Århus University
Department of Engineering

TI-INCO
Study of Decoding Performances of Convolutional codes by MATLAB Simulation

Authors:
201503242 Christoffer Clausen
201503241 Christian Siegel
201503892 Luca Spalierno
Date:
28 March 2016

Contents

1 Introduction

This document is intended to show the performances of convolutional codes through Matlab simulation. The following section describes the given convolutional codes, explaining their related characteristics.
The Matlab implementation includes the implementation of an end-to-end digital communication system and the simulation scenarios designed by ourself and shown in section 3.
Lastly, section 4 (Discussion) describes an analysis about the system's performances and evaluation of the results and section 5 (Conclusion) contains final remarks.

2 Given Convolutional codes

The simulation involved three convolutional codes, specifying the memory order and the generator sequences, as shown in Table 2.1.

Convolutional Code	Memory Order (K)	Constraint Lenght (K+1)	Generator Sequences	n	k	Coding Rate (r)
C_{conv1}	6	7	$g^{(1)} = (1,1,1,1,0,0,1)$ $g^{(2)} = [1,0,1,1,0,1,1]$	2	1	1/2
C_{conv2}	3	4	$g^{(1)} = (1,0,1,1)$ $g^{(2)} = (1,1,0,1)$ $g^{(3)} = (1,1,1,1)$	3	1	1/3
C_{conv3}	2	3	$g^{(1)} = (1,0,1)$ $g^{(2)} = (1,1,1)$ $g^{(3)} = (1,1,1)$ $g^{(4)} = (1,1,1)$	4	1	1/4

Table 2.1: Given convolutional codes and their properties

These codes differ for both memory order and number of generator sequences. All of them have the same free distance 10, hence they also have the same error correcting capability ($t = \left\lfloor \frac{d-1}{2} \right\rfloor = 4$).

Considering the higher number of generator sequences, we expect more robustness to a noisy channel from the second and third codes compared to C_{conv1}. On the other hand, the first code might assure more stability for short burst errors, according to his memory order.

From a performance and computational complexity point of view, a higher level of memory means a higher complexity of the convolutional decoder, and thus of the overall performance. Hence, the given codes increase in complexity from C_{conv3} to C_{conv1}.

3 Simulation

The digital communication system is simulated in MATLAB. Figure 3.1 illustrates how the random generated 10^6 bit message is passed to the convolutional encoder *convenc*(), which returns a continuous encoded sequence (see section 3.1). This sequence is then passed to different channel functions, which are further explained in section 3.2.

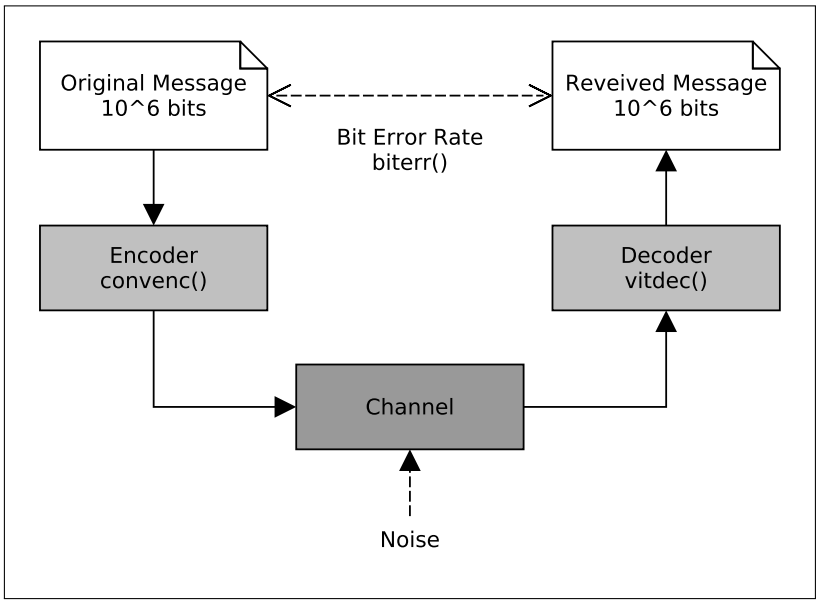

Figure 3.1: The simulated communication system

These channel functions introduce noise to the transmission of the sequence and eventually alter bits. The received encoded sequence is then passed to the *vitdec*() decoder, which applies the Viterbi algorithm to decode the received message (see section 3.3). In order to evaluate the performance of the decoding, the bit error rate (BER) of the received message is calculated using the *biterr*() function. This is further explained in section 3.4.

3.1 Encoding

The convolutional encoding is done with the MATLAB function *convenc*(). It takes the binary message vector and a trellis structure, describing the convolutional code, as parameters. Listing 3.1 shows a representative example of a random binary message passed to *convenc*() which creates an encoded sequence according to C_{conv1}. As can be seen, the trellis structure can easily be created using the function *poly2trellis*() by passing the constraint length and the generator polynomials of the convolutional code C_{conv1}.

```
1 message = randi([0 1],10^6,1);
2 trellis = poly2trellis(7,{'1 + x + x^2 + x^3 + x^6','1 + x^2 + x^3 + x^5 + x^6'
      })
3 code = convenc(message, trellis);
```

Listing 3.1: Trellis definition and message encoding

3.2 Transmission

The transmission is simulated over an binary symmetric channel (BSC). Two different channel functions were implemented. A 'Probability Channel' and a 'Fixed Burst Channel', which both can be parameterized (see probability_channel.m and fixed_burst_channel.m).

The Probability Channel is modeled by a two state Markov chain, illustrated in Figure 3.2. Both Random State and Burst State can have a different bit error probabilities assigned with *p_rand_err* and *p_burst_err* respectively.
The probability of staying in a state can also be set for both states with *p_rand_state* and *p_burst_state* respectively.
As can be seen this channel can be used as simple BSC without burst errors by setting *p_rand_state* to 1 and then controlling the bit error probability of the channel with *p_rand_err* only.
This channel can be used to observe the performance of a code for different random error scenarios, ranging from a simple BSC with different bit error probabilities (see section 4.1) to channels with burst errors of different lengths and probabilities of occurrence (see section 4.3)

The Fixed Burst Channel simulates a channel that is error free except one burst error with a specific bit length in the middle of the transmitted bit sequence.
This channel can be used to observe the performance of a code for different lengths of burst errors (see section 4.2).

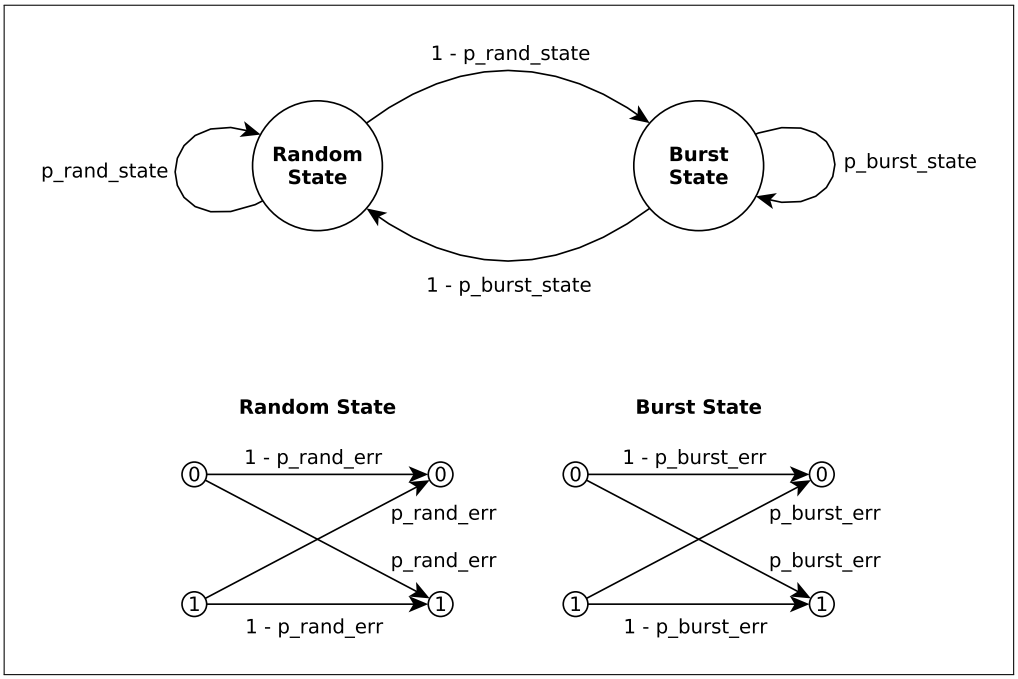

Figure 3.2: States and schema of the Probability Channel

3.3 Decoding

The convolutional decoding is done using the MATLAB function *vitdec*(). It performs the Viterbi algorithm. Analogous to the convolutional encoder *convenc*() it takes the binary code vector and the trellis structure, describing the convolutional code, as parameters. Additionally three more parameters are needed to parameterise the Viterbi algorithm:

- tblen specifies the truncation depth of the Viterbi algorithm. [1] suggests to use $\frac{2.5m}{1-r}$ as a rule of thumb for rate r convolutional codes. The rule of thumb is implemented in tblen_from_trellis.m as a function of the trellis structure.
- opmode specifies the operation mode of the decoder. It is set to '*trunc*' because the whole code is passed at once and preservation of continuity between successive iterations of the *vitdec*() function is not needed nor desired.
- dectype specifies the type of decoder decisions. Since the code consists of binary values the value '*hard*' is used.

Listing 3.2 shows a representative example of a received code sequence and all other necessary parameters passed to the Viterbi decoder *vitdec*(), which then returns the decoded message.

```
1 tblen = tblen_from_trellis(trellis);
2 decoded_message = vitdec(received_code, trellis, tblen, 'trunc', 'hard');
```

Listing 3.2: Decoding of received message

3.4 Performance Evaluation

The evaluation of the decoding performance of the convolutional codes under different channel errors is done by calculating the BER with the MATLAB function *biterr*(). It returns the percentage of matching bits of two given vectors, which are the original message and the decoded message in this particular case. Listing 3.3 shows a representative example of the BER calculation from a message and the decoded message.

```
1 [~, BER] = biterr(message, decoded_message);
```

Listing 3.3: Bit error rate calculation

To evaluate the performance of a single convolutional code in respect to the error scenarios described in section 3.2 three benchmark functions are implemented (see random_benchmark.m, burst_benchmark.m and fixed_burst_benchmark.m). They take a pre-generated message, the trellis structure describing the code and specific scenario parameters as parameters. After performing the encoding, transmission and decoding the BER is calculated for the decoded message as well as the received encoded sequence. The latter can be used as reference for later comparison. The BERs for each specified scenario case are then returned as vector.

4 Discussion

The error scenario benchmarks mentioned in section 3.4 are executed for each of the given three convolutional codes and plotted in graphs for further evaluation (see miniproject.m). The resulting plots are described and discussed in the following sections.

4.1 BSC with different Bit Error Probabilities

Figure 4.1 shows the scenario of a BSC that introduces single error bits which occur with a probability of p.

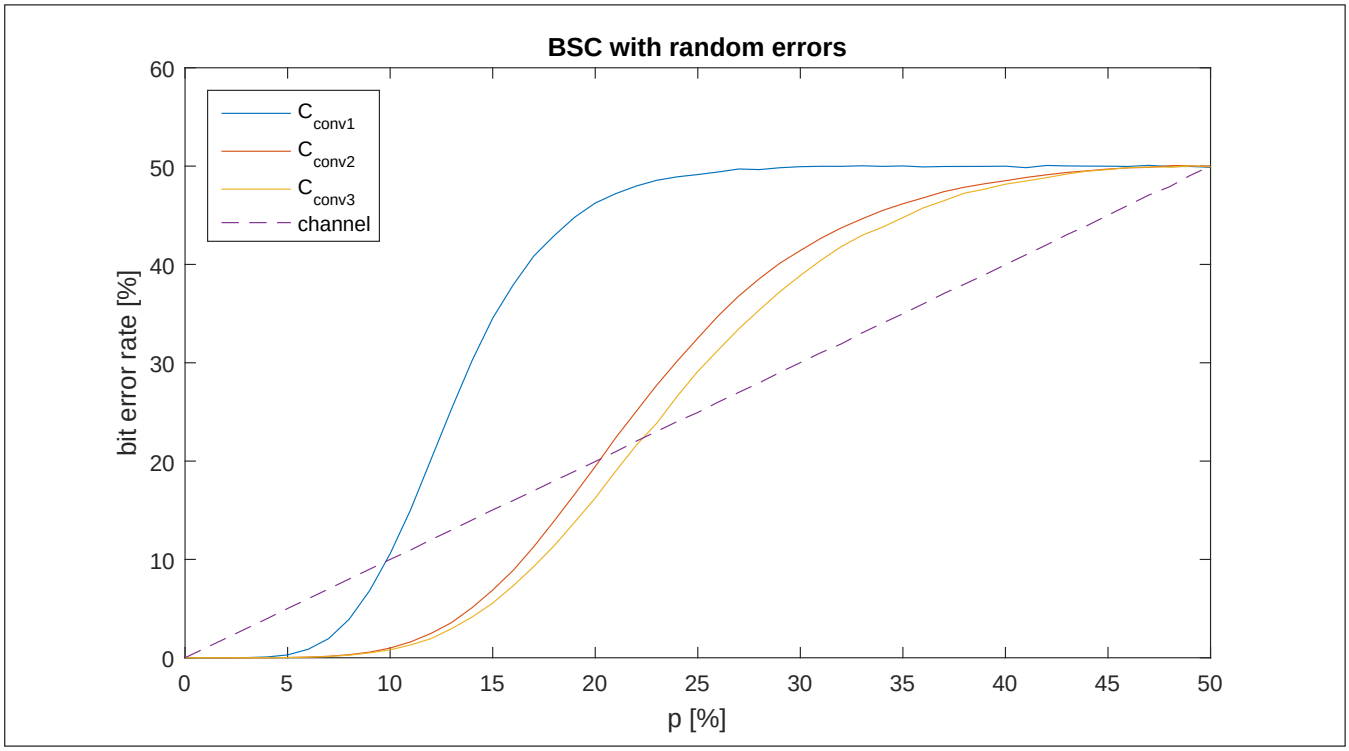

Figure 4.1: BSC with random errors

The dashed line shows the channel bit error rate, that would occur without the use of any coding and can be seen as reference.
It can be observed, that C_{conv3} corrects most errors and has a lower bit error rate that the underlying channel up to a bit error probability of $\approx 0.22\%$, while C_{conv1} corrects least errors and performs worse than the uncoded transmission with a bit error probability higher than $\approx 0.09\%$. It can also be seen that all three codes reduce the BER significantly for bit error probabilities $< 5\%$. The best performing C_{conv3} for example reduces the bit BER by a factor of ≈ 240 at $p = 5\%$ and even by a factor of ≈ 1660 at $p = 3\%$.
In favor of C_{conv1} it may be noted that it has the highest coding rate and introduces less redundancy than the other two codes (see section 2).

4.2 BSC with single Burst Error of different length

Figure 4.2 shows the effect of a single burst error with different lengths during the transmission over the BSC on the bit error rate of the decoded message.

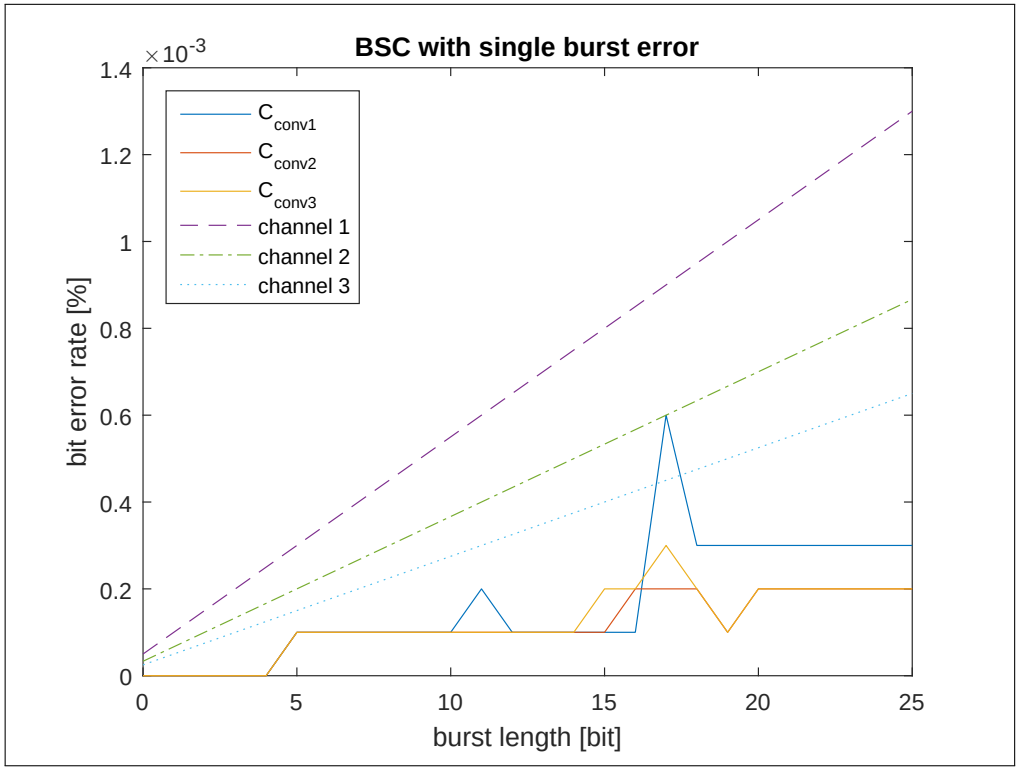

Figure 4.2: BSC with single burst error

Since the codes have different coding rates the fixed burst lengths have different effects on the bit error rate of the channels what can be seen on the dashed and dotted lines.
On the other hand, all codes have the same error correcting capability 4. Hence they all fail to correct more than 4 consecutive bit errors.
For burst errors of length > 4 the number of corrected errors depend on the encoded message, since it affects the traceback path of the Viterbi decoder. Figure 4.3 shows that C_{conv2} and C_{conv3} are able to correct more bit errors of longer burst errors while C_{conv1} starts to fail for burst errors of length > 50 bits and behaves approximately like a transmission without encoding.

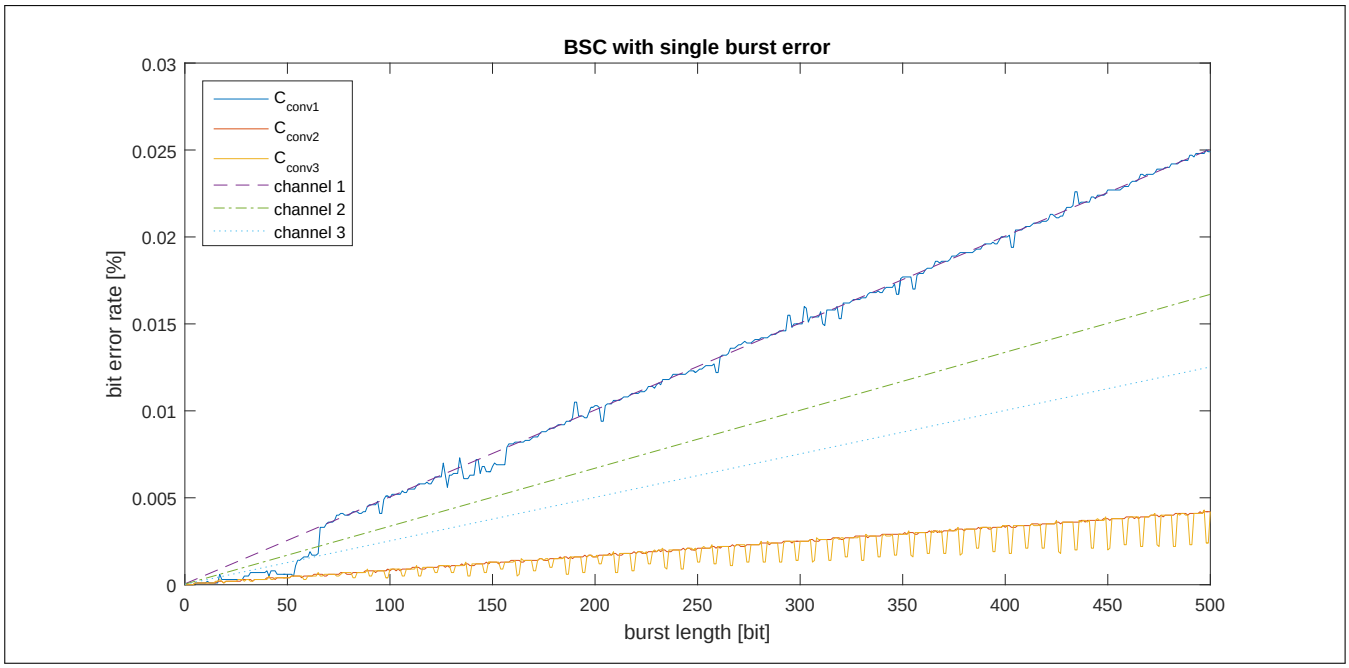

Figure 4.3: BSC with single burst errors

This effect could be explained by the higher number of states C_{conv1} possesses ($2^K = 64$ states; see Table 2.1). Especially for C_{conv3}, which has only 4 different states, consecutive errors may result in the correct state more often, and, thus decode the correct message.

4.3 BSC with Burst State of different Enter and Leave Probabilities

Figure 4.4 shows the performance of the depicted convolutional codes on a BSC with random burst error with different start and end probabilities.

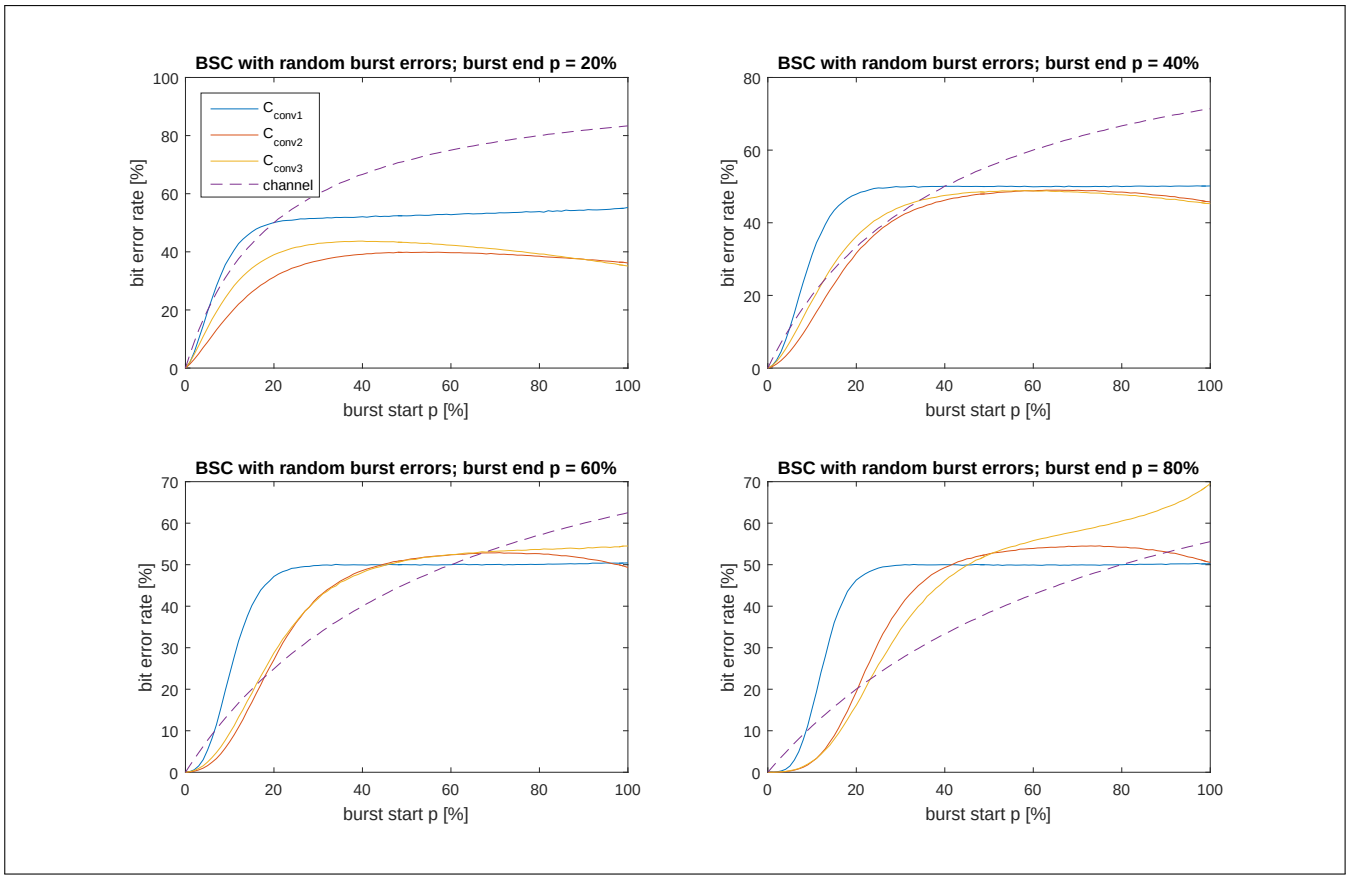

Figure 4.4: BSC with burst errors of different probabilities

While a low burst end probability, like 0.2% in the top left plot, means relatively long burst errors, a high burst end probability, like 0.8% in the bottom right plot, means shorter burst errors. The occurrence of these burst errors is influenced by the burst start probability, which is entered on the x-axis. As before the dashed line shows the bit error rate without coding and can be used as judging reference.

Like already observed in section 4.2 especially C_{conv2} and C_{conv3} perform relatively well if long burst errors occur. This can also be seen in the top left plot. For rising burst end probabilities, and hence shorter burst error lengths, all codes perform better for low burst start probabilities. In particular the bottom right plot with a burst end probability of 0.8%, resembles the plot of the BSC with single bit errors, analysed in section 4.1. The can be explained by the error correction capability of the codes, which allows the correction of short burst errors, especially if the distance of these burst is high.

5 Conclusion

The mini project had three main parts; first, the given convolutional codes were analyzed, underlining the related characteristics and using them to come up with a preliminary set of relevant scenarios for the actual simulating part.
Second, the end to end simulation of a communication channel was implemented in Matlab, through a convolutional encoder (representing the given codes), a BSC channel, and Viterbi hard decoder as decoding component. Furthermore, the error scenarios were added to the channel, plotting the result of our benchmark tests.
Lastly, outcomes of our simulation were tested, comparing the expected results with the actual results. This shown how the given codes react to a BSC with random errors or burst errors. In particular, C_{conv1} had worse performances in both cases. Due to a higher code rate (higher than C_{conv2} and C_{conv3}), which makes it more powerful, the relative BER rises for relatively low error probability, lower than the bit error rate associated with the other two codes (see section 4).

6 Appendix

Matlab.zip containing the source scripts:

miniproject.m
tblen_from_trellis.m
burst_benchmark.m
fixed_burst_benchmark.m
random_benchmark.m
probability_channel.m
fixed_burst_channel.m

References

[1] B. Moision, *A Truncation Depth Rule of Thumb for Convolutional Codes*, In Information Theory and Applications Workshop (January 27 2008-February 1 2008 San Diego California), Page 555–557, New York: IEEE, 2008.