EE2025 Independent Project (2019-20)
Programming Assignment - 2

See Google Classroom for submission deadline

Submission Instructions: You may form teams of size 1-2 students (only among students crediting the course). Exactly one of the team members must upload the simulation result online in Google Classrooms. You must upload the following, as two separate files (do not 'zip' them):

1. simulation results in '.pdf' format. Other formats are not allowed. This file must include the details of all the team members (name and roll number).
2. your program/script file(s).

Cheating, in any form, will be taken seriously. Please write the program on your own. You could be asked to present the script file and explain the algorithm orally at short notice.

Programming Language: You can use MATLAB, Python or any other tool for this programming assignment.

Objectives: In the previous assignment, you implemented a QAM modulator and demodulator and simulated the transmission of an image of Mona Lisa through an additive white Gaussian noise channel. In this assignment, we will improve the bit error rate (fraction of bits incorrectly decoded) using channel coding. We will see three different channel coding schemes, and the tradeoff between the transmission rate and the probability of error.

We will work with a larger image, which can be found as an attachment. Use the following code to load the image:

Python:
```
import numpy as np
from matplotlib import pyplot as plt
MSS = np.load('mss.npy')
plt.imshow(MSS,'gray'), plt.show()
```

MATLAB:
```
load mss.mat
imshow(mss, [0 1])
```

The image, in all, contains $400 \times 300 = 120000$ information bits.

Implementation: You will encode, modulate and transmit the image using 4-QAM modulation scheme that you implemented in Assignment 1 and the channel codes described below. Use the same parameters for T, f_c, f_s as earlier.

First, simulate the communication for the following values of the noise variance in the discrete-time model: $\sigma^2 = 20, 12, 7, 5$. This is an unfair comparison, since the total energy used to transmit the image is different for each code (because of the difference in rates). Next, simulate the communication for 5 values of $\mathcal{E}_b/N_o$: $-2, 0, 2, 4, 6$ dB. Since $\mathcal{E}_b$ is the energy *per message bit,* the corresponding noise variance depends on the rate of the code.

For each of these scenarios, you will plot the received image (which will be noisy) and report the number of pixels that were wrongly decoded.

Also plot the bit error rate, or BER (fraction of pixels that were wrongly decoded) versus E_b/N_0. For plotting the BER, use a logarithmic y-axis, i.e., use `plt.semilogy` instead of `plt.plot` in Python and `semilogy` instead of `plot` in MATLAB.

These four figures, the corresponding number of wrong pixels, and the plot must be reported in a single pdf file. Attach your code in a separate .py/.m/.ipynb file.

Blocks in the digital communication system

The input to the digital communication system is a binary image (a matrix) of size 400×300. This can be equivalently represented as a $n_m = 120000$ length binary vector $m_1, \ldots, m_{n_m}$. The transmitter has two blocks:

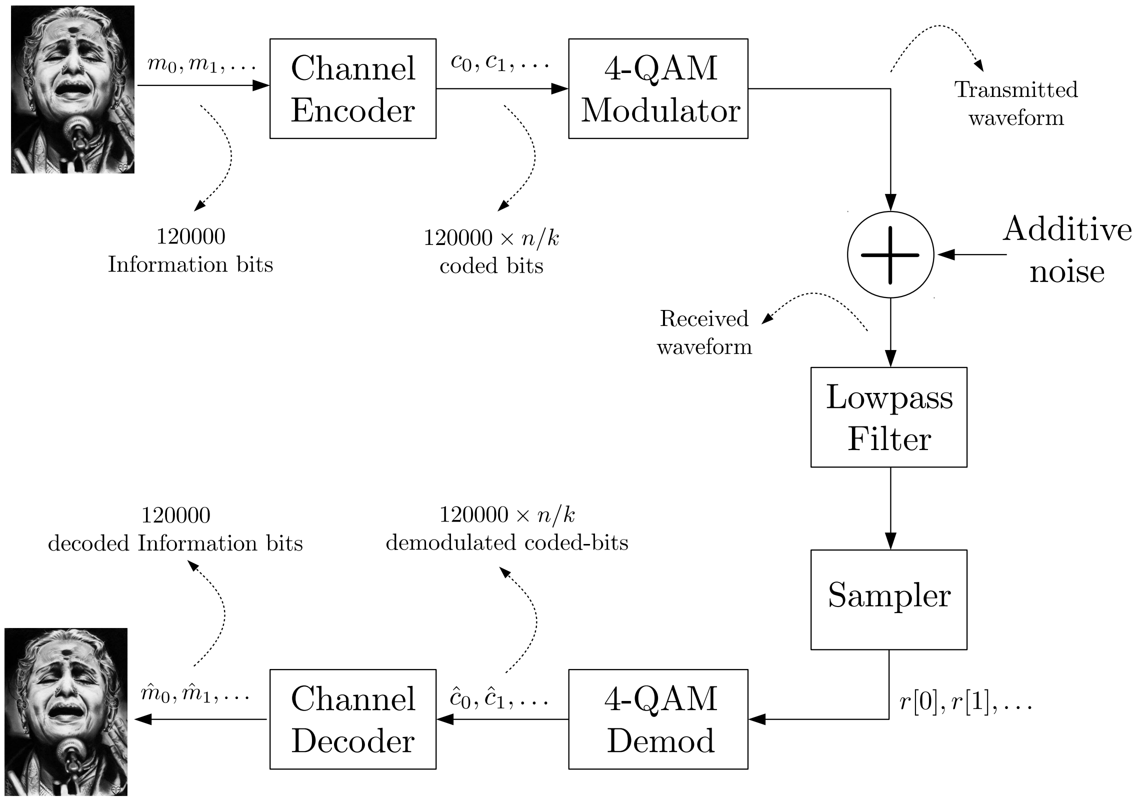

Figure 1: Block diagram of the digital communication system. We communicate an image file using an error correcting code and 4-QAM modulation scheme.

1. Channel encoder: This block takes k information bits at a time, and outputs n codeword bits. The codeword bits are all concatenated and sent to the next stage. For $i = 0, 1, \ldots, n_m/k - 1$, the encoder takes $m_{ik}, \ldots, m_{(i+1)k-1}$ and outputs $c_{in}, \ldots, c_{(i+1)n-1}$. We will describe this block in more detail later.

   The output of this block will be a binary vector of length $n_{ch} = n_m \times n/k = 120000 \times n/k$. Call this $c_0, \ldots, c_{n_{ch}-1}$.

2. Modulator: This block takes 2 codeword bits at a time, c_{2i}, c_{2i+1} and outputs a continuous signal of duration T. In the discrete-time model, the modulator takes c_{2i}, c_{2i+1} and outputs a vector of length $Tf_s = 50$ (in our case) with real-valued entries. Use the 4-QAM modulator from Assignment 1.

   The output of this block will be a real-valued vector of length $Tf_s \times n_m \times n/(2k) = 25 \times 120000 \times n/k$.

The transmitted signal is corrupted by additive white Gaussian noise with the appropriate variance, as in the previous assignment. It is important to note that $\mathcal{E}_b/N_0$ depends on the rate of the channel code. In particular, $\mathcal{E}_b$ is equal to $n/k\times$ energy of one coded binary symbol. Hence, you will need to recompute the noise variance for each channel code.

The decoder has two stages:

1. Demodulator: In the discrete-time channel model, the demodulator takes Tf_s samples at a time and outputs 2 codeword bits. All the codeword bits are concatenated and sent to the next stage. Use the demodulator from Assignment 1.

   The output of this block will be a binary vector of length $n_{ch} = 120000 \times n/k$. Call this $\hat{c}_0, \ldots, \hat{c}_{n_{ch}}$.

2. Channel decoder: This takes n demodulated bits at a time, say $\hat{c}_{in}, \ldots, \hat{c}_{(i+1)n-1}$ and decodes each to k message/information bits $\hat{m}_{ik}, \ldots, \hat{m}_{(i+1)k-1}$. We describe this in more detail later.

   The output of this block should be the decoded image, which is a binary matrix of size 400×300.

Channel coding and decoding

Recall that a channel encoder f is a function that takes k information/message bits as input, and outputs a codeword of n bits. The decoder g takes n received bits as input, and outputs k bits that correspond to an estimate of the message.

We will use three different channel codes:

1. A rate 1/2 linear code with $n = 8$ and $k = 4$. The relationship between the k information bit vector m^k and n-length codeword vector x^n is given by

$$x^n = \left[G_1^T m^k\right] \bmod 2$$

   where G_1^T denotes transpose of matrix G_1 and

$$G_1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

   The decoder is the standard minimum Hamming distance decoder: Given a received vector $y^n \in \{0,1\}^n$, the decoder outputs

$$\hat{m}^k = \arg \min_{m^k \in \{0,1\}^k} d_H(G_1^T m^k, y^n),$$

   where d_H is the Hamming distance between two vectors (i.e., the number of locations where the two vectors differ). For example, if $a = (1,1,1)$, $b = (0,1,1)$ and $c = (1,0,0)$, then $d_H(a,b) = 1$ while $d_H(b,c) = 3$.

2. A rate 1/3 repetition code: The encoder takes 1 bit m as input and outputs 3 bits (m,m,m), i.e., repeats the message bit 3 times.

   The decoder simply outputs the majority. Given $y^3 = (y_1, y_2, y_3)$, it outputs $\hat{m} = 0$ if there are at least 2 zeros in y^3, and outputs 1 otherwise.

3. A rate 1/3 linear code with $n = 12$ and $k = 4$: The encoder and decoder operate in a fashion similar to code 1, but instead of G_1 we take

$$G_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

   The decoder is the minimum Hamming distance decoder for this code.

Remarks

- Channel encoding and decoding is a computationally intensive process. The entire simulation should take about $25 - 30$ minutes on a modern processor (like a core i3-i5). In particular, minimum Hamming distance decoding takes the most resources since you have to compute 2^k distances every time you want to decode.

- In practice (e.g., cellular communication), we do not use the minimum distance decoder but instead carefully designed low-complexity decoders. It is remarkable how well modern codes perform: even with k in the thousands, our cell phones equipped with less powerful processors perform encoding and decoding quickly, and the bit error rates are far better than what you obtain here for the same rate and E_b/N_0.

  To give some perspective, codes with $n = 1248$, $k = 624$, and achieving BER 10^{-5} for E_b/N_0 of 2.5dB with decoding time less than a millisecond are being used in practice today.

- When testing and debugging your code, you can instead use the Mona Lisa image from the previous assignment, or generate 10^4 random message bits to improve speed.

- For the image provided in this assignment, when you plot the BER for different noise variances, you should observe that the bit error rate (BER) for code 1 is lower than uncoded transmission. This code has rate 1/2. The rate 1/3 repetition code further reduces the BER at the cost of lower transmission rate. Code 3 has even lower BER, but at the same rate 1/3. However, the price that we pay is increased complexity and delay.

  You will notice that the ordering is different when you simulate for different $\mathcal{E}_b/N_0$.