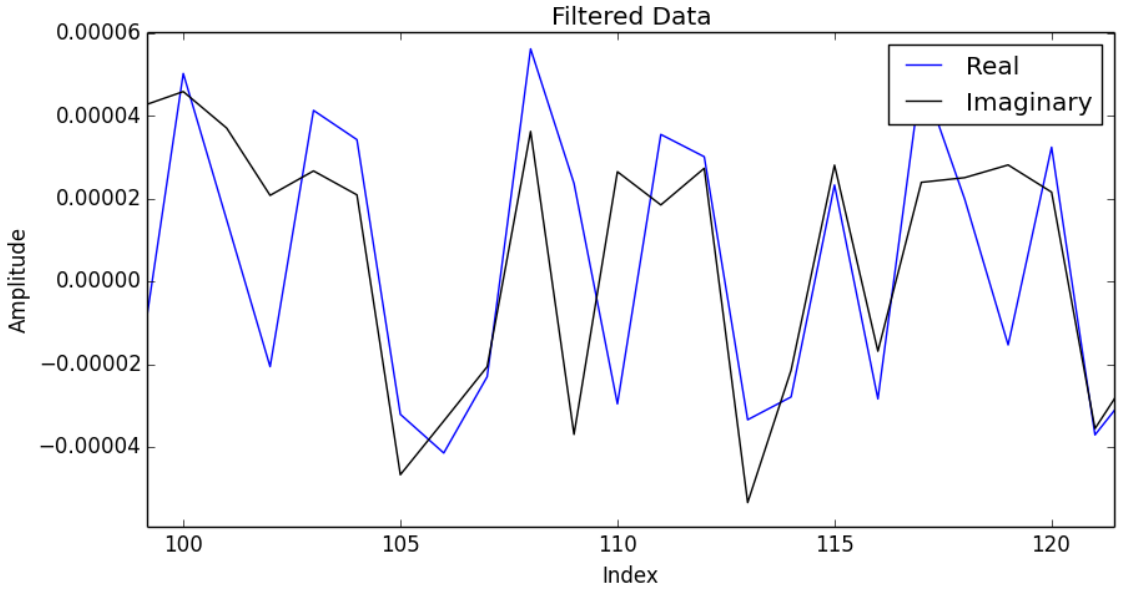


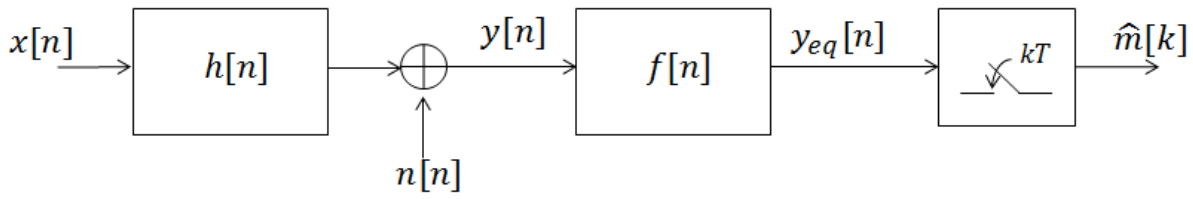
Output Data

The first figure here shows all of the output, 1552 bits in 776 symbols. It is hard to see any real features of the data at this scale, so the next figure is zoomed in in time around the index 100, which shows more of the features of the data.



The data can be seen more clearly in the zoomed in view below, which shows 20 symbols, none of which are errors. The noise in the data can be more clearly visualized, as the magnitudes that constitute a “1” or a “0” can vary significantly.

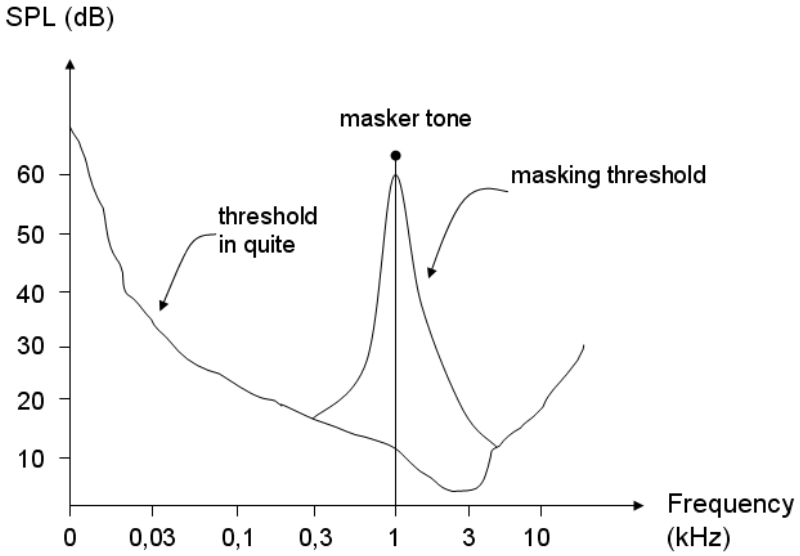




Masking encoded data

The limits of human hearing are well-modeled by people interested in compressing music efficiently, who are interested in determining the smallest amount of data that can be used to store music that is indistinguishable from the desired music. In order to compute this, they create models of human hearing to describe what qualifies two sounds as indistinguishable.

In silence, we are able to hear frequencies well between 20Hz and 20kHz. More precisely, the amplitude of sound necessary to be noticed by our ears is very large (SPL > 60dB) for tones below 20Hz and above 60Hz. Inside of this range, this amplitude (called the masking threshold) is much lower. In the presence of background sound, we become less able to perceive tones. In general, the smallest audible amplitude of a tone in the presence of a masking background tone is a function of the frequencies of the two tones. The figure below shows the masking threshold in the presence of a 1kHz tone.



For a masking threshold with multiple frequencies, the masking threshold will increase above the threshold in quiet by the sum of the increases of the individual tones. Thus, for a given non-changing spectrogram of background noise, a model of the masking threshold can be computed.

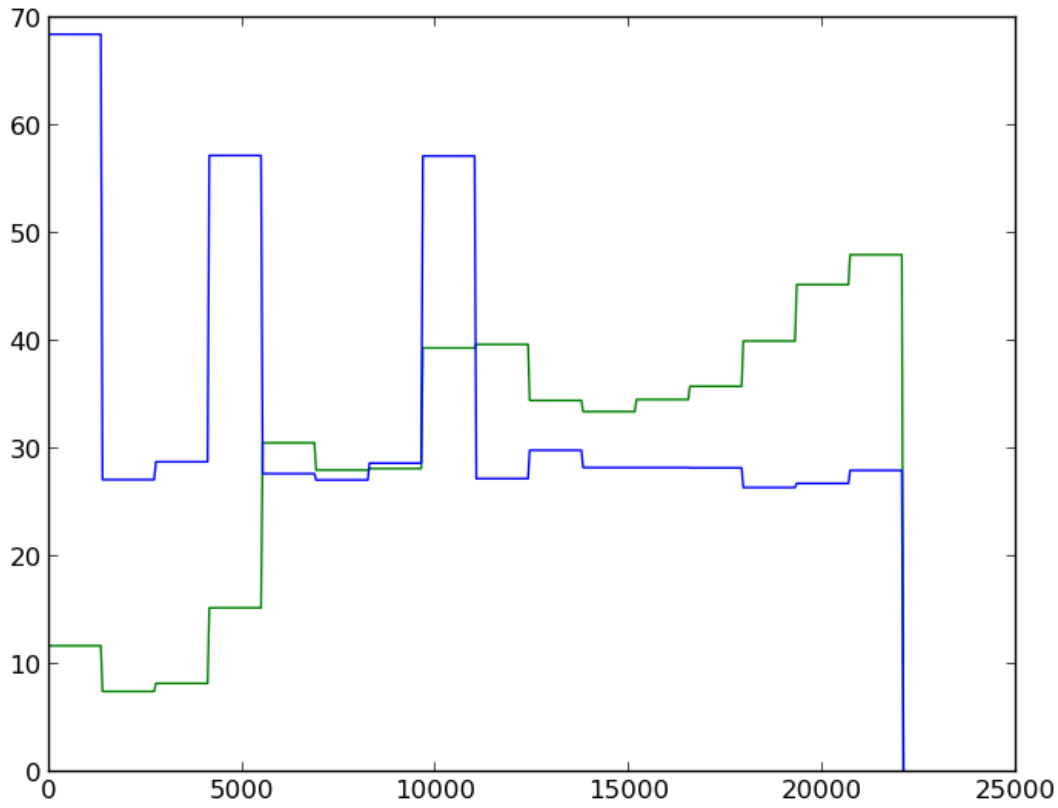
Since we were not interested in creating this model ourselves, we used open-source code from International Computer Science Institute in Berkeley which implements the masking threshold model used in MPEG1 Audio Encoding, the predecessor to the now-ubiquitous MP3 format. Since the algorithms are optimized for speed, the spectrogram and the masking threshold are calculated in frequency bins, rather than individual frequencies. We chose to use 24 frequency bins, which would eventually become our channels for sending data. Our bands were linearly spaced due to the fact that they were calculated using the fast fourier transform, and because

they would eventually be used as constant-bandwidth channels for sending data.

Because we did not want to make a modem that simply transmits data in a range that is inaudible without the mask, we chose to discard all frequencies greater than 15kHz, and less than 345Hz, since those are the least audible frequencies in quiet. While they would have been effective channels, we felt that it would be doing the project a disservice by hiding data where it wouldn't be heard anyway.

Our masking song would be changing rapidly in time, while the masking model assumes the mask has a time-constant spectrogram. Therefore, we computed the masking threshold in short samples, and, based on recommendations from the ICSI paper, we chose to calculate it using 1024 samples (0.023 seconds) of data at a time.

We used CTypes to create a python library that calls the MPEG1 C library to compute the masking threshold in the presence of samples of a song. Below is an example sample of the spectrogram of a song sample, and the computed masking threshold of that sample, divided into 24 frequency bins. The figure below shows the masking threshold for a sample of Let It Go, from the movie Frozen. The blue line shows the spectrogram of our sample, and the green line shows the masking threshold of that sample.



We observed that the masking threshold has the general appearance of a the spectrogram, convolved with a positively-skewed gaussian, scaled down and added to the mask in quiet. The mask in quiet was strongest in

Limitations

The data rate we used to achieve an error below 1% was partially limited by the quality of our speakers and microphones. We believe that a higher quality microphone and speaker would significantly increase the rate at which we could send data with the same error rate. This would probably also help the fact that in order to work well, our system requires that the laptops transmitting and receiving are quite close together and that the room is pretty quite. With better speakers and microphones, we could up the volume and the quality of the signal being transmitted so that external noise and distance wouldn't have such a negative impact on error rate.

Also, at times bits would be flipped seemingly randomly. We are not sure what would cause these flips, but that would be important to look into and minimize were we to move forward with this project.

Future Work

Currently, we have two separate systems, one that sends and receives data, and one that sends masked data. Some changes would need to be made to both to integrate them. First, the masking system implements PAM whereas the complete audio modem uses QAM. The receiver would also need to do the same masking calculations on the song we use to mask so that we know where to pick up data. We would also need to know what frequencies the data is being sent at so we decode at the right frequencies in the receiver.

To make the system even more elegant, we could use parts of the song to characterize the channel instead of sending white noise.