# Error Control Codes: Convolutional Codes

## Introduction

This is not really a “report,” but more a conglomeration of our notes, what we’ve learned, what we did, and what we actually did on the project. If you want to see our notes (more detailed in terms of the math; didn’t include them because it’s not really super imperative to our understanding with this specific project), or the current code for our unsuccessful implementation of the sequential decoding algorithm, feel free to email us.

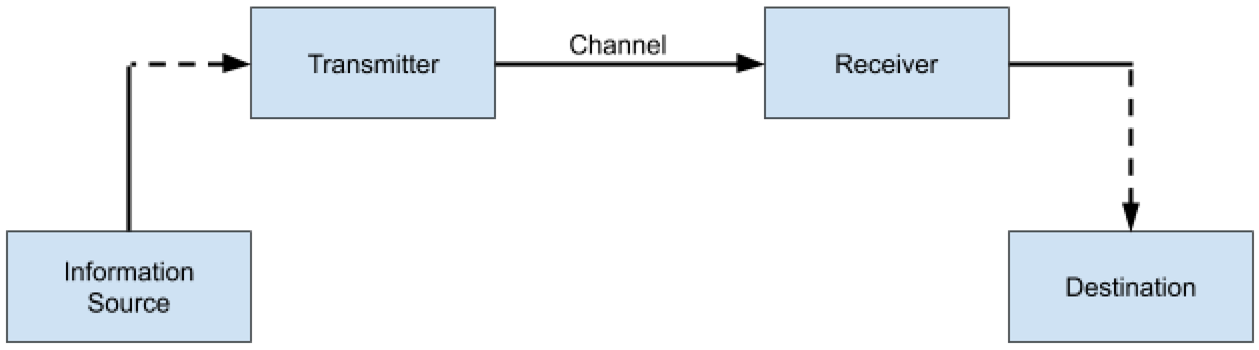
## Motivation

All of us kind of came together because we really liked the idea of doing error control codes. We were all in Computer Networks last semester, where we learned about the Hamming Code. In fact, two of our group members made a transistor-level implementation of Hamming error detection and correction for their Microelectronic Circuits final project.

In addition, we know that this topic was kind of Sarah’s domain, and so we knew that we would be able to get a lot of support and resources from her (which sadly, I don’t think we utilized that much).

## Overview

Error control codes are algorithms for detecting and more importantly, correcting, errors that occur during data transmission. The study of them first began with Claude Shannon in 1948, who published a book called *A Mathematical Theory of Communication*. In this source, he basically lays out how communication occurs nowadays. It starts off with an information source that wants to send a message. The transmitter actually sends the message. This message is sent across the channel. The message is then picked up by the receiver. This then arrives at the destination, whatever the message was intended for.



*After making this I realized it’s not a very compelling graphic (maybe because I made it in Google Drawings?). Anyways, this is the path the message takes.*

Error control codes are pretty important. It's very expensive to just increase transmission power, and it's also not possible to increase it to the point where no errors would be generated. Also, there is channel interference that will cause interference anyways, and so error control codes hopefully will eliminate that.

## Error Control Codes

The study of error control codes is obviously not limited to just convolutional codes. The following three sections detail three other error control methods:

### *Repetition Code*

Basically, this method involves sending one bit multiple times over the channel. However, this is very wasteful in terms of resources - it is more ideal to send bits once each time. In addition, if multiple bits are flipped, it's possible that error wouldn't be detected at all or not corrected.

### *Parity Bit Check*

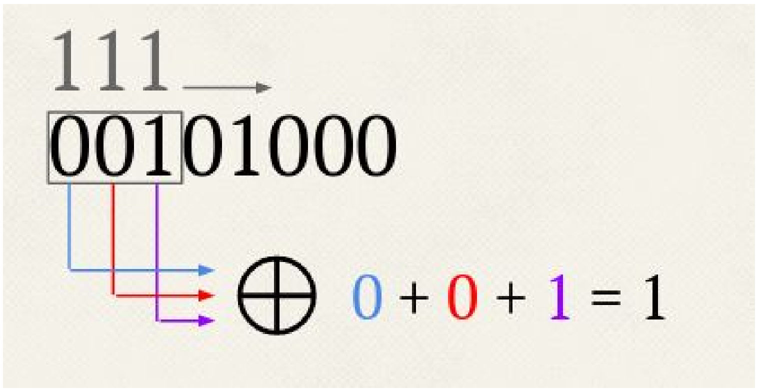
There are multiple ways to use parity bits as a form of error control. One way is to sum the number of 1s in a message and mod that together (basically, count if there is an odd or even number of 1s). If there is a bit flip, then the parity bit would not match the even/odd number of 1s, and the receiver can request retransmission. However, if there are two, or four, or six (basically even numbers) flips, it won't be detected. Also, what if the the parity bit is flipped? However, parity bits, the idea of an 'error check bit,' are fairly important in error control codes - and do appear in Viterbi.

### *Automatic Repeat Request (ARQ)*

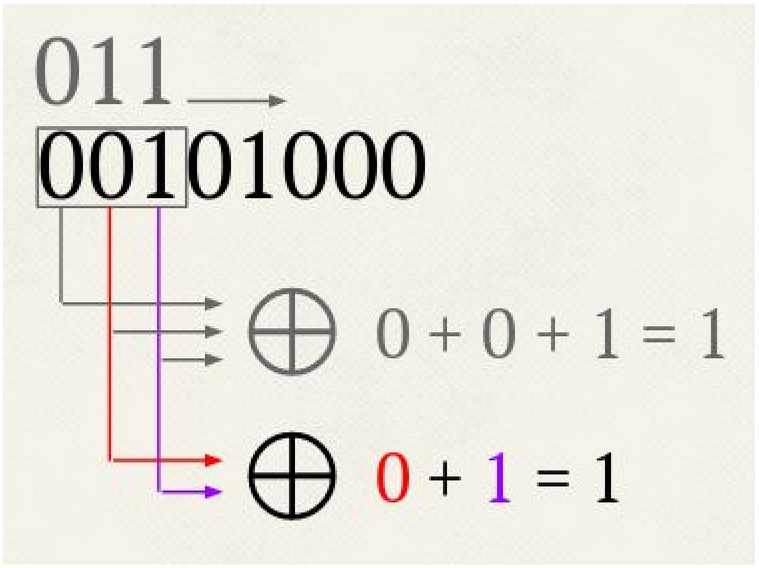
There are multiple amalgamations of this, but one particular one is a negative acknowledge request. Basically, if the receiver gets a message, it sends something back to the transmitter. Although this might seem wasteful, it's actually a pretty safe way. If the receiver only sent something when a message wasn't proper, it is possible it wouldn't make it to the transmitter. And so the transmitter would never 'know'. In this case, if the transmitter doesn't get an acknowledgement, it would simply send the message again - which might be mildly wasteful but the better alternative.

## Convolutional Encoding

Convolutional encoding only produces parity bits. The number of parity bits depends on how many generating polynomials there are, which are the things that basically determine how one would compute within a certain window size. The following two diagrams show the computation of the parity bit for two generating polynomials, $g_0$ = 111 and $g_1$ = 011. The window length is thus, 3 (the size of the polynomial) and the number of parity bits is 2 (for the number of polynomials). Our original message is 1010. We have to pad with 0s in order to get the correct computation of parity bits, hence we are actually encoding the message 00101000.



The above image shoes the computation for the first generating polynomial. Essentially, each bit in the window is ANDed with the polynomial. Then, binary addition is performed and the parity bit is computed. In this case it is 1.



Then, we to the second generating polynomial. Each bit is ANDed together again with the polynomial. Notice that since we have a 0 in $g_1$, the first bit of that window wasn't added. The parity bits for this first window are thus 11. This window continues one bit at a time until it reaches the end.

Though we never did a hardware implementation of this, we could use a shift bit register, an AND gate to compare the polynomials to the windows, and a XOR gate to perform the binary addition with the selected bits.

**Convolutional Decoding**

Convolutional decoding must decode the encoded sequence accurately, in a reasonable amount of time, and be able to correct any errors that occurred during transmission. Here we discuss two different decoding schemes.

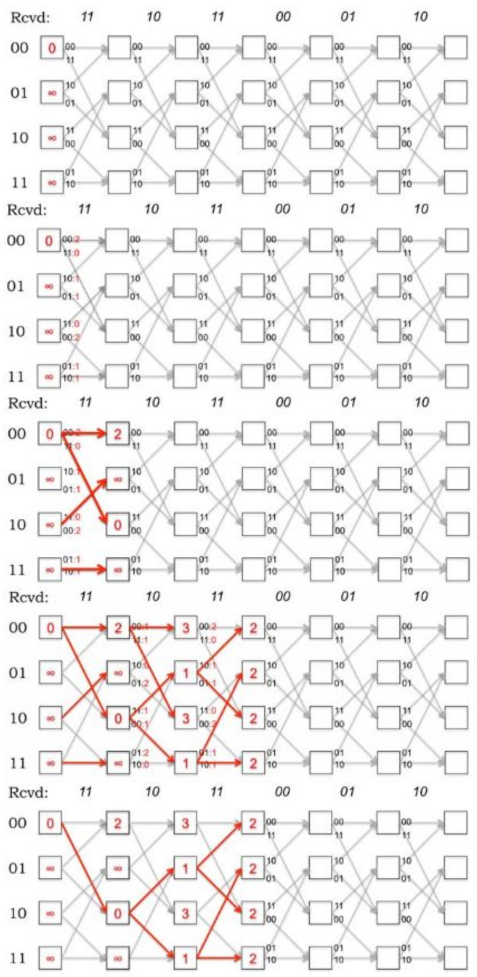
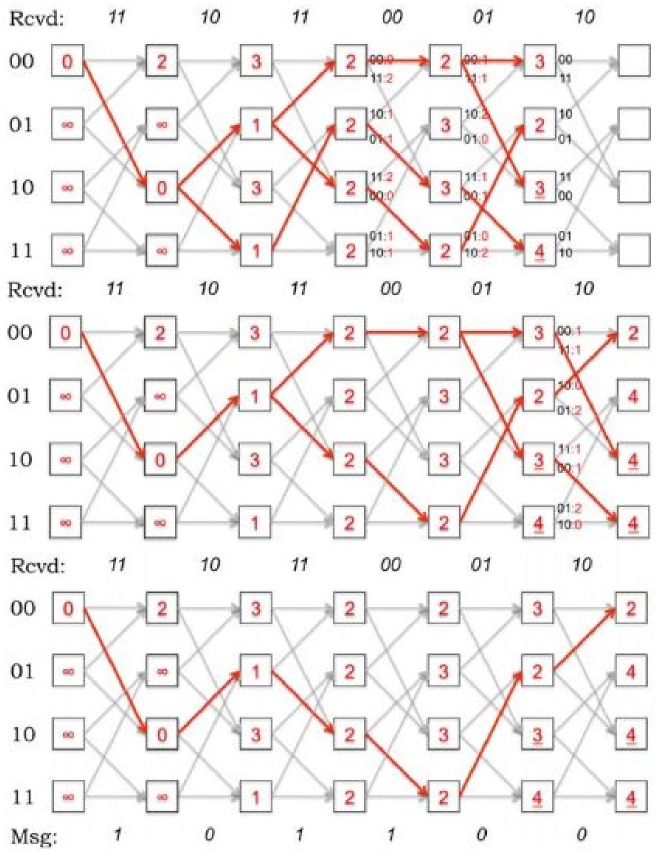
*Viterbi Decoding*

Convolutional decoding is a little involved obviously. So there are things called maximum-likelihood decoder. These look at all the possibility encoded sequences (there are $2^N$ sequences for a message of N-bits, so for our 4-bit one there 16 possibilities) and choose the one with the lowest Hamming distances. The idea here is that the minimal error is the most likely original sequence.

However, the time complexity for running all of that is very large, especially as the length of the original message increases. So, there is a modified version of maximum-likelihood, called Viterbi. Viterbi still explores all possible paths, but are able to 'delete' paths (so not every sequence is explored). Thus, the time complexity is a lot smaller, but the accuracy is the same as maximum likelihood.

We mention 'paths'. This is because Viterbi uses a structure called a trellis, which is like a finite state machine but it keeps track of the time steps as well. The following sequence of pictures shows the path through a trellis (from the MIT lecture), however we will describe the process. First, it is assumed that the first two bits will be 0s, so it starts off on the top right, as you notice. And then, it explores the paths from that box. A trellis is the same for codes of a certain bit length, so the actual structure is predetermined.

From there, we compare what we actually received to whatever the path says, and tally that up using hamming distances. This allows us to get rid of some of the paths because if two paths merge, and one has a smaller total Hamming distance, the other path can be deleted. Eventually we can see that by the end, only one path remains. This is then backtracked. The top two 'rows' of the trellis are 0s, and the others 1s - so this is how we can get our received message.



The picture on the left is the initial 5 iterations, while the right are the final iterations. This is not for the same message we encoded above, however, it is for the same convolutional code polynomials and the same number of them.

*Sequential Decoding*

Viterbi has a time complexity exponential to to the window length, so it isn't always ideal for large window length. When the window length becomes too large, other decoding schemes are more favorable. One is sequential decoding. What sets sequential decoding apart from Viterbi is that it doesn't explore all possible paths. Instead it makes a locally optimal decision based on its decoding stage. This, while it makes it faster, does make it less accurate. It is not always true that the locally optimal solution will contribute to the globally optimal solution. Once again, like many things in computer science it boils down to time complexity vs accuracy.

Sequential decoding uses the Fano matric, which is a metric based on probability. This metric is largely based on the length of the probable path - if the length is longer, it is most likely the decoded sequence. At each given decoding stage, it computes the most probable next bit that it should be and chooses that bit. It utilizes something called a code tree, which is basically a tree of all the possibilities. This is more of a scheme, and there are actually two sub algorithms that fit into this sequential decoding scheme, called Fano and Stack Algorithms.

The Fano algorithm is able to backtrack through the code tree if the metric gets updated. So at each stage, it compares its current path, its predecessor path, and its successor path (in terms of the code tree). From here, it can make a decision to stay on the current path of the tree, move to the predecessor path, or if it actually miscalculated the probability of the current / future paths and it actually has to move back.

The stack algorithm takes advantage of the stack data structure and stores the probabilities in there. It only goes through everything ones, and it is able to do this due to the storing and updating the current probabilities. Hence, the time complexity between this and Fano is a trade-off. Fano can backtrack and might have to explore the same paths multiple times because it doesn't save any information. However the Stack has to go through the probabilities in the stack, adding time.

They both should basically yield the same answer though, so it is just a matter of implementation. Fano is preferable in hardware implementation as it doesn't require the use of a memory space. However, the Stack algorithm lends itself very well to a software implementation.

**Our Work**

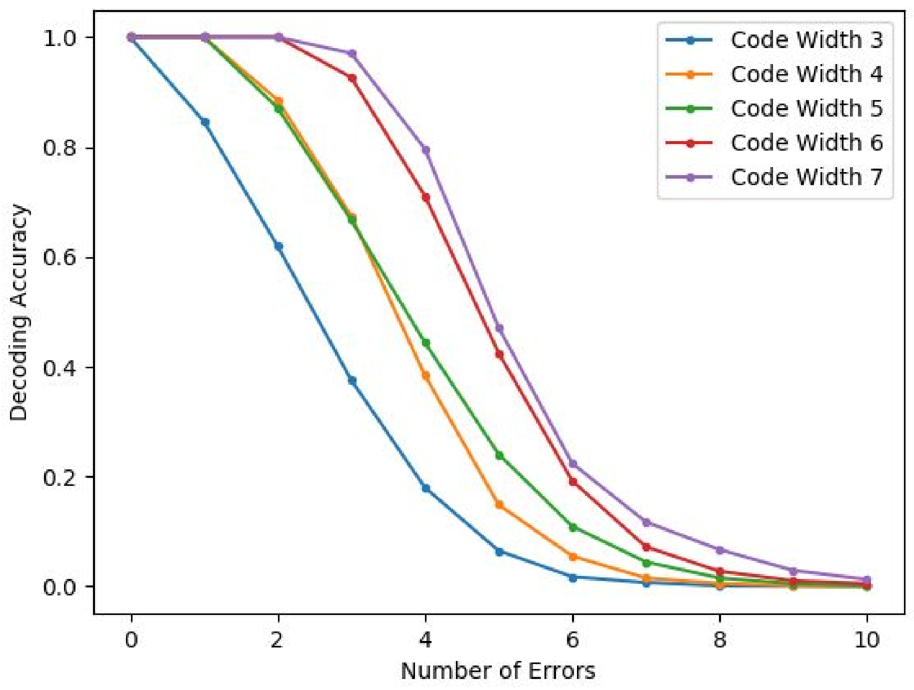
You can view our code here: https://github.com/prava-d/Convolutional-Codes

(The Viterbi decoding implementation should be fairly well commented, unlike the not-working code . . . which isn't uploaded anyways. But if you want to see that it can quickly be commented.)

Our implementation prioritized functionality over optimization, so our implementation isn't the best-case time complexity. However, it did work!

Our program can be run with a specified number of trials, and with various code widths. We are also able to introduce a specified number of errors.

However we wanted to conduct analysis on Viterbi (at least, our implementation of Viterbi) to see if it matched what we expected from the existing literature. So we generated a graph that varied code widths and number of errors introduced (as you can see on the following figure). For this we analyzed code widths from 3 to 7 on a 10 bit sequence, and generated 0 to 10 errors. The accuracy is an analysis of success for 5000 trials.



As error is introduced, no matter what the code width is, the accuracy decreased towards 0. Also, as the code width increases, the accuracy per number of errors also increases. There is a small deviation in the graph that shows that code width 5 had a slightly lower accuracy then code width 4 in for that particular number of errors. We are unsure, as these were 5000 random trials and we didn't output all the sequences. However, we just assumed it was the specific convolutional codes that we were using.

The codes that we used to generate this graph were found in a paper under a section called "non-catastrophic convolutional codes". We also tested out the section that was just "catastrophic convolutional codes," which, as expected, were all 0% accuracy for the tests we ran. Also originally when running this, we tested out whatever the *Voyager's* convolutional code is for their error control system, which is (79, 109). It also has pretty high accuracy.

**Future Steps**

Convolutional codes are very popular in data communications. However, both Viterbi and Sequential aren't actually the most popular decoding scheme, and neither is this particular encoding scheme. Instead, turbo codes are more popular. Unfortunately, we didn't really get to look into turbo codes.

**References**

**2006, 1st January. "What Error-Control Coding Can Do."** ***Legislation Content from Urgent Communications*****, urgentcomm.com/2006/01/01/what-error-control-coding-can-do/.**

This source was less technical and more explained the importance of error control codes.

**Adams, Sarah Spence. Introduction to Algebraic Coding Theory . 2008.**
We used this source in order to get a good grasp on error control codes in general, especially some of the “simpler” codes as well as the terminology that often surrounds error control codes.

**Balachandran, Hrudya. “Fano Algorithm.” *LinkedIn SlideShare*, 25 Jan. 2014, www.slideshare.net/hrudyabalachandran/fano-algorithm-30418265.**
This powerpoint was less technical in nature but provided a lot more intuition about how Fano and Stack algorithms work.

**“Error Detection and Correction.” *Wikipedia*, Wikimedia Foundation, 13 Dec. 2018, en.wikipedia.org/wiki/Error_detection_and_correction.**
Papers are really great, but sometimes one has to go ‘back to the basics,’ so to speak. We used it just for general understanding and overview at the beginning of out project.

**Gupta, Kapil, et al. “A Comparative Study of Viterbi and Fano Decoding Algorithm for Convolution Codes .” *Stability of Plane Couette Flow of Carreau Fluids Past a Deformable Solid at Arbitrary Reynolds Numbers: Physics of Fluids: Vol 30, No 7*, AIP Publishing LLC, aip.scitation.org/doi/pdf/10.1063/1.3526231.**
This paper was really great in terms of laying out the differences between Viterbi and Sequential Decoding.

**Han, Yunghsiang S. *Sequential Decoding of Binary Convolutional Codes*. Graduate Institute of Communication, National Taipei University, web.ntpu.edu.tw/~yshan/T_Sequential_decoding_convolutional_codes.pdf.**
This was a powerpoint we used to gain more knowledge about Fano’s algorithm and sequential decoding in general. It was very helpful for the unsuccessful implementation of Fano’s algorithm. (Unsuccessful due to our lack of knowledge, source was still great.)

**HKN, UConn. “Digital Communications: Viterbi Algorithm.” *YouTube*, YouTube, 3 Dec. 2017, www.youtube.com/watch?v=dKIf6mQUfnY.**
This video provided a really great walk through of how Viterbti decoding works through a trellis.

**Illinois, Computational Linguistics @. “Stack Decoding for Statistical Phrase-Based Machine Translation.” *YouTube*, YouTube, 28 Apr. 2017, www.youtube.com/watch?v=oWVmphEaHZI.**
This video provided a really great walk-through of how a machine-based translation would work using the stack algorithm.

**Larsen, K. “Short Convolutional Codes with Maximal Free Distance for Rates 1/2, 1/3, and 1/4 (Corresp.).” *IEEE Transactions on Information Theory*, vol. 19, no. 3, 1973, pp. 371–372., doi:10.1109/tit.1973.1055014.**

This publication gives examples of convolutional codes of different rates and k values with maximum free distance, and thus best suited for correcting a high number of errors. We used their example codes for rates ½ and window widths 3-7 for testing our encoding and decoding software.

**“Lecture Notes - Convolutional Coding.” *Web.mit.edu*, Massachusetts Institute of Technology, http://web.mit.edu/6.02/www/f2010/handouts/lectures/L8.pdf.**
These lecture notes were very helpful to give us knowledge about how convolutional codes/encoding works, as well introduce decoding and briefly discuss Viterbi.

**“Lecture Notes - Viterbi Decoding of Convolutional Codes.” *Web.mit.edu*, Massachusetts Institute of Technology, web.mit.edu/6.02/www/f2010/handouts/lectures/L9.pdf.**
These lecture notes from the MIT website gave us the necessary information about Viterbi encoding to design our software implementation.

**Li, Chih-Peng. *Convolutional Decoder and Its Applications*. MITs Lab: Intitute of Communication Technology, apwcs2014.nsysu.edu.tw/course/pdfdownload/9221/CC-ConvolutionalCode(III).pdf.**
This powerpoint talked about convolutional codes in general but we mostly used it for its information of the Fano and Stack algorithm.