PROJECT DESCRIPTION: ITERATIVE DECODING AND THE MAP RECEIVER

FREDRIK RUSEK, ETTN01

1. Introduction

Our intentions with this project are (i) to give you practical experience in Matlab, (ii) let you practice how to perform and document a set of simulations of a communication system, (iii) provide a case where the MAP receiver is used, (iv) give you some training with error correcting codes.

The system model for creating the received signal is shown in Figure 1 Your task is to

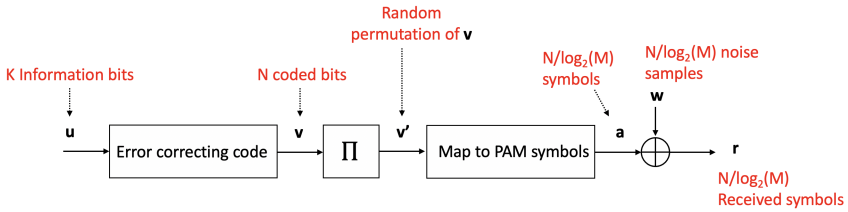

Figure 1. System model of received signal.

implement the receiver for the system, but before we discuss the receiver, let us discuss the system model in detail.

2. Transmitter

In this section we go through the blocks shown in Figure 1.

The first step is to generate a sequence of K random bits, (0/1). The value of K is in principle arbitrary, but we shall return in Section 6 later specific values. The K information bits are encoded by an error correcting code, which outputs N bits, where $N > K$. Although we have not studied error correcting codes, codes are used in all modern communication systems, and it is therefore important to be able to use them anyway. Here at LTH we have the course *Coding Theory* for those of you who are interested.

An error correcting code can be said to protect the information bits, at the expense of lower data rate. This is possible since not all sequences $\mathbf{v}$ are valid sequences. If the receiver encounters a signal which it knows could not have been transmitted, then the receiver can correct errors. This is very much the same as reading a normal text. A text has redundancy in the sense that it contains more letters than needed (c.f., $N > K$).

Now consider printing errors at the printing office (c.f., noise $N(t)$). If the printing office randomly would alter 25% of the letters, you would be able to read the text anyway. The reason is that there is redundancy in the text, i.e., there are more words than necessary, and the words are longer than what they would need to be. So for normal books, we find that using a verbose text, we can protect ourselves from printing errors at the printing office. An error correcting code works according to the same principle.

In this project we shall use a class of error correcting codes named *convolutional codes*. You do not have to implement the code yourself. At the we webpage, I have provided a routine *encoder.m* that takes a sequence $\mathbf{u}$ as input and maps it to a sequence $\mathbf{v}$. However, many types of convolutional codes exist, but in this project we will only consider two different ones, which we refer to as *Code 1* and *Code 2*; the routine expects this as input - see its documentation. What you need to know is the relation between K and N, which is

$$\text{Code1} : N = 2(K+2) \qquad \text{Code2} : N = 2(K+3).$$

The next step is to randomly scramble (a.k.a. interleave or permute) the encoded sequence $\mathbf{v}$. The reason for doing so will be explained shortly. In Matlab, you can use the built-in function *randintrlv* to do this. Since we only scramble the bits, we also have N bits as output.

The next step is to map the bits $\mathbf{v}'$ to a sequence of signal space vectors from an M-ary signal constellation. In this project we use only 1-dimensional signal constellations, so the signal space "vectors" boils down into 1x1 vectors (scalars). This is done as follows,

(1) Group the bits in groups of $\log_2(M)$ adjacent bits. In total we must have $N/\log_2(M)$ such groups.
(2) For each group, map the bits into a message m. There are M possible messages.
(3) For each message, map the message into a constellation point representing the signal space representation of the message.

Step 2 of the above can be done in many different ways, and we get back in Section 3 with different maps.

After obtaining a sequence of signal space values $\mathbf{a}$, the received signals are obtained by adding independent Gaussian noise with variance $N_0/2$ to all elements.

Let us now briefly return to the role played by the scrambler. Let us, for the sake of simplicity, use the approximation $N \approx 2K$. Further, assume $M = 4$ and $K = 1000$ so that we obtain $N = 2000$ coded bits and $2000/\log_2(M) = 1000$ messages $m_1, m_2, \ldots, m_{1000}$. A convolutional code is roughly what its name suggests: a convolution of its input with two impulse responses, and then a juxtaposition of the two results (which explains the "2" in the relation between N and K.). Therefore, each input bit in $\mathbf{u}$ has an influence in $\mathbf{v}$ that is only local, i.e., bit u_1 impacts $v_1, v_2, ..., v_8$ (perhaps), but clearly not v_{1000}. Therefore, bit u_1 only impacts the first messages $m_1 - m_4$. Now, with the scrambling, u_1 may impact message, say, m_{500}, but also u_{1000} may impact message m_{500}. As an outcome, the link between $\mathbf{u}$ and $\mathbf{m}$ is made more complex. The "temporal" structure of the convolutional code is lost, and this complicates decoding.

3. Signal constellations and bit-maps

In the project you should study 2 different setups.

(1) 4PAM with a Gray bit-mapping
(2) 4PAM without Gray bit-mapping

The constellations and bit-maps are shown in Figure 2 An important point here is to setup

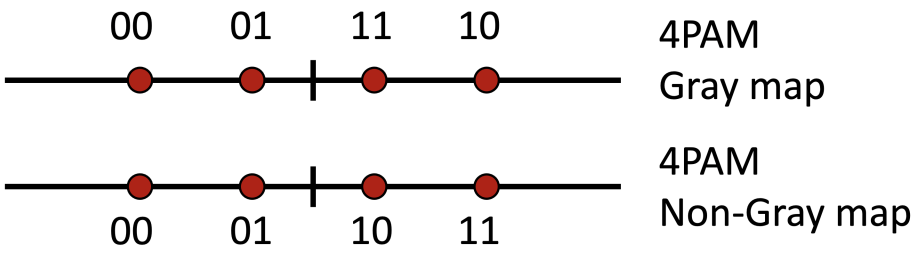

Figure 2. Constellations and bit-maps.

the signal constellation such that the average energy per information bit is under control. Note, that this refer to the energy per bit in $\mathbf{u}$, not in $\mathbf{v}$.

4. Non-iterative decoding

In this section we briefly go through non-iterative decoding, as a primer for iterative decoding. There are two versions of non-iterative decoding, *hard-output* and *softt-output*.

4.1. **Hard-output non-iterative decoding.** The simplest, but also worst approach, is to first demodulate the signal constellation using an ML receiver. This provides an estimated vector $\hat{\mathbf{v}}'$ which consists of 0/1. We can easily send this through the inverse scrambler (Matlab: randdeintrlv) to obtain $\hat{\mathbf{v}}$. Some of the bits in this estimate are incorrect, but we can rely on the error correcting code to fix those. On the webpage I have provided a routine (out_decoder.m) that takes an estimated vector $\hat{\mathbf{v}}$ and outputs the estimated vector $\hat{\mathbf{u}}$. However, to understand the syntax for the said routine, we must first go through soft-output non-iterative decoding. A block scheme for this type of receiver is shown in Figure 3.

4.2. **Soft-output non-iterative decoding.** The hard-output method just discussed is weak in the sense that the demodulation of the signal constellation does not provide any measure of reliability of its outputs. Consider for example a 2PAM constellation $\{\pm 1\}$ and assume that we face two received samples $r_1 = 0.001$ and $r_2 = 1.01$ The hard output demodulator would for these received samples output the estimated symbols $+1, +1$. However, we can be fairly certain that the second value is reliable, but we basically don't know if the first one is correct or not - it is close to a 50-50 situation. The decoder for

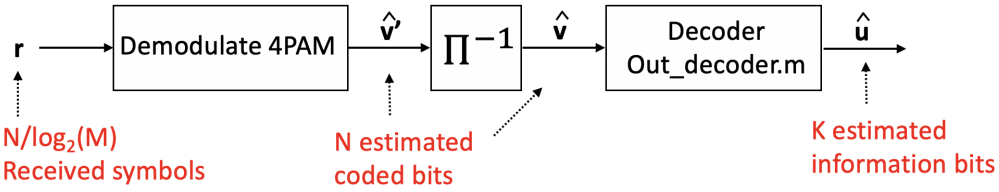

FIGURE 3. Block scheme for hard-output non-iterative decoding

the convolutional code would benefit from having some form of reliability information, and this is what a soft-output demodulator provides.

Let us for the sake of simplicity consider the first received sample which we denote by r_1. This sample contains the 2 bits v'_1, v'_2. We can now aim at computing the a-posteriori probabilities

$$p_k(1) = \Pr(v'_k = 1|r_1) \quad p_k(0) = \Pr(v'_k = 0|r_1) \quad 1 \leq k \leq \log_2(M) = 2.$$

Since we must have $p_k(1) = 1 - p_k(0)$, we can express this in the form of a log-likelihood-ratio (LLR) as

$$L'_k = \log\left(\frac{p_k(1)}{p_k(0)}\right).$$

Let us now return to the computation of $p_k(1)$ and $p_k(0)$. From Bayes' rule we have

$$\begin{aligned} p_k(1) &= \Pr(v'_k = 1|r_1) \\ &= \frac{1}{\Pr(r_1)}\Pr(v'_k = 1, r_1) \\ &= \frac{1}{\Pr(r_1)}\Pr(r_1|v'_k = 1)\Pr(v'_k = 1). \end{aligned}$$

We can now turn to $\Pr(r_1|v'_k = 1)\Pr(v'_k = 1)$. By the law of total probability, we obtain

$$\Pr(r_1|v'_k = 1)\Pr(v'_k = 1) = \sum_{\forall \ell\text{: bit } k \text{ of } s_\ell \text{ is } 1} \Pr(r_1|s_\ell)\Pr(s_\ell).$$

Altogether, we obtain,

$$p_k(1) = \frac{1}{\Pr(r_1)} \sum_{\forall \ell\text{: bit } k \text{ of } s_\ell \text{ is } 1} \Pr(r_1|s_\ell)\Pr(s_\ell)$$

and

$$p_k(0) = \frac{1}{\Pr(r_1)} \sum_{\forall \ell\text{: bit } k \text{ of } s_\ell \text{ is } 0} \Pr(r_1|s_\ell)\Pr(s_\ell),$$

wherefore

$$
\begin{aligned}
L'_k &= \log\left(\frac{\sum_{\forall \ell:\text{ bit } k \text{ of } s_\ell \text{ is } 1}\Pr(r_1|s_\ell)\Pr(s_\ell)}{\sum_{\forall \ell:\text{ bit } k \text{ of } s_\ell \text{ is } 0}\Pr(r_1|s_\ell)\Pr(s_\ell)}\right) \\
&= \log\left(\frac{\sum_{\forall \ell:\text{ bit } k \text{ of } s_\ell \text{ is } 1}\Pr(r_1|s_\ell)}{\sum_{\forall \ell:\text{ bit } k \text{ of } s_\ell \text{ is } 0}\Pr(r_1|s_\ell)}\right), \qquad (1)
\end{aligned}
$$

where the last equality holds since all symbols are equiprobable.

We can now send the sequence $\{L_k\}$ (of length N) through the descrambler and then let the error correcting decoder decode the sequence. This is precisely what the routine out_decoder does. It takes a sequence of values L_k and finds the most likely sequence $\hat{\mathbf{u}}$. The hard-output demodulator discussed above can be obtained by setting the values of L_k to large constants, i.e., if a hard output demodulator makes the estimate $\hat{v}'_1 = 1$, then we can set $L_1 = 10$ and if $\hat{v}'_1 = 0$, then we can set $L_1 = -10$ (This corresponds to saying that the probability that a certain bit is 1 is 0.99995 which is sufficiently close to 1). A block scheme for this type of receiver is shown in Figure 4.

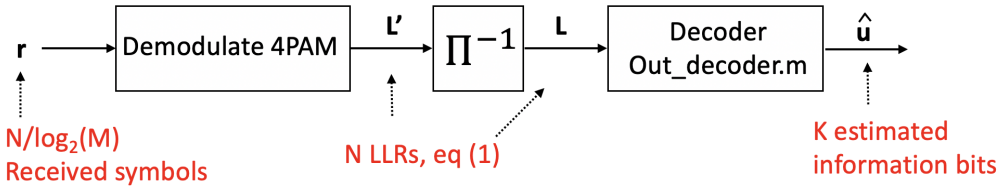

FIGURE 4. Block scheme for soft-output non-iterative decoding

5. ITERATIVE DECODING

Assume that you one day come home and finds a piece of paper at the kitchen table saying *"Hi honei. Cam yoi olease gorto thy sukerjarkit ond fuy somr frutts? Banenas and iranjes. Van yoy alto bry sime spoghitti and mayby some penots".*

This corresponds to a noisy message $\mathbf{v}$. We now have two options: (1) try to dig out the information in the text, or (2) try to correct the message. What we did in the non-iterative decoding above was to dig out the information $\mathbf{u}$. In this case, the information is "Buy Bananas, Oranges, Spaghetti, and Peanuts". The rest of the text is merely redundancy that allows us to find this information despite all the noise. But, as is quite apparent, we can also do something else, namely, try to correct the spelling mistakes. If we do that, we would obtain *"Hi Honey. Can you please go to the supermarket and buy some fruits? Bananas and oranges. Can you also buy some spaghetti and maybe some peanuts."*

An error correcting code can do the very same thing. Instead of observing the sequence $\mathbf{v}$ and aim for determining $\mathbf{u}$, it can aim at correcting the errors in $\mathbf{v}$ it can see. Again this can be done in a probabilistic fashion. This is precisely what the routine decoder.m does. It takes the sequence $\{L_k\}$ and updates it into an improved version. Thus, its output and

input represent the same thing, but its output is an improved version where some mistakes have been corrected.

Let us now look at what steps we have taken.

- We observe the sequence of received samples $\mathbf{r}$ and find a sequence $\{L'_k\}$ of length N, according to (1), containing the posterior probabilities of the bits in the sequence $\mathbf{v}'$. When doing so, all constellation symbols are uniformly distributed.
- We descramble $\{L'_k\}$ into $\{L_k\}$
- We let the error correcting code work on $\{L_k\}$ and find an improved version of it where some probabilities have been changed.
- We scramble the improved version $\{L_k\}$ into $\{L'_k\}$

We are now back at the demodulator again, but this time with some prior information about what the bits are. We can now redo the demodulation, but this time we will get a slightly different result since we are not in the same situation. When we did demodulation the first time, we did not know anything about the symbols, but now we do. Therefore, we can recompute the sequence $\{L'_k\}$ as

$$L_k'^{(2)} = \log\left(\frac{\sum_{\forall \ell:\text{ bit } k \text{ of } s_\ell \text{ is } 1} \Pr(r_1|s_\ell)\Pr(s_\ell)}{\sum_{\forall \ell:\text{ bit } k \text{ of } s_\ell \text{ is } 0} \Pr(r_1|s_\ell)\Pr(s_\ell)}\right), \tag{2}$$

where the symbol probabilities $\Pr(s_\ell)$ are found from $\{L'_k\}$.

An example follows. Assume $M = 4$ and 4PAM with constellation points $\{-3A, -A, A, 3A\}$. Let the bit mapping be (in the same order) $\{00, 01, 11, 10\}$. Now, assume that $L'_1 = 1$ and $L'_2 = -2$. It follows that

$$\Pr(v'_1 = 1) = \frac{\exp(1)}{1+\exp(1)} = 0.73 \qquad \Pr(v'_1 = 0) = \frac{1}{1+\exp(1)} = 0.27,$$

$$\Pr(v'_2 = 1) = \frac{\exp(-2)}{1+\exp(-2)} = 0.12 \qquad \Pr(v'_2 = 0) = \frac{1}{1+\exp(-2)} = 0.88.$$

The probability that the first symbol a_1 is $s_0 = -3A$ becomes, using the bit-mapping, $\Pr(s_0) = \Pr(v'_1 = 0)\Pr(v'_2 = 0) = 0.24$. In the same fashion,

$$\Pr(s_1) = 0.03 \qquad \Pr(s_2) = 0.09 \qquad \Pr(s_3) = 0.64.$$

It is now straightforward to calculate $\{L_k'^{(2)}\}$ based on (2). However, there is something that complicates the situation. It has been verified that to make the iterative process to work, one must slightly change $\{L_k'^{(2)}\}$ according to

$$L_k'^{(i)} = L_k'^{(i)} - L_k'^{(i-1)},$$

that is, the input LLRs must be subtracted from the output LLRs of the demodulator. Understanding why goes far beyond the scope of this project, but in the literature this fact is referred to as the *Turbo Principle* or *Principle of Extrinsic Information*. After doing so, we can send this through the descrambler, let the error correcting code improve it, and iterate this as many times as we would like. Once we have iterated a predetermined number of time, the error correcting code changes its objective finds the sequence $\mathbf{u}$.

A block scheme for this type of receiver is shown in Figure 5.

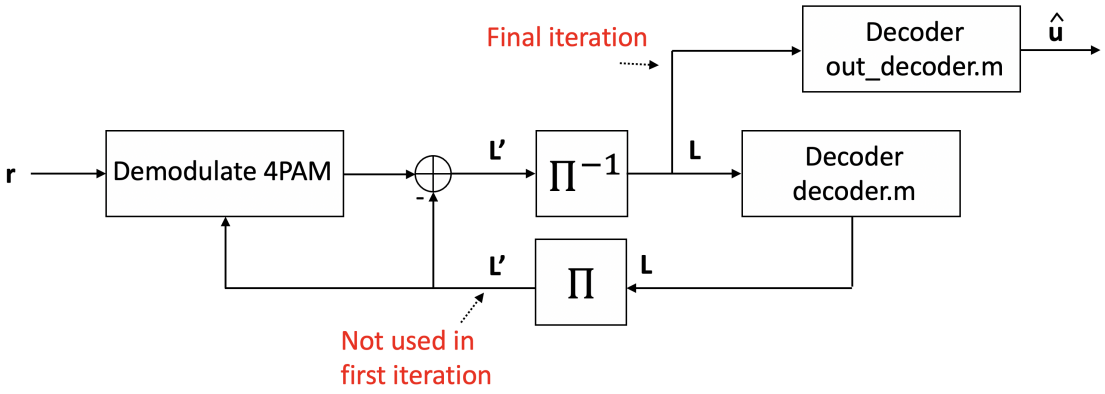

FIGURE 5. Block scheme for iterative decoding

6. TASKS

In this project you should implement a simulator that simulates the system using an iterative receiver. In particular, you should

- Write code that implements Equation (2)
- Investigate both codes for both bit-mappings
- Generate BER curves versus E_b/N_0
- Investigate the effect of iterations. This includes an investigation of how well the system performs with a single iteration, i.e., soft-output non-iterative decoding
- Investigate the performance of hard-output non-iterative decoding
- Investigate how many simulations that are needed

As you will find out, the value of K alters the performance. The BER is typically reduced as K grows. So, if you wish to simulate 10^6 bits, then you have many options. For example, you can set $K = 10^6$ and run the simulation once. Another alternative is to set $K = 1000$ and run it 1000 times, and then average the BER. In practice, the number K impacts latency. The larger the K, the more latency there is. I recommend to select $K = 1000$ as a default value.

You should write a short report where you present your results. Specific questions:

(1) Which is the better code?
(2) Which is the better bit-map?
(3) Are there unique answers to the above questions, or do they depend on other parameters such as SNR, number of iterations, etc?
(4) What is the gain of iterations?
(5) How well does hard-output non-iterative decoding work?
(6) Briefly investigate the impact of K on the performance, e.g., try $K = 1000$ and, say, $K = 10000$