CT111: Introduction to Communication Systems

End-of-Semester Project

In the end-of-semester project, you will develop the following:

1. Message Passing Channel Decoder for LDPC Codes,
2. A basic Modem (MOdulator and DEModulator), and
3. Viterbi Decoder for Convolutional Codes.

Following are several logistical notes about the project work:

- ▷ You will submit intermediate updates (e.g., a working Matlab code) of Item 1 above. You will proceed with Item 2 only if the TAs approve that you have successfully completed Item 1. Similarly, you will proceed with Item 3 only if the TAs approve that you have completed Item 2.
- ▷ At the end of the semester, e.g., *either* before or *after* your final exams are over, you will make a presentation of your project work.
  - → Those of you who do not want to be burdened with completing the project and preparing a presentation in the midst of the preparations for the final exam may decide to take the viva of their project work after the conclusion of the final exams.
  - → Students who wish to be relieved of the study-related commitments no later than the last day of the final exam (e.g., because they have scheduled their travel accordingly) can opt to take the viva earlier.
  - → Please coordinate with the TAs regarding the date that you will choose to take the viva.
- ▷ For the project, you are free to make your own groups of size about 10 to 12 provided each group belongs to the same lab group of approximately 60 students.

Provided that you are able to attempt Item 3, you will have covered all the blocks that make up the heart and the main logic that empowers (what is known as the "physical layer" of) the digital communication system whose diagram is shown in Fig. 1.

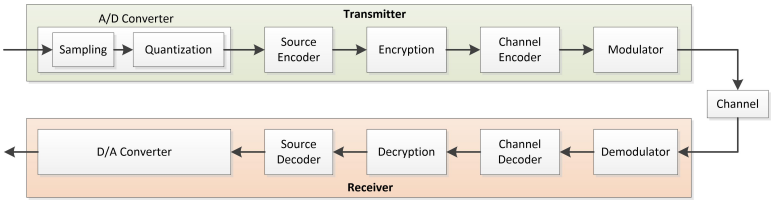

Figure 1: Block diagram of a digital communication system.

Instructions for Part 1 of the Project

1. In Part 1, you should first implement the decoder of a basic $(N = 9, K = 4)$ Product Code using message passing on the Tanner Graph that we have studied in the class.

2. You should first build an encoder that takes any (of 16 possible) binary string of length $K = 4$ bits and converts it into a product code of length $N = 9$ bits.

   ▷ Encoder can be implemented by generating $N - K = 5$ parity bits using the rectangular array formulation that we have studied in the class. Here, you take a randomly generated block of $K = 4$ bits, convert that into a 2×2 square array, and now generate 5 parity bits by deriving 5 SPC bits on each of the two rows and two columns of this 2×2 array. Refer to the lecture notes for a description of this.

3. Next build a decoder that implements the message passing on Tanner Graph.

   ▷ This is the main challenge of this part of the project. Here, you are asked to take the parity check matrix corresponding to $(9, 4)$ product code and convert the pattern of ones and zeros in this parity check matrix $\mathbf{H}$ into a Tanner Graph model in Matlab. The connections between the bit nodes and the check nodes of this bipartite graph are based on the locations of ones in $\mathbf{H}$.

   ▷ It is important that you make your code generic enough so that it does not fail to work with another matrix $\mathbf{H}$.

4. Your Matlab model of Tanner Graph should perform the message passing from bit nodes to variable nodes and vice versa.

   ▷ Recall that in the bipartite Tanner Graph, each bit node is connected to a total of d_v check nodes, and each check node is connected to a total of d_c bit nodes.

   ▷ Message passing for the BEC channel:

   → From the Check Node to Bit Node:

   * a check node sends message to each of d_c bit nodes connected to it. When sending the message to k^{th} bit node, the check node "listens" to the messages received from $\{1, 2, \ldots, k-1, k+1, \ldots, d_c\}$ bit nodes.
   * the message passed to k^{th} bit node is the correct value of that bit node provided *none* of $d_c - 1$ incoming message to this check node are erasures. Else the check node passes an erasure message to k^{th} bit node.

   → From the Bit Node to Check Node:

* a bit node sends message to each of d_v check nodes connected to it. When sending the message to k^{th} check node, the bit node "listens" to the messages received from $\{1, 2, \ldots, k-1, k+1, \ldots, d_v\}$ check nodes that it is connected to, and also the bit received over the BEC.
* the message passed is the correct value of the bit node provided either the bit received over the BEC channel is not an erasure or at least one of the messages received from $d_v - 1$ check nodes connected to this bit node is not an erasure. Only when (i) the bit received over the BEC is an erasure *and* (ii) all $d_v - 1$ check nodes also send erasures to this bit node, this bit node sends erasure to the k^{th} check node.

▷ Message Passing for the BSC Channel:

→ You may optionally implement the decoder for the BSC channel.

→ From the Check Node to Bit Node:

* a check node sends message to each of d_c bit nodes connected to it. When sending the message to k^{th} bit node, the check node "listens" to the messages received from $\{1, 2, \ldots, k-1, k+1, \ldots, d_c\}$ bit nodes.
* the message passed by the check node to k^{th} bit node is sum modulo two of $d_c - 1$ incoming message to this check node. Thus, each check node gives k^{th} bit node connected to it the following message: "I see the values that your $d_c - 1$ neighbor bit nodes have. I, the parity check enforcer, am telling you that your value must be 0 since I am seeing an even number of 1's in your $d_c - 1$ neighbor bit nodes (or that your value must be 1 since I see an odd number of 1's in your $d_c - 1$ neighbors).

→ From the Bit Node to Check Node:

* a bit node sends message to each of d_v check nodes connected to it. When sending the message to k^{th} check node, the bit node "listens" to the messages received from $\{1, 2, \ldots, k-1, k+1, \ldots, d_v\}$ check nodes that it is connected to, and also the bit received over the BEC.
* the message passed by the bit node to k^{th} check node connected to it is the majority-vote based. The bit node sends to k^{th} check node a value of 1 if majority out of $d_v - 1$ check nodes and the bit received over the BSC itself is 1; else it sends 0 to k^{th} check node.

5. To show that your message passing decoder is working correctly, store the number of erasures as a function of the iteration index (one iteration of the message passing is said to have been completed when the messages complete one cycle, i.e., go from from bit nodes to check nodes, and then check nodes to bit nodes). Plot this number as a function of iteration index and see that this plot is converging to zero as the number of iterations increase.

6. Keep some maximum number of iterations (say, $N_{iter} = 25$). If the erasures remain at iteration index N_{iter}, set a `decodingSuccess` variable (that records whether the decoding has succeeded or not) to 1 to record that the decoding has failed, else set this variable to 0 to record that the decoding has succeeded.

7. To test whether your code is working or not for $(9, 4)$ Product Code, perform many Monte-Carlo simulation experiments. In each experiment, randomly *erase* some bits. What is the maximum number of erasures that your decoder is able to correct (i.e., when `decodingSuccess`

is 1 for all of the simulation trials)? Collect this data coming out of your simulator and plot this data.

8. Repeat your simulation exercise for larger sized product codes, e.g., $(25, 16)$, $(36, 25)$, etc. product codes. You should observe an improved decoding performance as the size of the product code increases.

9. A larger sized $\mathbf{H}$ matrix will be placed in the project/lab subfolder under lecture folder. You should get your Matlab code working for this larger sized $\mathbf{H}$.