Turbo codes and Viterbi algorithm

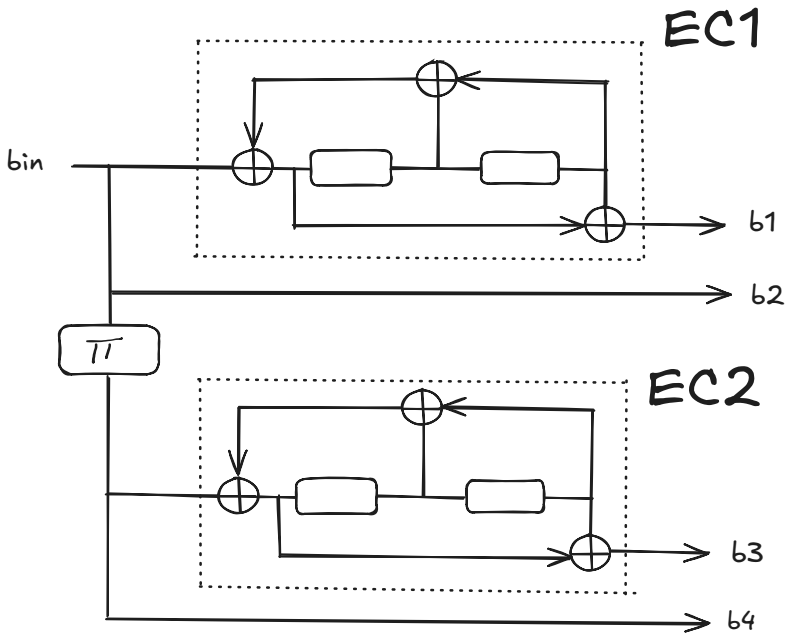
2025-05-23 13:34

Tags:

This project focused only on turbo codes and their simplest implementation - hard decoding. So no modulation, AWGN etc. Noisy channel is just a `bit_flip(message, p)` method, which flips each bit in message with probability p - 1. This text describes the project structure and the idea behind it. Also I mentioned some pitfalls you might encounter along the way. This note may be chaotic, so feel free to skip any part you want.

Encoding

This turbo coder consists of two connected RCS (recursive systematic convolutional) encoders. Both are connected via an interleaver:

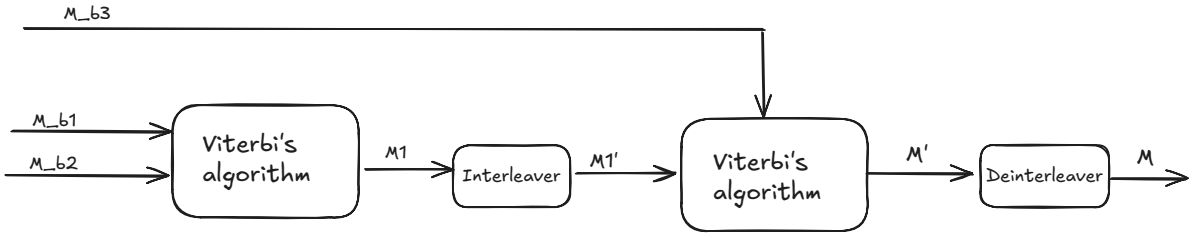


This entire encoder has $R = 1/3$, because the systematic bit $b4$ is just a permutation of $b2$, so ultimately only three bits will be sent. The message is structured as follows $b1, b2, b3 \dots$.

Interleaver takes the plain text message to be sent assembles it in a $(message.size/2) \times 2$ matrix, by filling it row by row, sort of like following a snake trail. Then reads this matrix column by column and there you end up with an interleaved message. This interleaved message is passed to `EC2` which is identical to `EC1` (in code for convenience I made it, so that EC2 doesn't return the systematic bit at all, but that's not necessary).

Decoding

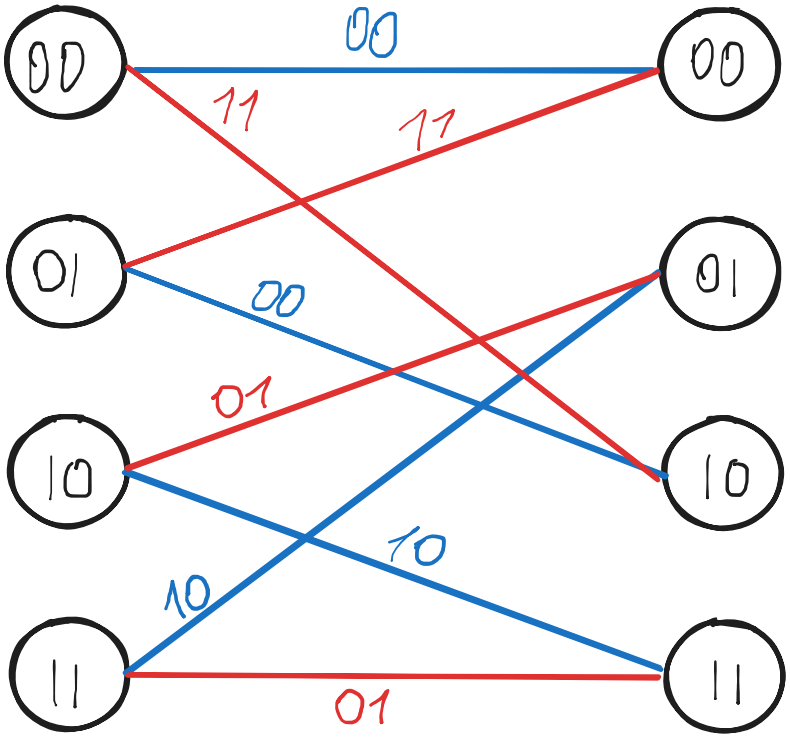
Decoder receives the encoded message in the format specified above. Here's a diagram that illustrates the decoder:



1. Pick out bits generated by `EC1` (marked as M_b1 and M_b2).
2. Decode them using Viterbi's algorithm for the shortest path in the trellis (M1).
3. Take this decoded message and interleave it (M1'). This interleaved message will serve as systematic bits for the interleaved encoded message coming from `EC2` (M_b3).
4. Assemble those new systematic bits (M1') with the encoded message, which came from `EC2` (M_b3) and decode using Viterbi's algorithm for the shortest path in the trellis to get M'.
5. Deinterleave M' to get final message M.

You could stop at step 2, but you'd end up with a much higher BER and it would be just a convolutional code - not a turbo code.

To do this in code, you'll need to draw a trellis:



Blue lines are for input 0 and red for 1. Please be cautious, when drawing your trellis. I spent way too long figuring out, what's wrong with my decoder, when it was simply wrong state transitions - remember about feedback!

Implementing Viterbi's shortest path algorithm was hard. It felt like a leetcode problem, but was actually the most rewarding part of the project. I referenced a nice source, which is not turbo codes related per se, but you can learn more about this algorithm.

Viterbi's algorithm

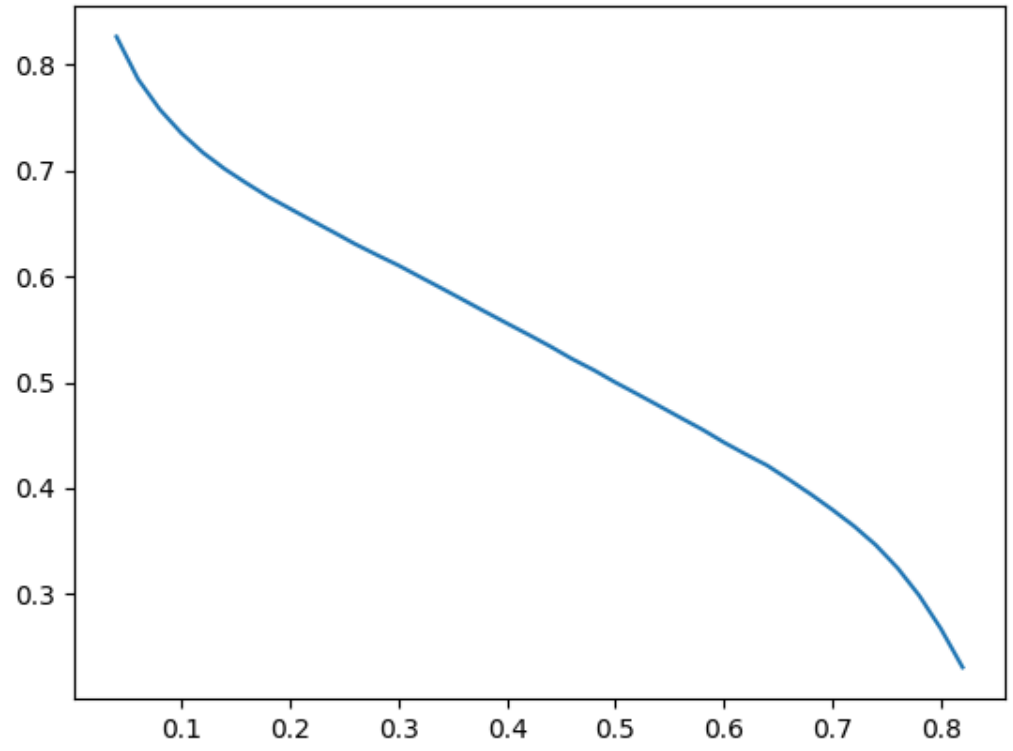
The big idea is finding the shortest path efficiently through the trellis. It's not an issue to come up with an algorithm that finds the shortest path - consider all paths. The key is to do it efficiently and that's what Viterbi did.

Usefulness of this algorithm depends on, what we can model using the trellis. In fact trellis is deeper than you think! Trellis is a visualisation of a Hidden Markov Model (HMM). One great application is speech tagging. It's about helping the computer understand a sentence. Sort of. Say some sentence has a word "watch". It's impossible to know whether it's a noun or a verb, once you get the entire sentence, you don't have to think twice to tell which part of speech it is. This is precisely modelled using a trellis, but instead of states (hidden states), you have parts of speech, path metrics are probabilities and observable states are words in a sentence (for turbo codes those are encoded bits).

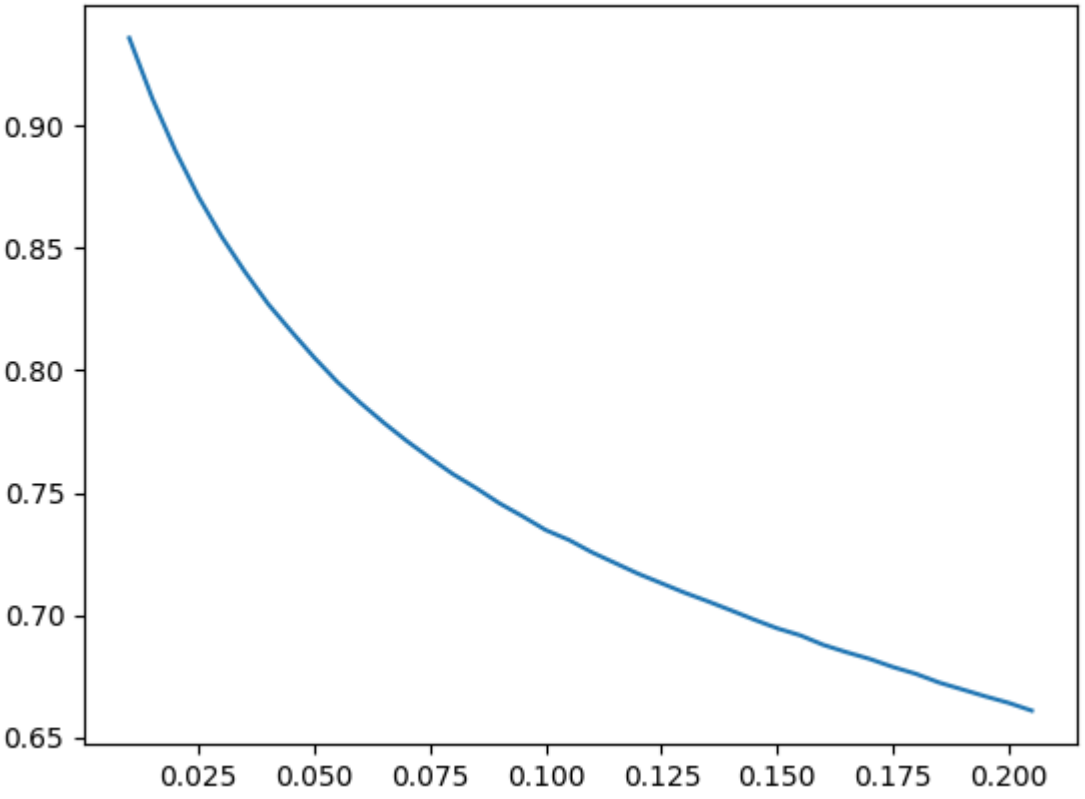
Testing

Sadly this section will be fairly short, since I haven't yet played enough with my code. One reason is that I manage time poorly and the other is that each plot takes like 2h. Of course I could make an effort to improve my code, but don't forget that turbo codes are demanding.

I tested BER for different p's:

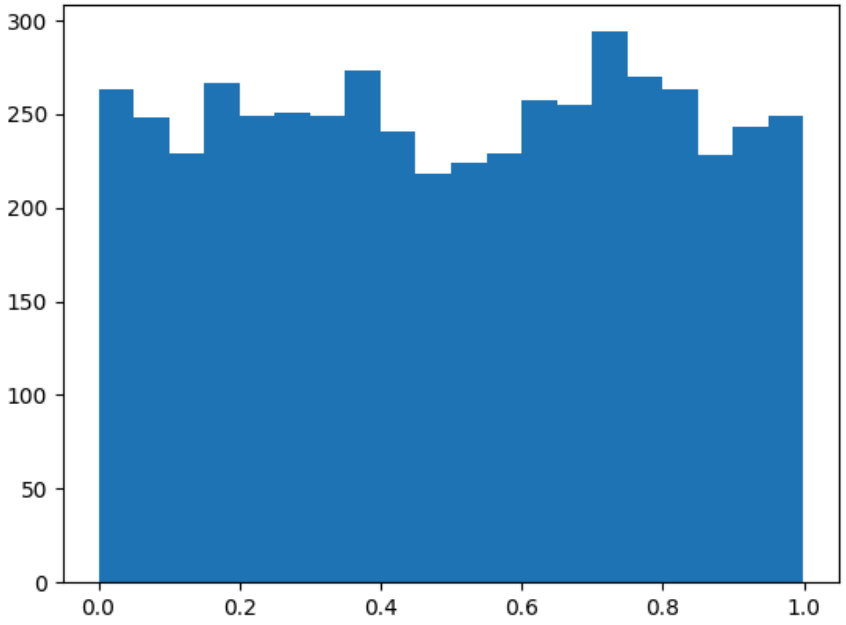


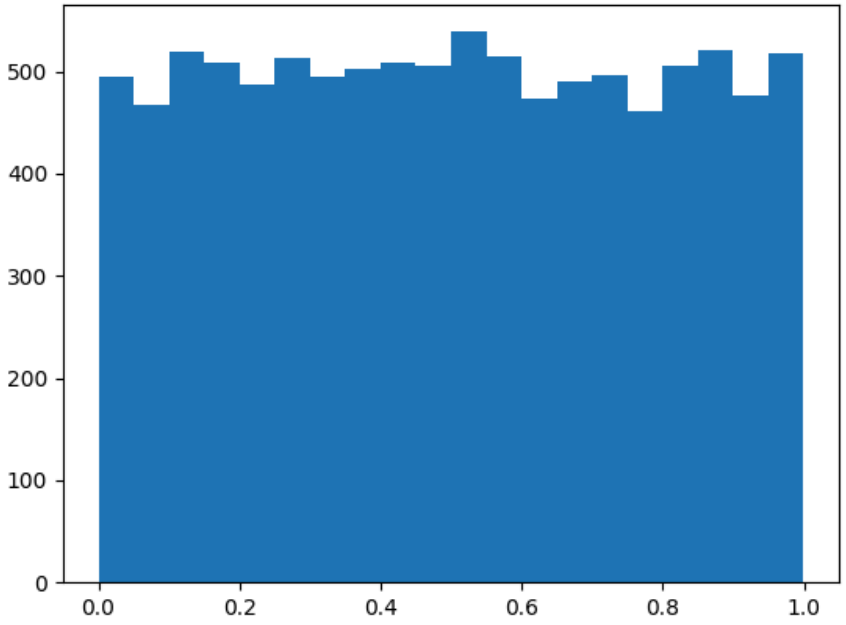
Closer look at



Graph looks more less as expected, for low p (high chance of bit flip) we have high BER and for low p low BER.

While testing I encountered a strange issue regarding the uniform distribution. In order to get a fairly uniform histogram, you have to sample a lot. A lot. Below there are two histograms: one of 5k and the other of 10k samples from $U(0,\ 1)$. As we can see 5k is far from uniform and I initially tried to do it for 200 samples!





Since decoding is very costly I initially tried to do it for smaller samples of $U(0,\ 1)$ and it backfired. Took me a minute to find this out.

References

- More complex version of turbo codes if you're feeling ambitious: https://daulpav.id/2018/10/16/turbo-codes.html
- Viterbi algorithm in python: https://pieriantraining.com/viterbi-algorithm-implementation-in-python-a-practical-guide/