ELEC 6131: Error Detecting and Correcting Codes
Lecture 1: Introduction

The course is divided into several modules. Each module (corresponding to a Lecture Note) should take one week to cover. I have tried to make the notes roughly the same size. However, depending on students' background and interest, it is conceivable that we spend more time on one topic than another.

There will be 11 lecture notes, thus 11 modules. I have intentionally kept the number of modules at 11 and not 12 to have some slack. Present note is the start of the course, i.e., Lecture Note 11.

Following is the list of the lecture notes:

- Lecture Note 1: Introduction (this note).
- Lecture Note 2: Introduction to Algebra.
- Lecture Note 3: Galois Fields.
- Lecture Note 4: Linear Block Codes.
- Lecture Note 5: Important Linear Block Codes.
- Lecture Note 6: Cyclic Codes.
- Lecture Note 7 : BCH Codes.
- Lecture Note 8 : Reed Solomon Codes.
- Lecture Note 9 : Convolutional Codes.
- Lecture Note 10 : Concatenated Coding and Turbo Codes.
- Lecture Note 11: LDPC Codes.

I assume that you have taken a basic digital communications course and are familiar with different types of modulation. If you have any problem regarding these topics you may refer to any text covering these topics. It may also help if you refer to my previous years ENCS 6161 or ELEC 6831 if you do not mind reading handwritten notes.

I have added an appendix presenting an informal coverage of some basic ideas from information theory without which appreciation of the effectiveness of coding/decoding schemes we are about to learn will be difficult.

- Appendix A: An Intuitive Look at Information Theory.

Now let's start talking about what we will learn in this course.

Why and How of Error Detection/Correction Coding

Most of the course will be spent on learning how to encode and decode the data to be transmitted or stored. So, the **HOW** part will be treated almost exclusively in the rest of the course. Let's pause for a few minutes and talk briefly about the **WHY.** That is, "Why do we use coding?"

You may find the question strange since the title of the course is “Error Detecting and Correcting Codes”. So, you may say it is clear, we use coding to detect and if possible correct errors, hence making communications reliable. While the answer is somehow correct, it is not all the answer since coding is not the only way to avoid errors.

Take as an example, a communications link using a digital modulation such as Binary Shift Keying (BPSK). Hopefully you remember from the digital communication course you have taken already, ELEC 367 or ELEC 6831, that the probability of bit error also called Bit Error Rate (BER) is given by:

$$p_b = Q\left(\sqrt{\frac{2E_b}{N_0}}\right). \qquad (1)$$

Appendix B summarizes results like this for quick reference. In this equation E_b and N_0 denote the energy per bit and noise power density respectively. $Q(x)$ is the tail probability of a Gaussian random variable with mean zero and variance one. You may refer to Appendix A for definition of Gaussian distribution and $Q(x)$.

Figure 1 shows the BER against $\frac{E_b}{N_0}$ for BPSK (also the same for QPSK).

Assume that your E_b/N_0 is 4 dB. Then you get a BER of roughly $10^{-2} = 0.01$. This means that on the average one out of 100 bits will be flipped from 0 to 1 or vice versa. This may have been tolerable if we still used old techniques such as Pulse Code Modulation (PCM) where the voice samples were sent sample after sample. Then a sample represented by eight bits may encounter an error with probability $1 - (1 - 10^{-2})^8 \approx 0.$. While 8 erroneous bytes out of 100 bytes is quite high, it is still better than nothing.

However, today’s digital communication systems use packet based transmission. This means that instead of transmitting data a byte at a time they send a bunch of bytes into a packet and send them together. For example in the Linear Predictive Coding schemes used from 2G onward, every 20 ms of speech, i.e., every 160 speech samples is compressed into a 160 bit packet (assuming 8 fold compression). With $BER = 10^{-2}$, the average probability that a packet contains one or more error is$1 - (1 - 10^{-2})^{161} \approx 0.81$. This means that 80% of what you say over the phone will be lost. To see this effect for yourself, I suggest that you record a minute or two of your voice using your computer’s microphone or your cell phone, save it in a binary file, add to it a random bit stream that generates one with probability 0.01 and see what happens. This will only be five to ten lines of Matlab code and the whole effort should not take you more than 10 minutes.

However, today’s digital communication systems use packet based transmission. This means that instead of transmitting data a byte at a time they send a bunch of bytes into a packet and send them together. For example in the Linear Predictive Coding schemes used from 2G onward, every 20 ms of speech, i.e., every 160 speech samples is compressed into a 160 bit packet (assuming 8 fold compression). With $BER = 10^{-2}$, the average probability that a packet

contains one or more error is $1-(1-0.01)^{161} \approx 0.8$. This means that 80% of what you say over the phone will be lost. To see this effect for yourself, I suggest that you record a minute or two of your voice using your computer's microphone or your cell phone, save it in a binary file, add to it a random bit stream that generates one with probability 0.01 and see what happens. This will only be five to ten lines of Matlab code and the whole effort should not take you more than 10 minutes.

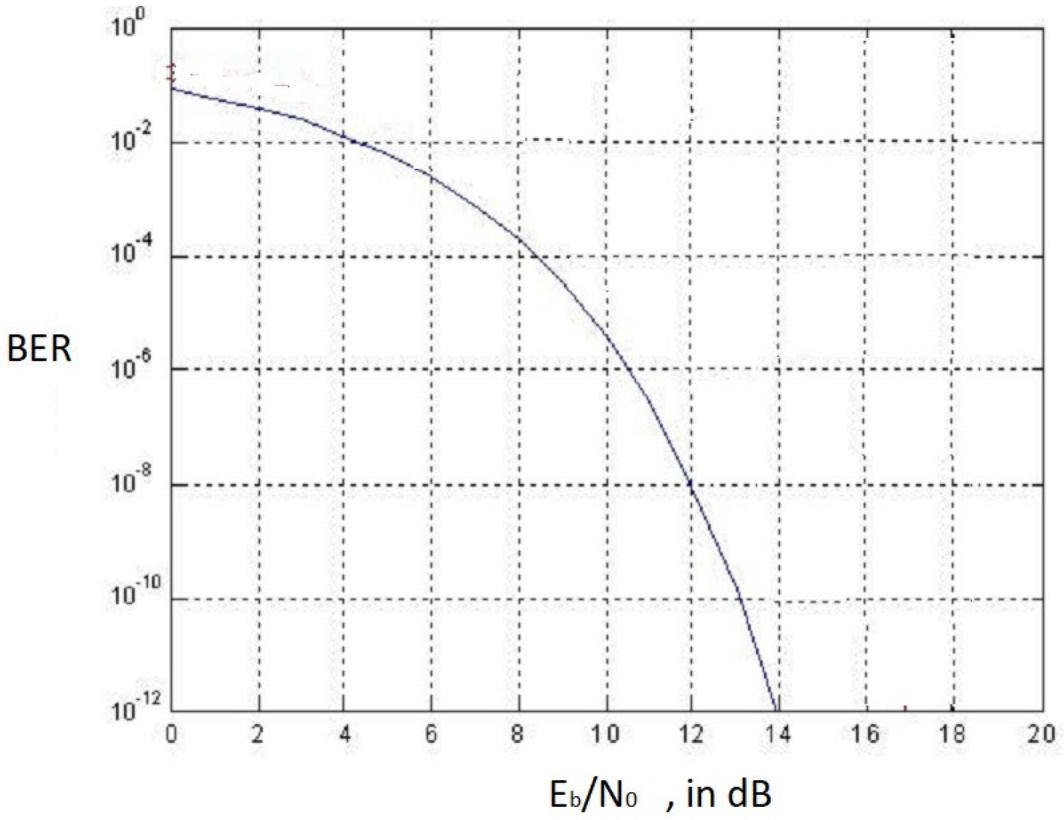

Figure 1: Performance of BPSK

I would like to emphasize the fact that speech and even audio are easy applications in terms of bandwidth requirement. Image and video are much more bandwidth hungry and also are transmitted in longer packets.

For example in video coding technique used almost everywhere for video transmission and storage that frames of video after being compressed are packed into transport stream (TS) packets each with a length 188 bytes (1504 bits). In order to have low packet loss, we need to have very low BER.

Assume the probability of bit error is $BER = 10^{-7}$. This means one erroneous bit in 10 million bits. While this seems quite low it is not sufficient for decent quality video. The probability that a packet si in error can be found to be $1-(1-BER)^{1504} \approx 1.510^{-4}$. Consider a TV program with a bit arte of 20 Mbps (one million bits per second). A simple calculation shows that the

number of packets transmitted per second is roughly 13,300. Multiplying these two numbers, the number of packets to be in error per second is found to be 2. Note that while two erroneous packets in 13,300 packets may seem low, the effect is far from being negligible. In an MPEG data streams packets are correlated, i.e., a packet may contain information to be used for decoding of other packets and loss of one packet may result in many other packets ending up being useless.

To avoid loss of packets, usually a bit error rate of 10^{-10} or lower is required. To have $BER = 10^{-10}$, according to Figure 1, we need to have an $\frac{E_b}{N_0}$ of 13 dB.

Now, let's make a detour to see what is the theoretical minimum required $\frac{E_b}{N_0}$ and how low we can go using modern day error control coding schemes.

A bit of Information Theory:

In 1948, Claude E. Shannon defined the capacity of a channel as the maximum rate at which you may communicate over a channel with arbitrarily small probability of error. The probability of error tends to zero asymptotically as the size of the codeword increases without bound.

In another word, the capacity of channel, C, is a quantity assigned to a given channel. Using appropriate signalling (coding) you may communicate over that channel at a rate as close to C as you wish. On the other hand, the rate cannot exceed the capacity without significant deterioration of the BER.

In particular, Shannon showed that for an Additive White Gaussian Noise (AWGN) channel is:

$$C = Wlog_2\left(1 + \frac{P}{N}\right) \text{ bps.} \qquad (2)$$

W is the available bandwidth in Hz., P is the received signal's power and N is the noise power.

Assume that we transmit at the maximum allowable rate of $R = C$. Then $R = Wlog_2\left(1 + \frac{P}{N}\right)$ bps. The energy per bit is the power (energy per second) divided by the bit rate (the number of bits transmitted per second, i.e., $E_b = \frac{P}{R}$. We can also write the spectral density N_0 as the noise power divided by the bandwidth, $N_0 = \frac{N}{W}$. Substituting theses in Equation (2), we get:

$$\frac{R}{W} = log_2\left(1 + \frac{R}{W} \times \frac{E_b}{N_0}\right) \text{ bps/Hz.} \qquad (3)$$

The quantity $\eta = \frac{R}{W}$ in the units of bits/second/Hz. Is the bandwidth efficiency, i.e., the number of bits transmitted each second for each Hz. of available bandwidth. For BPSK, the bandwidth efficiency is $\eta = 1$ bps/Hz. Using (3), one can write the required $\frac{E_b}{N_0}$ to get error free transmission for a bandwidth efficiency η:

$$\frac{E_b}{N_0} = \frac{2^{\frac{R}{W}}-1}{\frac{R}{W}} = \frac{2^{\eta}-1}{\eta} \qquad (4).$$

For $\eta = 1$ bps/Hz, we get $\frac{E_b}{N_0}$=1, i.e., 0 dB.

Going back to the required $\frac{E_b}{N_0}$ of 13 dB calculated in previous section, we observe that using BPS without coding we need to waste 13 dB (20 times) more power than is theoretically foreseen. This means that, for example, in case of mobile communication, our handset will be much bigger and will consume 20 times more power. So, instead of charging every 23 hours, you need to charge your phone every 1.2 hours.

The coding techniques developed in the past 10-20 years such as Turbo Codes and Low Density Parity Check (LDPC) codes can close the gap operating very close to Shannon limit.

The cost of reducing power consumption is an increase in bandwidth requirement and coding and decoding complexity. We will talk about this during the course.

Now that we have talked about the WHY of coding, let's talk briefly about the HOW. This gives us a preview of what we will be talking about in the rest of the course.

Error Control Coding Strategies:

How sharper than a serpent's tooth it is to have a rhankless child!

When reading the above expression, you very quickly detect an error: you find the term rhankless strange. A little reflection, encourages you to change "r" to "t" and correct the expression into:

How sharper than a serpent's tooth it is to have a thankless child!

The first observation, i.e., realizing that there is an erroneous word in the sentence is an example of *error detection* while finding, or at least guessing the correct word, is an instance of error correction. The former is possible due to the fact that not all combinations of the letters of the alphabet are legitimate words, i.e., the fact that the English language, like any other language has some structure. The latter, i.e., error correction was possible because there is enough difference (distance) between words.

The same idea is used for error detection and error correction in digital communications. Assume that the messages to be transmitted are divided into k bit words. If we transmit the data uncoded, we have to use all 2^k possible combinations of k bits. Any error in a transmitted k-bit word generates another k-bit word and since we may have any of the 2^k possible k-bit words, the receiver has no way to know whether an error has occurred or not. For example, assume that $k = 4$, that is, we transmit 0000, 0001, …, 1111. Assume that the transmitter sends 0000. Any error in one or more bits results in receiver getting of the other 15 bit patterns and therefore, not being able to detect an error.

Now assume that in addition to 4 bits, the transmitter sends 5 bits, the fifth bit being a parity bit. A parity bit may be formed by counting the number of ones in the stream and if it is odd adding a one and if the number of ones is even add a zero. For example, 0000 will changed into 00000 and 0001 will be changed into 00011. So, the receiver expects to receive 5-bit patterns with an even number of 1's. So, if the transmitter sends 00000 and one of the bits is flipped, the receiver receives a pattern with one 1 and rejects it as an error. Adding one parity bit the number of possible patterns at the receiver has increased to 32, while the size of our dictionary (in technical terms, the codebook) has remained the same. So, we have taken one half of the 5-bit patterns as legitimate codewords and discarded the other half.

Forming of parities as discussed above is done using exclusive-OR (XOR) operation or mathematically speaking doing modulo-2 addition. The fifth bit added in the above example is given by $p = x_0 \oplus x_1 \oplus x_2 \oplus x_3$ where $x_0, x_{1,}, x_2 \; and \; x_3$ are the data bits.

Parity bits can also be used to add error correction capability. In order to do that, we need several parity bits each acting on a subset of bits. As an example, take the same 4 bits we talked about in the above paragraph. Let's form 3 sets each containing 3 of the bits. The sets will be distinct but will overlap. To each set, we add a parity bit that is the result of the modulo-2 addition of the input bits in that group. The set are:

$$S_0 = \{x_0, x_1, x_3, \; p_0 \; \}$$

$$S_1 = \{x_0, x_2, x_3, \; p_1 \; \}$$

$$S_2 = \{x_1, x_2, x_3, \; p_2 \; \}$$

This is called a (7, 4) Hamming code whose Venn diagram is shown in Figure 2. It is called a (7, 4) code since it encodes the 4 bits $x_0, x_{1,}, x_2 \; and \; x_3$ into 7 bits $c_0, c_{1,}, \ldots, c_6$ defined as: $c_0 = x_0$, $c_1 = x_1$, $c_2 = x_2$, $c_3 = x_3$, $c_4 = p_0 = x_0 \oplus x_1 \oplus x_3$, $c_5 = p_1 = x_0 \oplus x_2 \oplus x_3$ and $c_6 = p_2 = x_1 \oplus x_2 \oplus x_3$.

Assume that the four data bits are 0101. The encoded bits will be 1010101. These 7 bits are transmitted over the channel and the receiver instead of receiving it receives 1010111. The decoder computes three syndromes $s_0 = p_0 \oplus x_0 \oplus x_1 \oplus x_3$, $s_1 = p_1 \oplus x_0 \oplus x_2 \oplus x_3$ and $s_2 = p_2 \oplus x_1 \oplus x_2 \oplus x_3$. The result will be $s_0 = s_2 = 1$ and $s_1 = 0$.

The syndrome s_0 being equal to one indicates that one of the bits x_0, x_1, x_3 (or possibly p_0) has been flipped. The syndrome s_1 being zero, the decoder concludes that the error is not among the bits x_0, x_2, x_3 and p_0 . Syndrome $s_2 = 1$ points finger at x_1, x_2, x_3 and p_2. Bits x_2, x_3 already exonerated and p_2 never implicated, the conclusion is that the error is in x_1. Flipping x_1, the decoder reconstruct the original message 1010101. Referring to Figure 2, we can see that this decision is based on the fact that the result of syndrome calculation place us in the intersection of the sets S_0 and S_2 excluding S_1, i.e., $S_0 \cap S_2 \cap \overline{\overline{S_1}}$ and from the Venn diagram, we find that

the erroneous bit must be x_1. Note that with 7 intersection areas, this code can correct a single error in any of the seven bits.

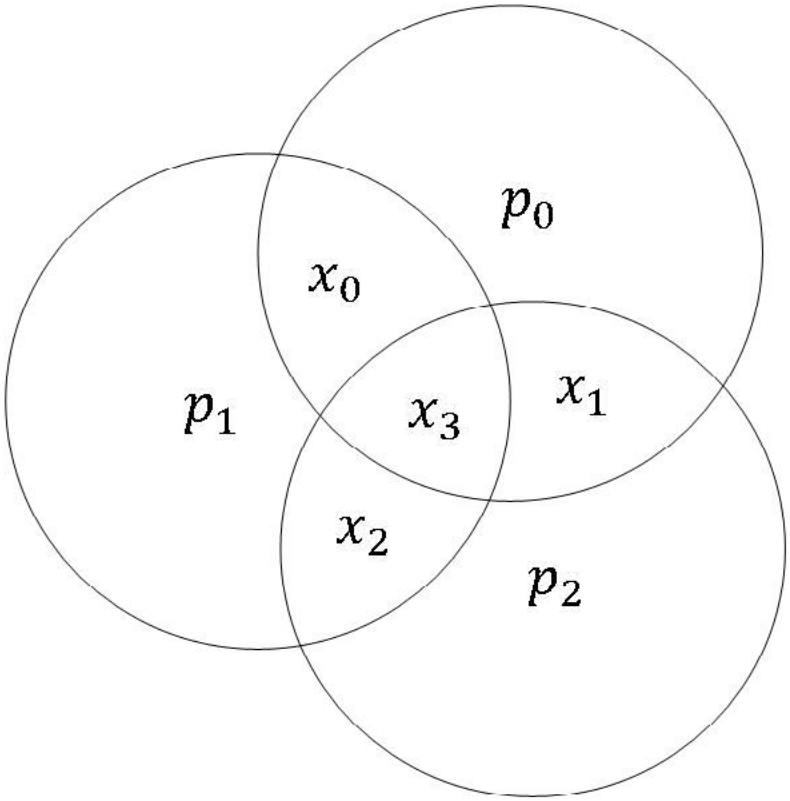

Figure 2: (7, 4) Hamming code

The (7, 4) Hamming code discussed above while not very efficient and performant, is very useful tool for learning and appreciating the mechanism of error control coding. Throughout the course, we will encounter it many times as a toy example to clarify some fundamental concepts that are difficult to explain using a longer code.

Now that we have some sense of error detection and error correction, you may ask whether to perform error detection or error correction. More importantly, you may ask what the value of detecting errors if we cannot correct them. For one thing you may discard erroneous messages. You may also ask for retransmission. This is the basis of the error control strategy called Adaptive Repeat Request (ARQ) where the receiver checks the message for the presense of errors and if it detects any error asks for retransmission. ARQ is very efficient since the detection of errors is simpler and requires less overhead as we saw in the example of the simple Hamming code we discussed before. Furthermore, there is retransmission only if there is an error. This makes the scheme adaptive to the channel condition. The problem with ARQ, however, is the delay in retransmission when there is need for one. This may make ARW unsuitable for real time delay sensitive applications such as live video and voice calls over the channels such as satellite communication links where the propagation delay is quite long.

The other error control strategy is Forward Error Correction (FEC) where enough parities are added so that a number of errors expected to occur depending on our information about the channel condition can be corrected without need for retransmission. The problem with FEC is that often times, the channel condition is better than the average condition for which we have chosen the code and the parity bits are of no use. On the other hand in cases where the *channel condition is worse than our preconceived average condition, the errors remain* uncorrected and even the decoder may add extra errors when trying to correct the errors.

Different Types of Codes

There are basically two main classes of codes:

- Block codes and,
- Convolutional Codes.

In a block coding scheme k bits enter the encoder and n>k bits are generated, i.e., a k-dimensional binary vector is mapped into an n-dimensional vector. Since the number of k-dimensional and n-dimensional binary vectors is 2^k and 2^n, respectively for each of the 2^k codewords there are $2^{n-k}-1$ binary vectors that are not codewords. These illegitimate bit streams that grows exponentially with the number of parity bits $n-k$ provide the code with error detection/correction capability. The rate of the code is defined as $r=\frac{k}{n}$. This is a dimensionless entity that you should be careful not to mix with the transmission rate in bits per second.

The number of the bits at the output of the decoder exceeds the number of input bits by n-k. If the k input bits (information bits) appear unaffected and all next to each other followed by n-k parity bits, the code is called *systematic.*

Another way to classify codes is based on whether they are linear or not. In a *linear* code the sum of any two codewords is another codeword. Linearity gives a very nice structure to a code simplifying its representation, encoding and decoding.

In a linear code parities are formed by modulo-two addition (XORing) of the information bits. So, in the construction of a linear encode the only gate we need is exclusive OR (XOR) and no AND or OR gates are used.

In this course, we mostly talk about linear codes. That is why the first thing we do after this introduction is to go over some basic concepts from *linear algebra.*

We talked about block codes. The other class of codes consists of convolutional codes. Convolutional codes are linear trellis codes.

In a convolutional code, the input to the encoder is a k-bit word, the same as block codes (k is, however, usually small, e.g., one or two). The output, unlike the block codes, is not only a function of the input bits but also depends on previous inputs (and possibly previous outputs).