Investigation into the impact on performance created by the components of turbo error correcting codes

Research Question: To what extent do internal components impact the performance of parallel concatenated convolutional codes?

Computer Science Extended Essay

Word count: 3986

Table of Contents

Introduction

In the modern age, data transfer is not only essential, but also omnipresent. From the applications of terrestrial communications to communicating with satellites orbiting Mars, the need to exchange information is present. But providing reliable and correct data is not simple, as in reality, received signals do not only consist of meaningful information, but also of noise. One way to counteract this property of the environment would be to increase the signal strength. This is not always possible or desirable, due to power consumption requirements or compact designs of transmitters. In order to overcome those limitations, the first error correction codes (ECC) were created. These codes are special methods which aim to protect transmitted information by adding redundant data. Essentially, ECCs transform any message in a way which makes it less prone to being received as incorrect. Even if an error is introduced in the message through the means of noise interference, the receiver will be able to detect the error and correct it, provided that the magnitude error is small enough, as error correction codes do not have infinite correction capabilities.

Turbo codes are a class of ECCs, which have combined properties of block codes, which encode blocks of data of fixed size, and convolutional codes, which can encode infinite sequences. They possess very high performance, unmatched by simpler error correcting schemes and very close to Shannon's limit, the theoretical upper boundary of nearly-errorless transmission (Berrou et al.). The main feature in which they differ from other codes is the fact that they have two encoders and decoders corresponding to the decoders. During decoding, the output data from one decoder is passed to the second decoder, which outputs data which is fed back to the first decoder. The feedback feature of the turbo decoder is the namesake of the ECC (Valenti 2).

Methodology of testing

The study aims to provide accurate comparisons between the variations of components of turbo codes, as well as other codes as a point of reference to the wider field of error correcting codes. In order to provide an accurate simulation of environmental noise, an Additive White Gaussian Noise channel was chosen. To further adopt a stricter and more realistic framework of data transmission, the Binary Phase Shift Keying modulation scheme was adopted and simulated numerically. The implementation of the noisy channel, as well as modulation is based on the paper by Sadinov et al.

Binary Phase Shift Keying (BPSK)

Binary Phase Shift Keying is a modulation scheme based on the modifications to the phase of a base signal, which produce output signal in result. In binary phase shift keying, there are two phases outlined, which are commonly denoted as 180° and 0°. Though BPSK does not have the best transmission rates, it is the most resilient of all phase shift keying modulation schemes, as it encodes only one bit with each phase change. (Mahesh Naidu et al.) In the simulations conducted for the purpose of this study, the phase shifts along with the waveforms were not simulated directly, but instead a numeric simulation scheme was used. Bits 0 and 1 were modulated as -1, and +1, respectively. This can be conveniently expressed with this equation

$$m_i = 2 * b_i - 1 \qquad \text{(Equation 1)}$$

Where m_i is the modulated output and b_i is the original bit. In order to determine whether a transmitted, modulated symbol is a 0 or a 1, the following threshold is applied:

$$y_i = \begin{cases} 1 & \text{if } m_i > 0 \\ 0 & \text{if } m_i < 0 \end{cases} \qquad \text{(Equation 2)}$$

Additive White Gaussian Noise (AWGN)

The notion of AWGN in the implementation can be broken down into three main features:

1. Additive – the noise is added to the transmitted signals, and is completely independent of them.
2. White – there are no differences in the severity of errors among different transmission frequencies.
3. Gaussian – the values of errors are dictated by the standard normal distribution with mean 0 and standard deviation dependent on the noise density.

The usage of the normal distribution is meant to mimic the natural environment, as the normal distribution is considered as a good model for many processes which occur naturally, including environmental noise. The additive aspect of AWGN tells an important fact – the interference need not always introduce errors. This is evident when considering the thresholds (Equation 2), as an example, a positive noise value will not flip the +1 symbol, and therefore will not introduce an error. Given that the noise is white, the tests do not need to take into account the frequency of the transmission. The only variable in the tests is the noise density, which is calculated from the Signal-to-Noise Ratio (SNR) in decibels, also written $\frac{E_b}{N_0}$. The tests are conducted at changing values of SNR, where the lowest value of SNR denotes the highest density of errors. The density of errors can be obtained from the SNR in the following way, assuming that the E_b is always 1.

$$N_0 = \left(10^{\frac{\mathrm{SNR}}{10}}\right)^{-1} \qquad \text{(Equation 3)}$$

Methodology of testing

Depending on the needed accuracy, the tests were conducted in different ranges of SNR. The interval between consecutive values of SNR is always 0.5 dB. A range of tests was covered, including tests which compare different possibilities of implementation of turbo codes, as well as those which compare certain performance metrics between different error correction codes.

The key metric which was measured in all tests was the Bit Error Rate, which is calculated as the proportion of the number of erroneous bits in the decoded message to the length of the original message in bits. The BER values were calculated for a randomly generated message of a given length, which was transmitted multiple times. The values available as results are averages, and were calculated by this formula

$$BER = \frac{\sum_{i=0}^{m} \frac{E_m}{N}}{m} \qquad \text{(Equation 4)}$$

where m is the number of tests, N is the length of the message, and E_m is the number of errors which occurred in a given test.

Tested codes

The tests were conducted on the following error correction codes, including their variations:

- Hamming (7, 4) block code (for context and comparison purposes)
- Reed-Solomon (15, 7) block code (for context and comparison purposes)
- Non-systematic Convolutional Codes with memory size 3 and rate 1/2
    - Gates 1, generated by $G_1 = \begin{bmatrix} 1 + D^2 + D^3 & 1 + D + D^2 \end{bmatrix}$
    - Gates 2, generated by $G_2 = \begin{bmatrix} 1 + D + D^2 + D^3 & 1 + D + D^3 \end{bmatrix}$
- Recursive Systematic Convolutional Codes (RSC) with memory size 3 and rate 1/2
    - Gates 1, generated by $G_1 = \begin{bmatrix} 1 & \frac{1+D+D^2}{1+D^2+D^3} \end{bmatrix}$
    - Gates 2, generated by $G_2 = \begin{bmatrix} 1 & \frac{1+D+D^3}{1+D+D^2+D^3} \end{bmatrix}$

- Turbo codes (Parallel Concatenated Convolutional Codes) with rate ½ and interleaver length 1000
    - RSC Gates 1 as the constituent encoder
    - RSC Gates 2 as the constituent encoder

Both variations of the turbo code were also tested under three different interleaving methods: the S-Random interleaver, a dummy “switch” interleaver designed for the purpose of the tests, as well as no interleaving at all. It is also important to note that the turbo codes have been punctured – only half of the redundant bits were transmitted in order to increase the rate from 1/3 to ½. This rate of the code is similar to the other codes, which provides a more appropriate point of comparison, as convolutional codes which transmit more redundant data have better performance (Saini, Sharma)

Simulation results

Tabular presentation of data

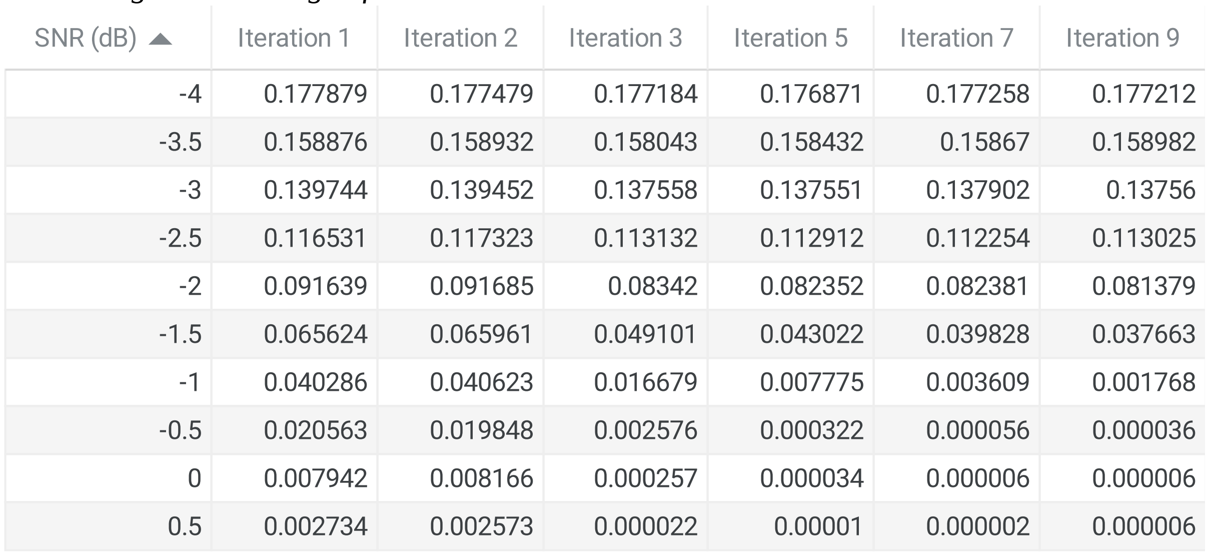
Table 1: BER simulation results for Nonsystematic Convolutional Codes and their equivalent Recursive Systematic Convolutional Codes in the range of 0 dB to 7.5 dB

SNR (dB)	NCC Gates 1	RSC Gates 1	NCC Gates 2	RSC Gates 2
0	0.049744	0.04313	0.032969	0.033105
0.5	0.032557	0.028816	0.019553	0.020782
1	0.019696	0.018345	0.010805	0.01186
1.5	0.011406	0.01114	0.00523	0.006112
2	0.006176	0.00583	0.002723	0.00318
2.5	0.002998	0.003136	0.001172	0.001543
3	0.001467	0.001544	0.000503	0.000749
3.5	0.000629	0.000654	0.000264	0.000291
4	0.00023	0.000297	0.000099	0.000128
4.5	0.000093	0.000111	0.000045	0.000044
5	0.000043	0.000026	0.000019	0.000014
5.5	0.000015	0	0.000014	0
6	0	0	0	0
6.5	0	0	0	0
7	0	0	0	0
7.5	0	0	0	0

Table 2: Simulation results for comparison of hard-decision Viterbi decoding and soft-decision BCJR decoding in the range of -4 dB to 0.5 dB

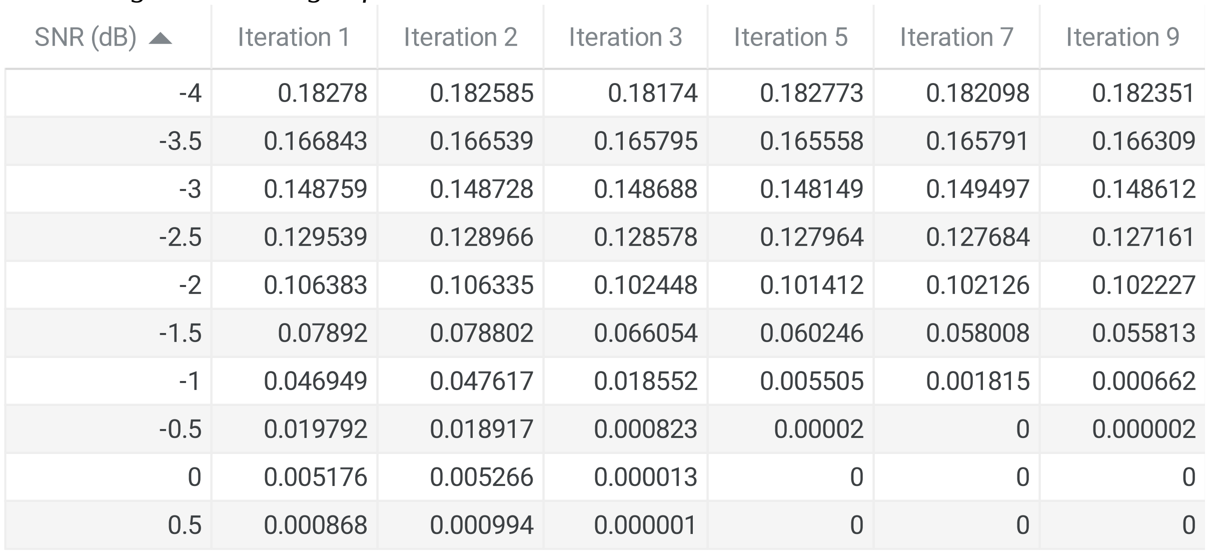
SNR (dB)	Viterbi Gates 1	BCJR Gates 1	Viterbi Gates 2	BCJR Gates 2
-4	0.224437	0.152131	0.226792	0.165096
-3.5	0.20301	0.127081	0.202348	0.138825
-3	0.17952	0.099575	0.179003	0.109763
-2.5	0.154889	0.073727	0.152457	0.079876
-2	0.129422	0.050173	0.125117	0.052216
-1.5	0.105052	0.032351	0.098759	0.030328
-1	0.082189	0.018698	0.072222	0.015749
-0.5	0.061426	0.009957	0.050606	0.007459
0	0.042922	0.004856	0.033852	0.003295
0.5	0.028494	0.002235	0.020461	0.001309

Table 3: Simulation data gathered on different iteration levels of the turbo code based on RSC with gates 1 in range of -4 to 0.5 dB



SNR (dB) ▲	Iteration 1	Iteration 2	Iteration 3	Iteration 5	Iteration 7	Iteration 9
-4	0.177879	0.177479	0.177184	0.176871	0.177258	0.177212
-3.5	0.158876	0.158932	0.158043	0.158432	0.15867	0.158982
-3	0.139744	0.139452	0.137558	0.137551	0.137902	0.13756
-2.5	0.116531	0.117323	0.113132	0.112912	0.112254	0.113025
-2	0.091639	0.091685	0.08342	0.082352	0.082381	0.081379
-1.5	0.065624	0.065961	0.049101	0.043022	0.039828	0.037663
-1	0.040286	0.040623	0.016679	0.007775	0.003609	0.001768
-0.5	0.020563	0.019848	0.002576	0.000322	0.000056	0.000036
0	0.007942	0.008166	0.000257	0.000034	0.000006	0.000006
0.5	0.002734	0.002573	0.000022	0.00001	0.000002	0.000006

Table 4: Simulation data gathered on different iteration levels of the turbo code based on RSC with gates 2 in range of -4 to 0.5 dB



SNR (dB) ▲	Iteration 1	Iteration 2	Iteration 3	Iteration 5	Iteration 7	Iteration 9
-4	0.18278	0.182585	0.18174	0.182773	0.182098	0.182351
-3.5	0.166843	0.166539	0.165795	0.165558	0.165791	0.166309
-3	0.148759	0.148728	0.148688	0.148149	0.149497	0.148612
-2.5	0.129539	0.128966	0.128578	0.127964	0.127684	0.127161
-2	0.106383	0.106335	0.102448	0.101412	0.102126	0.102227
-1.5	0.07892	0.078802	0.066054	0.060246	0.058008	0.055813
-1	0.046949	0.047617	0.018552	0.005505	0.001815	0.000662
-0.5	0.019792	0.018917	0.000823	0.00002	0	0.000002
0	0.005176	0.005266	0.000013	0	0	0
0.5	0.000868	0.000994	0.000001	0	0	0

Table 5: Performance of alternatives to interleaving on PCCC based on RSC with gates 1

SNR (dB) ▲	No Interleaver	Switch Interleaver
-4	0.176568	0.177327
-3.5	0.158807	0.159281
-3	0.139474	0.140127
-2.5	0.117892	0.117926
-2	0.093103	0.094251
-1.5	0.068814	0.071159
-1	0.045612	0.048941
-0.5	0.027808	0.030078
0	0.014578	0.017056
0.5	0.006863	0.008982

Table 6: Performance of alternatives to interleaving on PCCC based on RSC with gates 2

SNR (dB) ▲	No Interleaver	Switch Interleaver
-4	0.182613	0.181891
-3.5	0.166507	0.166821
-3	0.148431	0.148932
-2.5	0.129904	0.130578
-2	0.107498	0.107647
-1.5	0.081773	0.081843
-1	0.054026	0.055653
-0.5	0.030694	0.031435
0	0.014909	0.015314
0.5	0.006066	0.006573

Table 7: Performance of other error correcting codes and control tests of the BPSK channel in the range of -4 to -0.5 dB

SNR (dB) ▲	BPSK Channel	Hamming (7, 4)	Reed-Solomon (7, 3)
-4	0.1863184	0.1773632	0.1766119766
-3.5	0.1721353	0.1588688	0.1580957581
-3	0.158393	0.1402538	0.1386258386
-2.5	0.1444111	0.1217464	0.118987419
-2	0.1306903	0.1040028	0.09972499972
-1.5	0.1169326	0.0868855	0.08154718155
-1	0.1037215	0.0711498	0.06425546426
-0.5	0.0908691	0.0566948	0.04903524904
0	0.0785616	0.0441075	0.03570843571
0.5	0.067092	0.0332586	0.0248018248

Graphical presentation of data

Please note that the vertical axes on the graphs are presented in logarithmic scale. Any results in the order of magnitude of 10^{-6} were deemed as inconclusive and truncated,resulting in lines on the graph which appear "cut off".

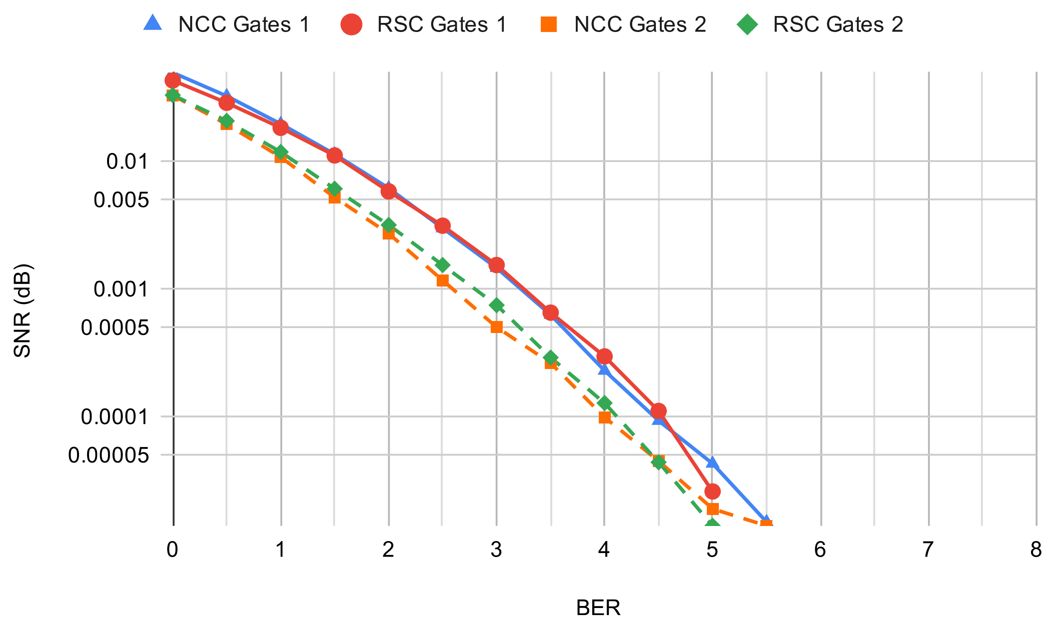


Figure 1: Performance of nonsystematic convolutional codes (NCC) and recursive systematic convolutional codes (RSC) with both gates 1 and 2

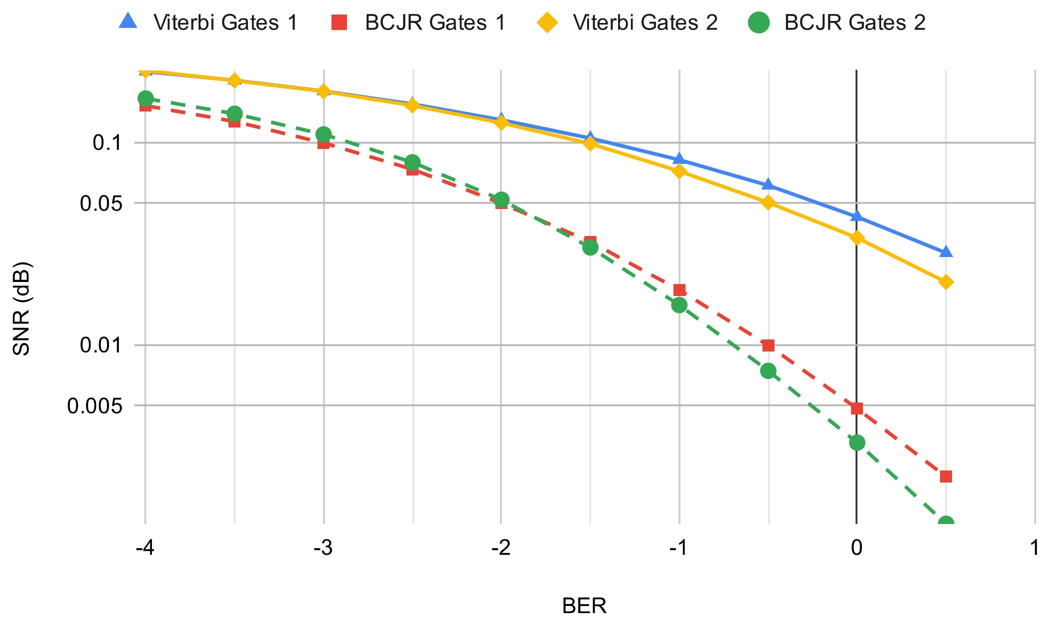


Figure 2: Performance of BCJR and Viterbi decoding on RSC with gates 1 and 2

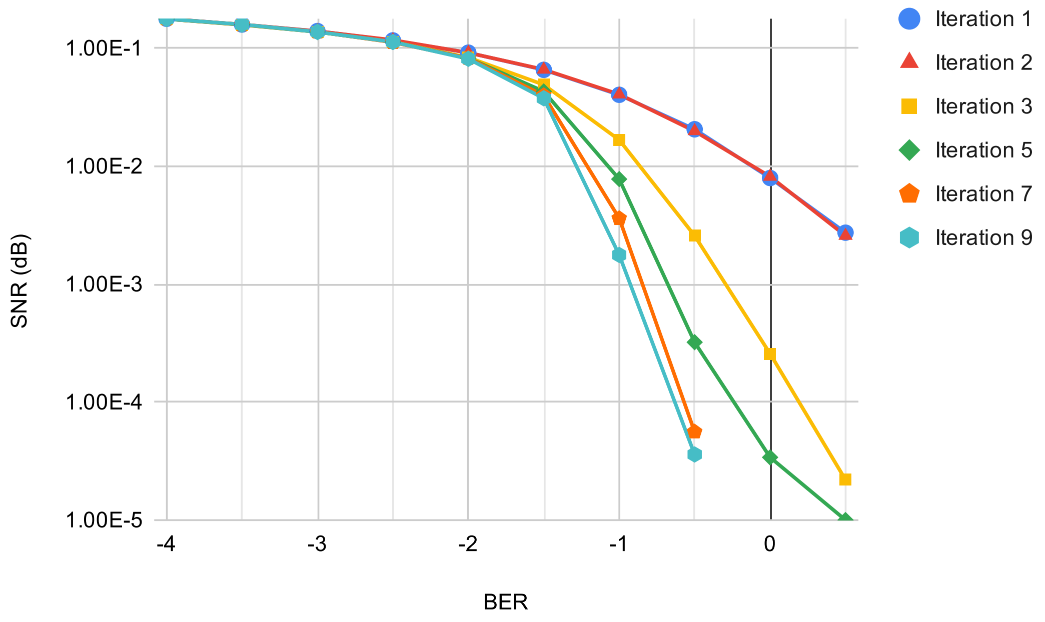


Figure 3: Performance of different levels of iteration of the turbo code based on RSC with gates 1

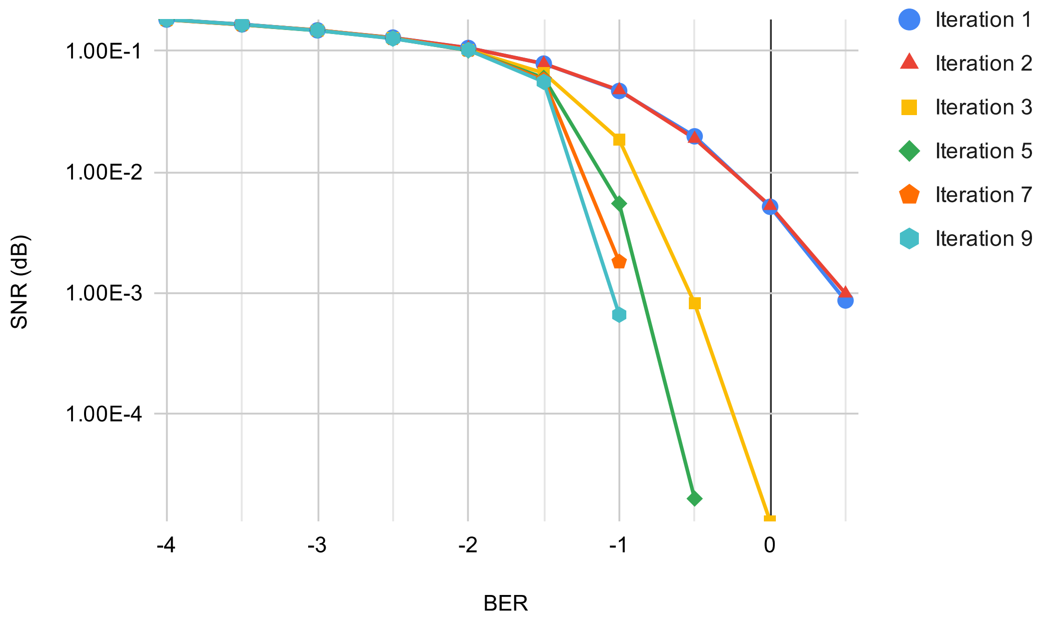


Figure 4: Performance of different levels of iteration of the turbo code based on RSC with gates 2

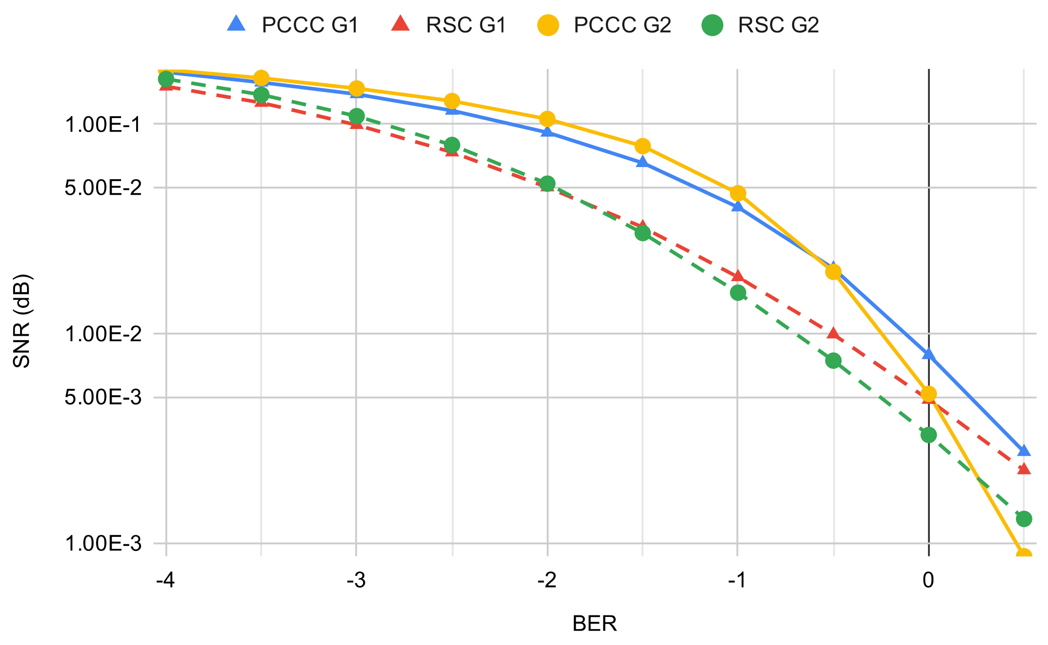


Figure 5: Comparison of the turbo codes (PCCC) compared with their constituent RSC codes

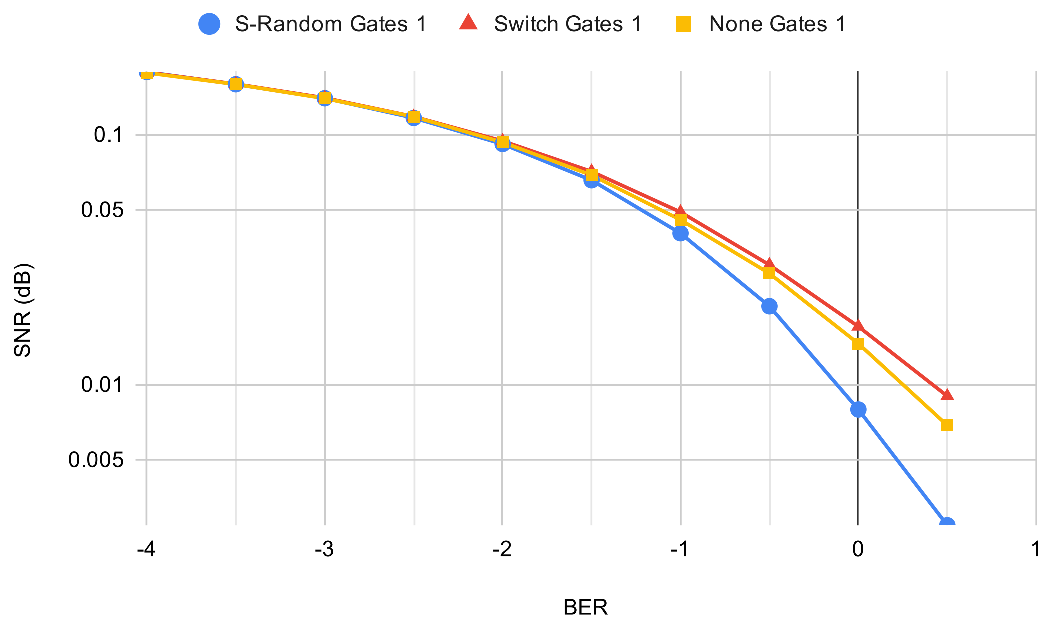


Figure 6: Comparison of interleaving methods on PCCC with gates 1 and one turbo iteration

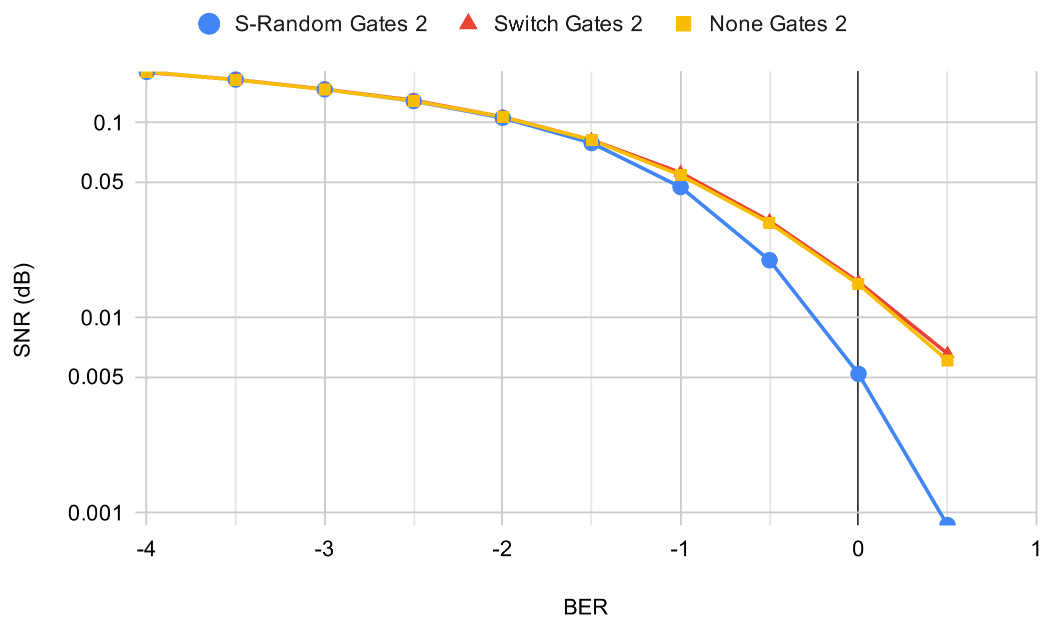


Figure 7: Comparison of interleaving methods on PCCC with gates 2 and one turbo iteration

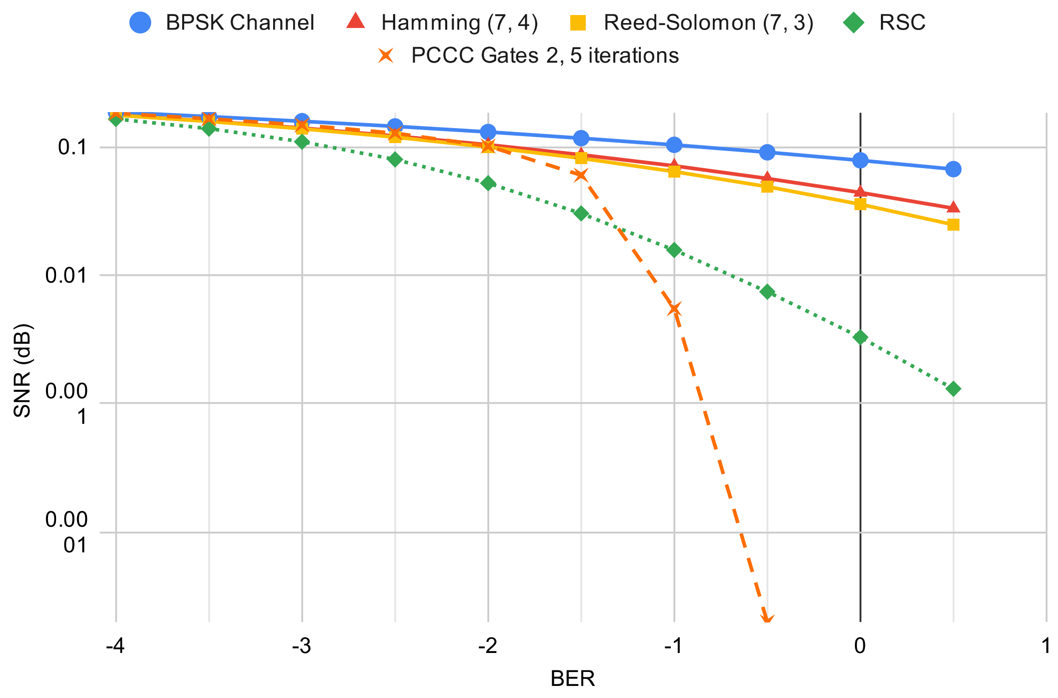


Figure 8: Performance of selected block codes compared to PCCC with 5 turbo iterations and RSC, both featuring gates 2

Recursive Systematic Convolutional Codes

The first coding techniques which were based on the idea of convolution were described by Elias in 1955 as convolutional codes (Valenti 1).

Structure of RSC

Any convolutional code must have an internal memory, a set of registers each holding exactly one bit. During encoding, the bits of the message are shifted through the registers and a set of outputs is generated by using a set of XOR functions on the input data and memory.

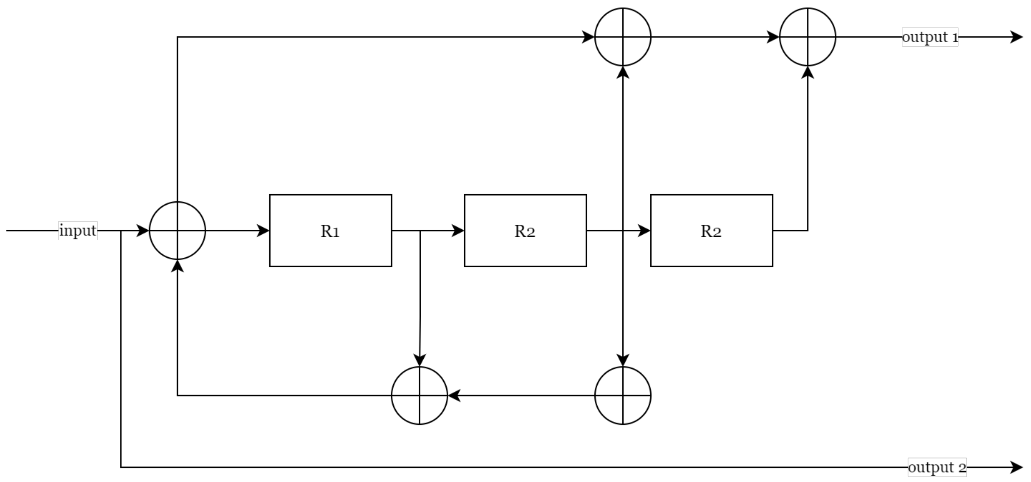


Illustration 1: Circuit diagram of RSC with gates 1

As it is visible in Illustration 1, the structure of the RSC includes sending the unmodified input bit along the calculated bit, hence the systematic features. These features are crucial when building a turbo code, which uses the systematic bits as a point of reference between its two constituent codes. The structure involves feedback, which modifies the input bit before it is fed into memory. In effect, the bits in memory are recursive, as they depend on previous bits.

The main advantages of the RSC structure is that it provides an understandable and simple implementation, which provides a solid point of entry into building a more robust system.

Trellis diagram

The convolutional encoder is a finite state machine, meaning that the output is only dependent on some discrete internal state – the memory (Johannesson, Zigangirov). The trellis diagram is a graph containing the states of the convolutional codes as nodes, the possible transitions as edges and the output during those transitions as weights. The trellis is used extensively during decoding, as the maximum likelihood sequence turns out to be the path through the trellis which has the lowest weight, which means that it differs the least from the received sequence (Abubeker).

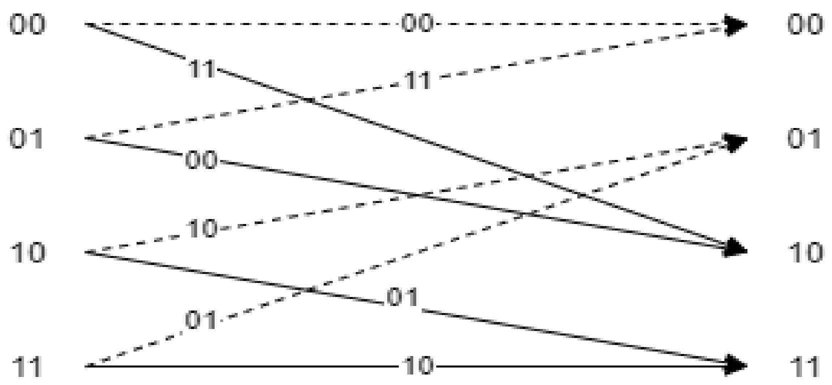


Illustration 2: An example trellis diagram. The dashed lines denote input 0 and the solid lines denote input 1.

But there is an important question – is there a performance penalty for replacing one non-systematic bit with a systematic bit? From the data in Table 1, the average difference between an NCC and its equivalent RSC in their bit error probabilities is about 0.07 percentage points for gates 1 and 0.02 percentage points for gates 2, which proves that they have very similar performance. This is due to the properties of the generator matrices, as each NCC has an equivalent RSC, which should yield similar performance (Johannesson, Zigangirov).

$$\begin{aligned} G_{NCC} &= \begin{bmatrix} 1+D^2+D^3 & 1+D+D^2 \end{bmatrix} \\ G_{RSC} &= \begin{bmatrix} 1 & \frac{1+D+D^2}{1+D^2+D^3} \end{bmatrix} \end{aligned} \qquad \text{(Equation 5)}$$

Equation 5 depicts a NCC and its equivalent RSC code in their generator matrix forms. The number 1 denotes the input bit, and D is the delay operator, such that D^n is the bit at time $t-1$, which is stored in the memory. The RSC is constructed by dividing both of the output generators by one of them.

The simple construction and systematic properties, as well as no performance penalty from sending the systematic bit makes RSC codes an appropriate design choice for constituent codes of turbo codes.

Impact of the generator on performance

The two codes utilized in this study have different generator matrices, which means the structure of the XOR gates in the code is different. Due to the structure of convolutional codes, the choice of the gates determines the resulting codewords (output). Because the convolutional codes create an output dependent on the previous inputs and the current input, it is simple to create a bad code, which does not make use of much of the information stored in memory, e.g. the gates are dependent on only one memory bit. But to what extent is changing the structure useful?

As demonstrated in Figure 1, the difference in performance between codes which have different gates is clearly distinct, which suggests that the gates have a significant impact on the performance. From Table 1, the average difference between RSC with gates 1 and RSC with gates 2 is about 0.3 percentage points, which is an order of magnitude higher than e.g. the difference between equivalent codes. This information is very useful, as it may indicate that the choice of the constituent code may impact the overall performance of the turbo code.

Soft decoding – BCJR algorithm

In order to perform the iteration process, which is crucial in the case of turbo decoding, the constituent decoders of the turbo code need to assess soft input and produce soft output (Berrou et al.). The algorithm which has those properties is the BCJR algorithm, originally designed by Bahl, Cocke, Jelinek and Raviv. Instead of calculating the most probable path through the trellis (like the Viterbi algorithm), the BCJR algorithm calculates the probabilities of a given bit being 1 or 0 on a bit-by-bit basis based on the received sequence and knowing the possible states and transitions from the trellis of the code (Huffman, Pless). It is also important to note that the implementation of the BCJR algorithm presented in this study is in the logarithmic domain, in order to simplify calculations.

Path metric

The BCJR algorithm takes a more complex approach than maximum likelihood decoding mentioned in section. Instead of the weight, the BCJR algorithm calculates a path metric called the Euclidean distance, as described by Abrantes. Because the output provided to the decoder is not quantized, i.e. it can take real values, the notion of weight is not applicable. Hence, the Euclidean distance γ of a branch between state s and its previous state s' at time k is calculated as such:

$$\Gamma(s', s) = \frac{u_k L(u_k)}{2} + \frac{L_c}{2}(p_1 r_1 + p_2 r_2)$$

(Equation 6, Abrantes, adapted for the purposes of the code used) where $L(u_k)$ is the a priori probability (the assumed beforehand probability of a symbol being 1 or 0), u_k is the bit associated with the branch, L_c is the channel reliability measure (a constant), p_1, p_2 are the prototype output bits (saved in the trellis and quantized) and r_1, r_2 are the received bits.

Due to this process, the BCJR decoding method utilizes the data about the magnitude of errors, which is discarded in the case of hard-decision decoding.

Forward and backward metrics

In contrast to the path metrics, the forward and backward metrics are associated with states on the trellis diagram. They can be calculated recursively, and represent probabilities the encoder being in a given state given the previous or the following sequence of received symbols.

The forward alpha metric is connected with the probability of the encoder being in state S at time k given the previous sequence, and is calculated as:

$$\mathrm{A}_k(s) = max^*_{s'}[A_{k-1}(s') + \Gamma_k(s', s)]$$

(Equation 7, Abrantes)

where

$$\max{}^*(a, b) = max(a, b) + ln(1 + e^{-|a-b|})$$

(Equation 8, Abrantes)

The max* operation is a result of the logarithmic implementation of the algorithm. As seen in Equation 7, the alpha metric is recursively dependent on the previous alpha metrics, as the max operations is executed over the two possible previous states s’. It is also important to note the purpose of the path metric. The probability of the transition occurring influences the probability of the encoder being in the next state. The base case for the recursion is that the all-zero state at time 0 has probability 1, while all other states at time 0 are impossible, as the convolutional codes were implemented to always reset to that state before encoding.

The beta backward metric is similarly calculated by recursion. It represents the probability of an encoder's state being achieved at time k given the following received sequence.

$$\beta_{k-1}(s') = max_s^*[\beta_k(s) + \Gamma_k(s', s)]$$

(Equation 9, Abrantes)

Equation 9 shows that beta metrics are dependent on the next states and the branch metric. By combining this definition with the implementation of the codes, a problem of the base case for recursion arises. How can the final state of the encoder be known, if the original message is not known before decoding?

In the implementation, this problem was solved by assigning each of the final nodes (the base states for recursion) with an equal probability, e.g. 1/8 for the code with memory 3 and 8 possible states.

The log likelihood ratio

The end result of BCJR decoding, which is the LLR is computed as a logarithm of the ratio of probabilities of the bit being one over the probabilities of the bit being zero. This calculation is represented as:

$$\begin{aligned} LLR(u_k) &= max_{u=1}^*[A_{k-1}(s') + \Gamma_k(s', s) + \beta_k(s)] \\ &\quad - max_{u=0}^*[A_{k-1}(s') + \Gamma_k(s', s) + \beta_k(s)] \end{aligned}$$

(Equation 10, Abrantes)

Due to the logarithmic implementation, the subtraction replaces taking the ratio of two values. As the transmitted bits are associated with the branches, the LLR calculation takes into account all branches with an associated bit, their branch metrics, the alphas on the "past state" node as well as the betas on the "future state" node. A positive LLR indicates that the decoded bit is a 1, while a negative value indicates a 0. The absolute value of the LLR indicates the level of confidence in this result, and is critical when performing iterative decoding.

Impact soft BCJR decoding

The implementation and the specific features of the BCJR algorithm allow it to achieve a greater performance than maximum likelihood Viterbi decoding. Because BCJR utilizes much more information, its performance is higher – the RSCs decoded with BCJR have an average of 6 pp of difference on both gates (Figure 2, Table 2), which is a very significant performance boost. The possible reasons for this increase is usage of data which is normally discarded by Viterbi decoding.

Impact of the generator in BCJR

It is also important to consider the impact of the choice of the generator matrix in the case of BCJR decoding. The impact on performance on basic hard-decision decoding has been evaluated in the section Impact of the generator on performance, but does the relative performance change for BCJR decoding?

The collected data suggests that the relative performance is similar for both decoding algorithms. From Table 2 and Figure 2, gates 1 are slightly better at high levels of noise for BCJR decoding and similar for Viterbi decoding, but then their performance starts to decrease in favour of gates 2, which outperform the former at lower noise levels. The collected data not only tells that the choice of the generator is crucial, but that the relative performance of two codes is independent of the decoding method.

Furthermore, the choice of the generator is crucial for designing turbo codes. Figure 5 presents an interesting result of the simulations – the turbo codes have the feature of "inheriting" the properties of their constituent encoders. The PCCC based on gates 1 has slightly higher performance on high level of noises, but lower on lower noise levels. On the other hand, the PCCC based on gates 2 has opposite features. Summarizing, their relative performance is similar to their constituent encoders, which means that the choice of the constituent codes should be an important step in the design of a turbo code.

Turbo decoding

Structure of turbo codes

That means turbo codes consist of two constituent convolutional encoders, which then have their outputs concatenated. In fact, the convolutional codes are identical, and not all of the outputs of the constituent decoders is concatenated – the systematic bits are the same, which means that they only need to be transmitted once. (Berrou et al.)

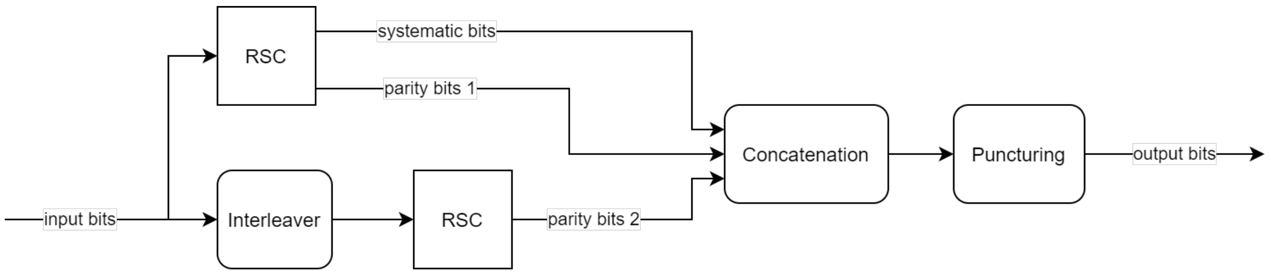


Illustration 3: General structure of the turbo encoder

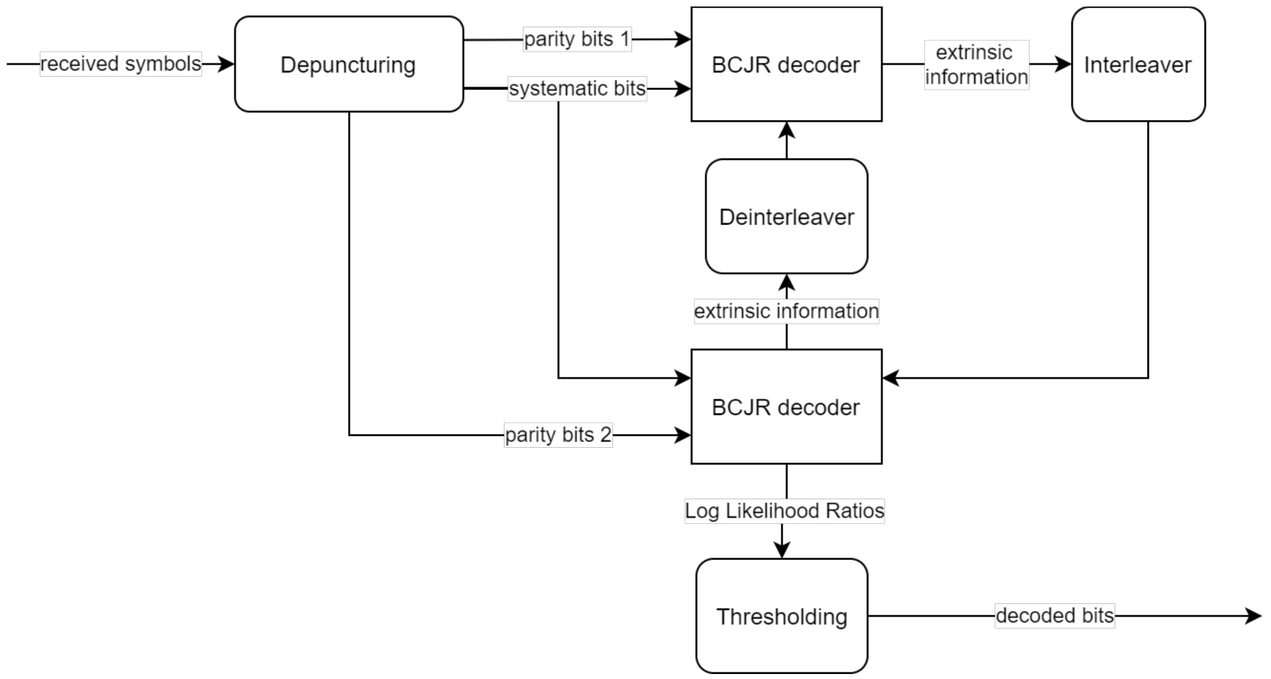


Illustration 4: General structure of the turbo decoder

In the implementation, the concatenated redundant output is also punctured – the transmitted parity bit alternates between parity bit 1 and 2.

Interleaving

The presence of the interleaver in the structure means that the input bits are rearranged in order before being passed to one of the encoders. The purpose of this process is to break the correlation between the output of the output sequences of the RSC codes (Sahnoune, Berkani). In essence, this allows the turbo code to be resilient to bursts of errors. Another important change is the fact that the presence of the interleaver makes the turbo code a block code, as a sequence of fixed length (interleaver length) needs to be present before encoding After deinterleaving, if there were any bursts of errors, they will be spread apart and they stop being a significant threat to the decoding process. Recalling Equation 7, Equation 8 it can be concluded that significant burst errors may affect the entire decoding process, especially if they are close together, as the forward and backward metrics are dependent on the closest branch metrics, which will in turn be affected by errors.

Impact of interleaving

In the simulations, a random interleaver was implemented, which is said to provide good performance due to its pseudo-random generation (Yanchun Shen et al.). In order to support a hypothesis that random interleavers should be better than non-random ones, a dummy “switch” interleaver was introduced, which switches pairs of bits in the sequence.

The results of the simulation support the evaluation – the interleaver provides a significant improvement in code performance. The dummy interleaver also provided data in support of the stated hypothesis.

From Table 5, Table 6, the differences between the interleaving methods are less significant until around -1 dB, where the performances start to clearly diverge. The average difference amounts to 0.3 pp and 0.4 pp for generators 1 and 2 respectively. An interesting result in case of the PCCC with gates 1, as it yielded weaker performance than no interleaving at all.

As to the reason why the difference is negligible between at high noise levels, one possible explanation is simple – with a big enough amount of errors, no matter how much the bits are arranged, they can form bursts.

Iterative decoding

The main feature of turbo codes is the ability to perform iterations of the decoded data. Illustration 4 shows the feedback structure of the turbo decoder. As shown in the illustration, the decoders do not exchange LLR values, but instead exchange extrinsic information, which is calculated from the LLR:

$$\mathrm{L}_e(u_k) = LLR(u_k) - L_c y_k - L(u_k)$$

(Equation 11, Abrantes)

where L_e is the obtained extrinsic information, L_c is the channel reliability factor, y_k is the received systematic bit and $L(u_k)$ is the a priori probability used in decoding.

The decoders use the extrinsic information by substituting it for the a priori probability of the bit. That is why the new extrinsic information is obtain by subtracting both the received bit value and the a priori probability, as they are redundant and do not introduce any new information in the iteration.

Impact of iterative decoding

The iteration counts of both PCCC with gates 1 and gates 2 were compared, and proved that the amount of iterations is the most significant factor when it comes to impact on performance. Even though the difference between the first two iterations is negligible and amounts to only 0.03 pp. on average for both generators, a great improvement on performance occurs with iteration 3, where the average difference is 0.84 pp for G1 and

0.73 pp for G2. An even greater improvement is observed for iteration 9, which amounts to 1.14 pp and 1.02 pp for G1 and G2 respectively. An important notion can also be concluded from the results, which prove that infinitely good performance is not achievable, as six more iterations amounted only to 0.34 pp of improvement for G1 and 0.15 pp for G2.

Conclusions

This study analysed experimental results based on a simulation and investigated the possible reasons as to why the differences in performance are present, as well as the magnitude of those differences.

Concluding from the performed analysis, the factor which has the most impact on the decoding performance is the number of iterations as the average difference is the greatest of all encountered in the tests. Though it is important to note that the iteration count has a significant impact only to a certain extent – with each iteration after a certain point, the improvement relative to the base case decreases, which shows that the performance converges towards a certain limit.

The factors of choice of generator matrix and the choice of interleaver contribute a similar improvement in performance. Even though these factors do not provide as great an improvement of performance as the iteration count, their importance cannot be ignored, as the iteration count can be adjusted even during the operation of the code, but it is rather hard to exchange an interleaver or a constituent code in practice, as they are placed on integrated circuits. The constituent code needs to be chosen not only based on the comparison of the average performance, but by the specific properties, such as performance on high levels of noise, which was shown to not be necessarily proportional to the performance at lower noise levels.

As for the Interleaver, a reliable way to choose it is to generate it randomly, as demonstrated by Yanchun Shen et al. and supported by collected data.

Summarizing, the study concludes that the factors which affect the performance of a turbo code are in order:

1. Iterative decoding and exchange of extrinsic information, facilitated by soft output decoding
2. Choice of an appropriate constituent encoder with good performance
3. Choice of an interleaver which is as random as possible

With those features in mind, excellent turbo codes can be designed to drastically outperform other error correction codes, as presented in Table 7 and Figure 8.

Further areas of research

The following possible further areas of research have been outlined, basing on the conclusions reached and data collected in this study:

- Convolutional code termination
- Other decoding schemes than Viterbi or BCJR decoding, such as BEAST decoding or list decoding
- Impact of different interleaver size on the performance of the turbo codes
- Performance of turbo codes on other noise channels, such as the Binary Symmetric Channel
- Investigation into the impact of the code rate on performance
- Investigation into the impact of different scaling factors used in the implementation of the BCJR algorithm

Works cited

Matthew C. Valenti (1), *The Evolution of Error Control Coding,* (2004), accessed 3 June 2021

Matthew C. Valenti (2), *Turbo Codes and Iterative Processing,* (1998), accessed 20 September 2021

Claude Berrou, Alain Glavieux and Punya Thitimajshima*, Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes* (1993), accessed 20 September 2021

Stanimir Sadinov, P. Daneva and Kogias, Panagiotis and J. Kanev, Ovaliadis, Kyriakos, (2019) Binary phase shift keying (BPSK) simulation using matlab, ARPN Journal of Engineering and Applied Sciences, 14. 222-226, accessed 6 May 2021

G. Mahesh Naidu, O. Sudhakar and V. Sailaja, *Simulation of BPSK Modulation and Demodulation on System Generator,* International Journal of Scientific Engineering and Research (IJSER) (2013), accessed 2 June 2021

Madan Lal Saini and Dr.Vivek Kumar Sharma, *Effects of Code Rate and Constraint Length on Performance of Convolutional Code,* International Journal of Information, Communication and Computing Technology (2016), accessed 23 November 2021

S. A. Abrantes *From BCJR to turbo decoding: MAP algorithms made easier* (2004), accessed 27 August 2021

Ahmed Sahnoune and Daoud Berkani, *On the performance of chaotic interleaver for turbo codes* (2021), accessed 23 November 2021

Yanchun Shen, Guozhong Zhao, Shengbo Zhang, Yashang Li and Shuai Li, *The Design of RA Code Pseudo-random Interleaver,* Journal of Physics: Conference Series 1237 (2019), accessed 3 September 2021

K. M. Abubeker, *Maximum likelihood Decoding of Convolutional codes using Viterbi algorithm with improved error correction Capability* (2013), accessed 23 November 2021

W. Cary Huffman and Vera Pless, *Fundamentals of Error-Correcting Codes,* Cambridge University Press (2003)

Rolf Johannesson and Kamil Sh. Zigangirov. *Fundamentals of Convolutional Coding (2nd. ed.),* Wiley-IEEE Press, (2015)