Developing an ODFM-OOK Communication System

ECE 4670, Spring 2022

Michelle Davies (mdd94)

ECE B.Sc, Class of 2022

May 9th, 2022

Michelle Davies (mdd94)
ECE 4670 (Lab 4 Report)
May 9, 2022

Lab 4: Report

Building a Digital Communication System

For the Culminating Design Experience portion of this course (i.e. Lab 4 of ECE 4670), I have built my own communication system that transmits the signal via. CommCloud software. This document contains: (1) the standard of this communication design, (2) principles of design for the encoder and decoder, and (3) a quantitative analysis of the performance of this lab.

This system encodes a signal of length N_b bits into N_{enc} symbols (sample length N_{sample}) encoded via Orthogonal frequency-division multiplexing ("ODFM") combined with On-Off Keying ("OOK").

ODFM is a method of digital transmission of data which uses "eigenvectors instead of singular vectors to pre- and post-distort". (Lab 4 handout) For the purposes of this system, this definition implies that we can define a set of known parameters for signal measurements that hold true for the signal being transmitted at all stages of this system. This is because ODFM pre- and post-distorts a signal using eigenvectors, which remain the same for a circulant matrix and therefore do not vary with the impulse response of the system. Due to this property, we do not have to account for fluctuations associated with the impulse response; all other parameter values and their dimensions can be determined by design.

The role of OOK in this communication system is designing the encoding process of the system such that the bits are grouped and then translated into symbols that either equate to 0 or a complex-valued symbol. I have chosen to constrain the magnitude A of the symbols transmitted in this system to remain constant as the phase of each symbol is calculated. Symbols which correspond with a low Signal to Noise Ratio ("SNR"), i.e. symbols which contain more noise than it does desired signal information, will be optimized in order to conserve transmission power as appropriate.

Communication Standard: ODFM-OOK System

The following diagram illustrates the encoding process of this system:

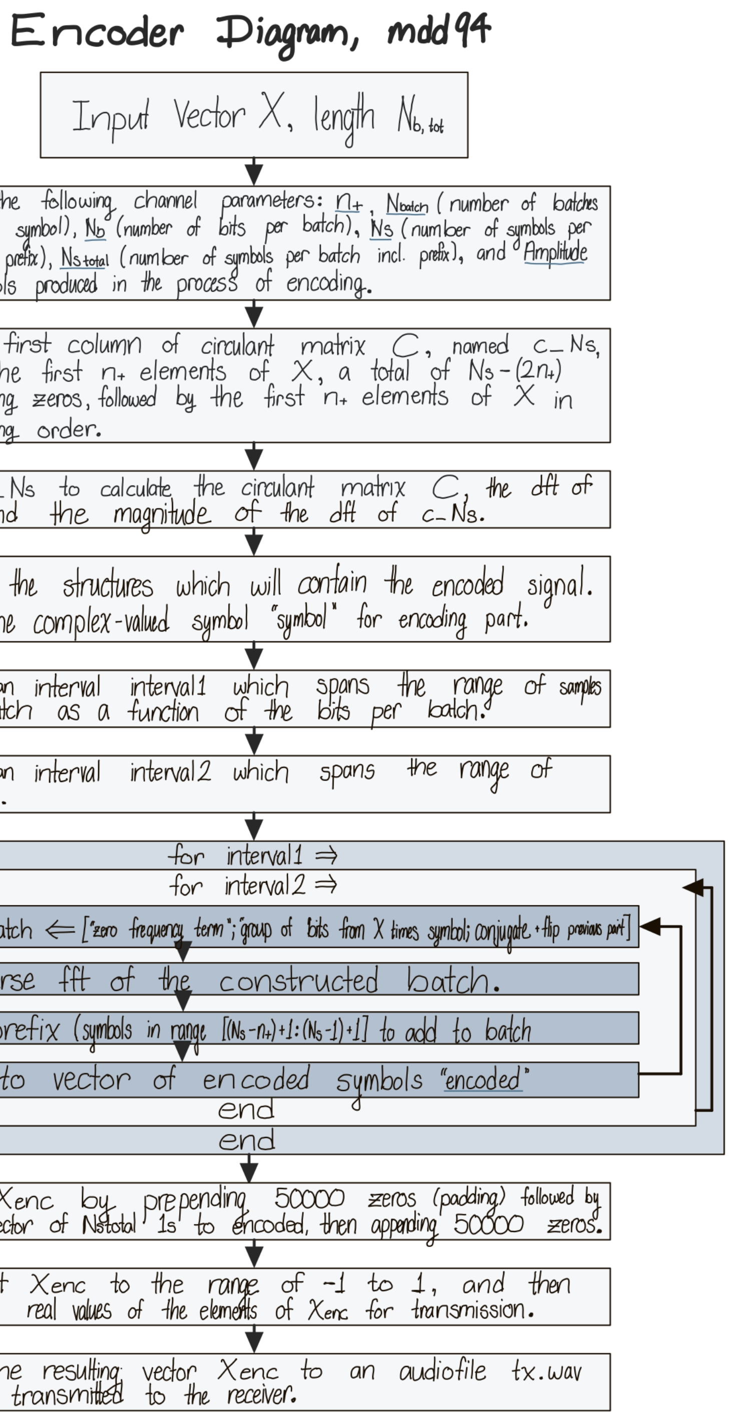
Encoder Diagram, mdd94
Input Vector X, length $N_{b,tot}$
Define the following channel parameters: n_+, N_{batch} (number of batches per OFDM symbol), N_b (number of bits per batch), N_s (number of symbols per batch, excl. prefix), $N_{s\,total}$ (number of symbols per batch incl. prefix), and Amplitude of symbols produced in the process of encoding.
Get the first column of circulant matrix C, named c_Ns, using the first n_+ elements of X, a total of $N_s-(2n_+)$ intervening zeros, followed by the first n_+ elements of X in descending order.
Use c_Ns to calculate the circulant matrix C, the dft of c_Ns, and the magnitude of the dft of c_Ns.
Initialize the structures which will contain the encoded signal. Also define complex-valued symbol "symbol" for encoding part.
Define an interval interval1 which spans the range of samples per batch as a function of the bits per batch.
Define an interval interval2 which spans the range of Nbatch.
for interval1 ⇒
for interval2 ⇒
construct the batch ⇐ ["zero frequency term"; "group of bits from X times symbol; conjugate + flip previous part]
Take the inverse fft of the constructed batch.
Calculate prefix (symbols in range $[(N_s-n_+)+1:(N_s-1)+1]$ to add to batch
Add batch to vector of encoded symbols "encoded"
end
end
Build Xenc by prepending 50000 zeros (padding) followed by a pilot vector of Nstotal 1s to encoded, then appending 50000 zeros.
Constrict Xenc to the range of -1 to 1, and then take the real values of the elements of Xenc for transmission.
Write the resulting vector Xenc to an audiofile tx.wav to be transmitted to the receiver.

Figure 1: An illustration of the process of encoding an input signal using ODFM and OOK methodologies, which contains $N_{b,tot}$ bits, into a signal containing N_{enc} symbols comprising a vector of sample length N_{sample}.

As the above diagram illustrated, an input signal *X* is transmitted by first learning the channel by finding the first column of the circulant matrix for this system using *X,* and setting the channel parameters for measuring the significant points in the signal *X*. Once this has been done, we now prepare the system to do the encoding. Part of this preparation is defining the symbol that will represent these encoded bits using the following form:

$$S = Amplitude * e^{i*\phi_X} = Amplitude * e^{i*(2\pi*N_b)}$$

Now, the system iterates through all of the bits in *X*, and repeats the following steps to assign and encode the bits to generate a transmittable signal $X_{symbols}$:

1. The batch is constructed. To construct the batch, there are 3 main features that are put together to comprise this vector: (a) the "zero frequency term," where the first symbol of the main part of the batch is set to 0; (b) the element-wise multiplication of the group of N_b bits to be encoded in a batch with the symbol *S;* and (c) the flipped conjugate of (b).
2. Take the inverse FFT of the batch that was constructed in the previous step.
3. Calculate the prefix of the batch by taking the symbols from the batch that fall within the positions in the range of $(N_s+n_+)+1$ to $(N_s-1)+1$, and then prepend this prefix to the batch.
4. Add the components of resulting batch to the full vector of encoded bits.

Once the above steps have been completed, we have the fully encoded vector of the signal *X*. Finally, we prepare the encoded signal for transmission by prepending a pilot vector of N_s 1s to the encoded signal, and then padding this pilot+encoded signal vector with 50000 zeros at the beginning and 50000 zeros at the end. The result is a vector that is ready to be converted to an audio (.wav) file to be transmitted from the encoder function.

This design assumes the following facts to hold true throughout the transmission process:

(1) The number of information symbols to be transmitted in one large "OFDM symbol" can be predetermined and set as a reliable constant in spite of any fluctuations that occur in transmission.

(2) We can predict or rather predetermine the distortion that will occur in the signal's prefix and postfix, and assume that those settings are consistent.

Design Principles

Encoder Design

In the encoder, the main design choice that I made for this part of the system was to encode the bits of the system all in one go; that is to say, I opted to construct all of the signal's batches together in one set of nested loops and then and the padding and pilot signal after the fact but prior to transmission, rather than creating and sending the batches one by one.

My reasoning for making this design choice was because I wanted to observe the impact of minimizing and normalizing the impact of channel fluctuation across different symbols due to oddities associated with CommCloud tendency of truncating symbols as much as possible by making sure that the data was plentiful; I was concerned that transmitting only one bit at a time on CommCloud would worsen the impact of sending short signals via CommCloud on each individual batch, even after using padding to lessen the effects.

The cost of making this design choice may have contributed to the oddities in the length of the decoded signal being a complete mismatch to that of the signal inputted into the encoder. Due to the fact that any unavoidable truncation due to CommCloud is happening non-uniformly and it's unpredictable, it is that much more challenging to account for changes in the spacing between th symbols and the batches.

Decoder Design

The outline of my decoder's design is as follows:

1. Recall the known parameters set as system properties as before.
2. Calculate the value of the symbols used to encode the original vector using some of the known parameters.
3. Grab the received signal's audio file and write the data to a vector.
4. Calculate the number of samples per OFDM symbol.
5. Using the value from step 4 and the length of the received signal data vector from step 3, obtain the number of OFDM symbols to iterate through.
6. For each OFDM symbol, repeat the following steps until we have gone through all OFDM symbols:
    1. Use a defined power threshold to determine where the start of a symbol is located.
        1. If the value of this position is infinite, remove symbols with a low signal to noise ratio.

    2. Define (predictive) beginning and ending positions for the region of the OFDM symbol.
    3. Decode the OFDM symbol:
        1. Use a defined power threshold to determine the region to be decoded.
        2. Delete the batch's prefix.
        3. Get the complex component of the symbols.
        4. For each batch:
            1. Define region of signal to avoid operating on (prefix).
            2. Delete this region.
            3. Use On Off Keying to decode the remaining, desired portion of the signal.
            4. Place the result of step 6.3.4.3 in the decoder's output
    4. Add the decoded bits to the output vector of the decoder function;
7. Remove any remaining symbols which may be associated with a low signal to noise ratio.
8. Return the resulting decoded signal of bits.

In the decoder design, the key design decisions that I made here were to (1) capture the output of the transmission right in the decoder function, (2) using a power threshold to locate the symbols in the received signal, (3) deviating from the encoder's design by decoding the whole signal step by step, symbol by symbol, and finally, (4) electing to forego using the symbols associated with a low signal to noise ratio. All of these decisions were ultimately made in order to allow me to attempt to optimize the performance metrics for my system, but each of these decisions takes a different group of metrics into consideration.

For instance, electing to forego using the symbols associated with a low signal to noise ratio was an important design choice because it helps to improve performance by allowing the system to conserve power that it would otherwise expend on less significant symbols that do not impact that data being returned with the same urgency.

Deviating from the encoder's design by decoding the whole signal step by step, symbol by symbol, was a choice made in order to ensurer a lesser chance of bit errors by applying the decoding to smaller groups at a time.

Using a power threshold to locate the symbols in the received signal made sense because we can assume that regions of lower power (below threshold) precede regions where more power (above threshold) is required for the symbol, so power levels make for an ideal indicator of where to start defining regions in the signal to decode.

Finally, the reason that I want to capture the output of the transmission right in the decoder function is so that the decoders call limit the number of retransmissions that occur for this lab.

Results: Performance Analysis of the System

Regrettably, I ended up with a communication system that successfully performs the mechanism of the individual steps of building and deconstructing a decoded symbol as well as attempts to improve, but doesn't seem to return the same number of bits as the system receives at the encoder, regardless of the length of the input vector and regardless of the number of total iterations run for this system. I.e., my signal is off in terms of length by at least a couple thousand bits every time. After speaking to and reviewing this system with Professor Doerschuk taking the time constraint and limitations of a one-person group into account, I have resolved to analyze the system that I have as is with the metrics I have obtained from testing my code, and then predict the performance metric trends based off of what I have.

```
>> lab4_monte_carlo
```

```
% encoder
encoded size
    430200        1
Size of X (-> tx.wav)
    532351        1
```

```
% decoder
# samplespersym
     15057
length(Y_bar)
    524283
# ODFM Symbols
   35
```

Number of runs	nextODFMsample position	Size of Y_bar	beginning of range	ending of range	Current symbol that program points to:
1	1	535206	1	15057	1
2	12916	535206	12916	27972	2
3	25831	535206	25831	40887	3
4	38746	535206	38746	53802	4
5	51661	535206	51661	66717	5
6	64576	535206	64576	79632	6
7	77491	535206	77491	92547	7
8	90406	535206	90406	105462	8
9	103321	535206	103321	118377	9
10	116236	535206	116236	131292	10
11	129151	535206	129151	144207	11

Number of runs	nextODFMsample position	Size of Y_bar	beginning of range	ending of range	Current symbol that program points to:
12	142066	535206	142066	157122	12
13	154981	535206	154981	170037	13
14	167896	535206	167896	182952	14
15	180811	535206	180811	195867	15
…			…		
36	452026	535206	452026	467082	36

Figure 2: Code output and table(s) depicting the results obtained by this system for running using Monte Carlo for tot=10 iterations.

Data Rate Metric

I was able to get a data rate metric in spite of the mismatching in dimensions.

For the value of R_0 (the data rate in terms of units of bits/sample), the average data rate obtained by the system was 0.375 bits/sample.

For the value of R_1 (the data rate in terms of units of bits/second), the average data rate obtained by the system was 16568 bits/second.

Evaluating R_1, I can see that the the data rate for the system was a reasonable, but not most optimal for the constraints of the system designed.

Bit Error Probability Metric

Computing this metric was a bit difficult for me due to the challenges that I described earlier in this section. After much deliberation and testing, I have noticed that the lowest error I have managed to obtain in this setup was 0.54 due to this incorrect random offset that I can't seem to get rid of, so I have opted to use the best scenario and the worst case value I got was 99998, and obtain a range of possible ranges for the Overall Performance Metric, just for the sake of being able to get a value of an Overall Performance Metric to analyze so that I have a value to work with.

Transmitter Power Metric

The average power metric obtained for this system reached a value of 0.812154. There seems to be a higher transmission power metric for my lab4 setup than I observed for my lab3 setup, which suggests that my signal is traveling to the decoder with a heightened strength. This is not unexpected for ODFM systems.

Overall Performance Metric

The value that I obtained for this metric using the best-case error value of 0.54 was M = 0.000578.

The value that I obtained for this metric using the worst-case error value of 99998 was M = ~0. While neither of these results are most optimal for ODFM, I started with an error rate with ~101250 wrong bits per run, so this is an improvement if you consider that perspective. Still, my results are unfortunately unstable. I was told that whatever is wrong is not very obvious based on the encoder/decoder logic, so that is where I left off.

Conclusion & Final Remarks

Overall, in the process of developing this OFDM communication system, a valuable lesson learned was about the harsh realities of communication systems design. Although a system may be well designed in theory in terms of its encoder and its decoder, there is an aspect of uncertainty in how well that system will perform in practice for different iterations of a system run. With this system, I often found that certain processes that I would expect to work in theory, would not work for my system because the actual measurements (e.g. measurements between my OFDM symbols) were off sometimes despite my best efforts to predict the locations of the symbols, areas with a low Signal to Noise Ratio, and the prefixes of the symbol's batches. Regardless, I ended up with a reasonable system in terms of the mechanics of the transmission process and theoretical performance.