Use of Convolutional Codes for Collaborative Networks

Master's Thesis

performed in Telecom Engineering,
by

RAFAEL CABALLERO ZARAGOZA

Montreal July 20, 2009

Use of Convolutional Codes for Collaborative Networks

Master Thesis

performed in *Telecom Engineering*,
Dept. of Electrical and Computer Engineering
at Concordia University

by **Rafael Caballero Zaragoza**

Examiner: Professor Reza Soleymani

Concordia University

Montreal July 20, 2009

Acknowledgements

First of all, I would like to thank my family, my parents and my sister, for the support and the encouragement they have given me throughout all this university years, in particular in those difficult moments that I had to overcome especially during finals periods. I am very grateful because without them it would have been a lot harder, yet impossible, to fulfil this challenge.

I also would like to thank my supervisor at Concordia, Professor Reza Soleymani for giving me the opportunity to work with him and his group. It has been a very enriching experience for me to follow up my thesis in such a delightful environment.

I am also very grateful to my tutor at my host university, Maria Julia Fernández-Getino García who has supported me not only to take advantage of this experience abroad but also throughout all my years at the university with very valuable advice and encouragement.

I cannot forget about my co-workers at the Concordia lab, Patrick, Hesam and Ding who has helped me out with the guidance and resolution of the various queries I have faced over the realization of this thesis. They have been a key part in the achievement of my results and the knowledge of their own work.

Finally, I would like to mention and thank all the people who have shared those years at the university with the good and the bad moments, especially Jorge, Ramon and Irene.

Abstract

The objective of this thesis was to acquire the knowledge about the works perform by professor Soleymani and his student group in collaborative communications. We decided to start from the beginning simulating a wireless communication scenario and we thought it would be of great value to implement and simulate a convolutional channel code with Viterbi decoding. The last part has been about studying collaboration trough the references and some implemented simulations.

The main issue of this thesis has been the implementation and usage of a convolutional code with Viterbi decoding in a collaborative scenario. Throughout the period of duration of the work, we have first modelled a simple convolutional code that we have tested and compared with the case in which we did not have any channel coding. Later on, in a second phase of the project, we have used that convolutional code implemented combined with collaboration.

To help with the simulations, we have implemented our code in MATLAB and we have made a comparison in performances between different possible scenarios.

Contents

List of figures

List of tables

1. Background

Introduction

Convolutional encoding and Viterbi decoding are error correction techniques widely used in communication systems to improve the bit error rate (BER) performance. However, at some point, we will not be able to enhance our performance due to the impairments of the channel. To overcome these problems we will use collaboration with our channel coding method. We will take advantage of the higher diversity of our system when using the help of other nodes.

Purpose

The initial intent of this project was to design a collaborative scenario to learn the basics behind this technique and its advantages in terms of improving the performance of a whole wireless system.

To do that we have first designed a simple scenario with one source and we have been adding different layers. The most complicated part or, at least, the most laborious one has been the implementation of the convolution encoder and the Viterbi decoder. Once we have done and tested both, we have passed to next step which has been the inclusion of a relay in the system helping the source to get better performances at the destination.

The actual purpose of this work is to actually test the enhancement in the behaviour of the whole system when we add collaboration to it. To accomplish the task, we have been getting used to the topic through different literature.

Method

To realize this project, MATLAB was chosen for development mainly because of its ease of use. The simulations carried out are not that long and heavy but further research on this topic, could suggest an implementation in C or C++ for better simulation performances.

Report structure

In the next chapter we will present our simple scenario for the first part of the thesis followed by an introduction to convolutional coding and Viterbi decoding. Then we will talk about the behaviour of the Viterbi algorithm system.

After that, we will discuss the different fading models we can face in a wireless radio channel because we will need to simulate the fading in our channel.

In the following chapter after the fading topic, we will get into the collaboration part explaining the main ideas behind it that we have gathered out of the massive reading of papers and different references.

The last chapter will be made of different simulation of all the different case scenarios we have studied throughout the whole thesis, from the most basic one, without fading, to the

collaboration scenarios with the different positions of the relay. The whole point of this thesis is ending simulating different collaborative network scenarios in which we will apply the convolutional channel codes we have studied so far. To conclude, plots of simulation results are given and the results are discussed.

2. Description of the scenario

System overview

We are going to implement a basic case scenario, adding different steps at a time to obtain the performances of a convolutional code compared to the ones of different coding techniques.

We are following a series of steps to fulfil this simulation of a convolutional code with Viterbi decoding. At the very beginning, we will have a simple AWGN scenario with a BPSK modulation. Afterwards, we will add the fading due to multipath and we will compare both simulations.

From now on, we will explain all the different steps which are necessary to implement our wireless communications system but as an introduction, this will be our block diagram.

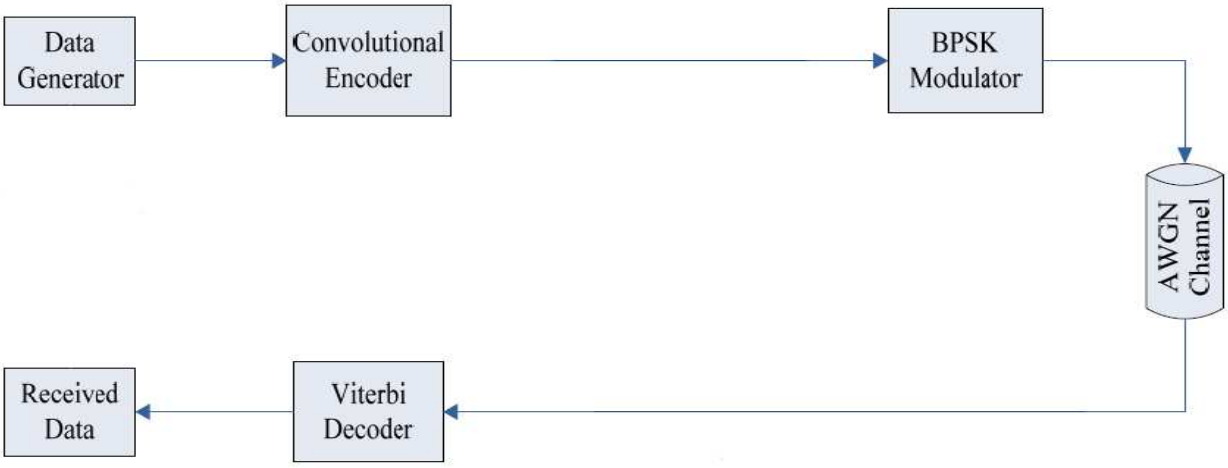


Figure 2.1: system overview

As we can see, it is a very simple block system in which we have space for other blocks that we could add to make the system more real. We consider that we make de demodulation process in the Viterbi decoder because we have soft decision.

The platform we are going to use to simulate our system will be Matlab so if we mention any function it will be something related with the Matlab language.

Generating the data

We use a random number generator that gives us a certain number of bits which will represent our message or information. That information will be encoded and transmitted through the channel. We will use the *rand* function in Matlab.

Convolutional encoder

Convolutional codes are applied in applications that require good performance with low implementation cost. They are used with data streams not with static blocks. We have designed a very simple convolutional encoder with a code rate of *r=1/2* and a memory length of *M=2*.

We can denote a convolutional code with three parameters *(n,k,L)* where *n* is the number of outputs at the encoder, *k* is the number of bits we introduce at the input of the encoder at a time and *L* is the memory length plus the input (*L=M+1*).

As we said before, and following the notation used to describe a convolutional code, in our case we are using a *(2,1,3)* convolutional encoder as we can see in the figure below.

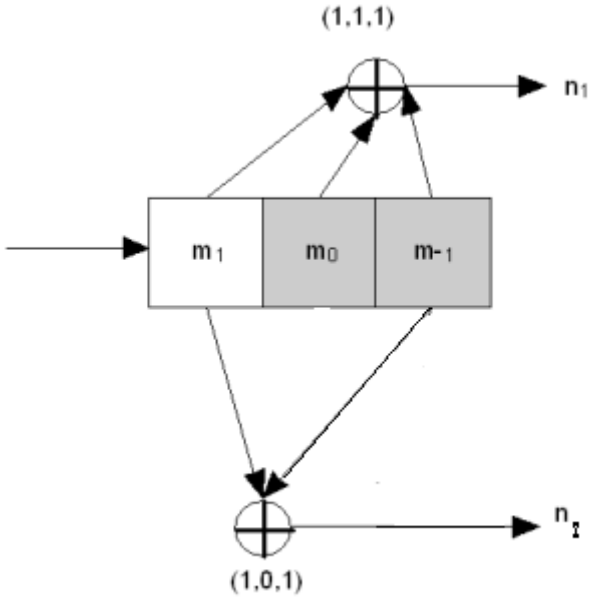


Figure 2.2: convolutional encoder

We can also define our code by series of vectors or sequences that specify a convolutional code completely. If we put those vectors all together we obtain the generator matrix which defines the connections between all the slots and the outputs of our system.

In our case, this generator matrix would be:

$$G = \begin{bmatrix} 111 \\ 101 \end{bmatrix}$$

As we can see, we have as many rows as outputs and as many columns as memory units (counting the input itself).

We have three slots which correspond to the input and the two memory units. To obtain the output of the encoder in each time instant, we just have to make a matrix multiplication of the information in the slots and the generator matrix.

This matrix multiplication could be seen as a modulo-two addition as well. We could think about our system as a cascade of exclusive-or gates. These exclusive-or gates perform modulo-two addition of their inputs following this table:

Input A	Input B	Output (A xor B)
0	0	0
0	1	1
1	0	1
1	1	0

Table 2.1: truth table

This is the truth table that we will follow to obtain the output of the encoder in each time instant.

If we go back to our convolutional code, we can see from the structure that each input has an effect on three successive pairs of the output symbols. That is an extremely important feature and it is what gives the convolutional code its error correcting power.

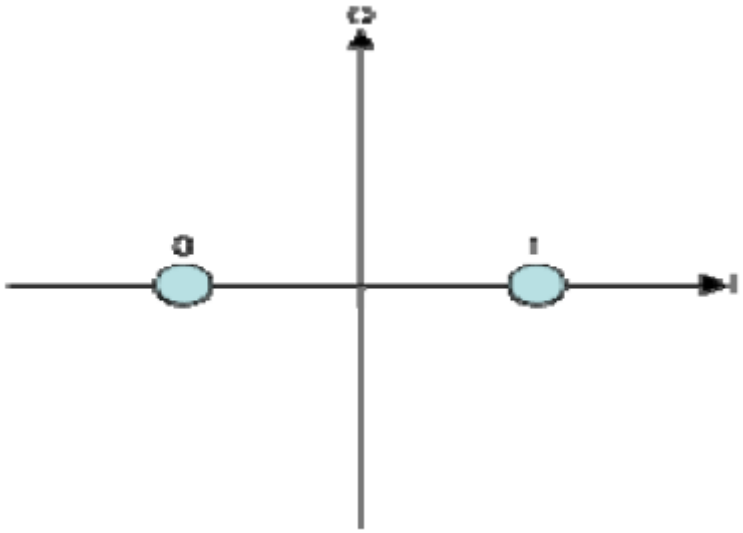
So, for the last bit of our block or data given, in order to affect three pair of output symbols, we need to output two more pair of symbols (maybe we will be

discarding those bits at the end but we have to do that for the last bit). This is accomplished in our example encoder by clocking the convolutional encoder memory slots two more times (*M* times in a general encoder) while holding the input at zero. This process is called 'flushing' the encoder and results in two more pair of output symbols.

Another important aspect to take into consideration is that the encoder must start and end in a known state for the decoder to be able to reconstruct the input data sequence properly.

BPSK modulator

This is the simplest modulation we could have in a system. It is also the simplest form of a Phase Shift Keying (PSK) modulation. We use two phases which are separated by 180º. To make it simpler, the two constellation points are positioned on the real axis, at 0º and 180º.



The binary data will be carried through the following signals:

- For binary '0':

$$s_0(t) = \sqrt{\frac{2E_b}{T_b}}\cos(2\pi f_c t)$$

- For binary '1':

$$s_1(t) = \sqrt{\frac{2E_b}{T_b}}\cos(2\pi f_c t + \pi)$$

Figure 2.3: the constellation diagram for BPSK

Mapping the channel symbols to signal levels

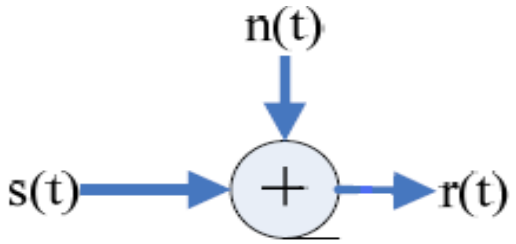
To transmit this binary signal, we are going to use an antipodal baseband signalling scheme. This is just a matter of translating '0's to '+1's and '1's to '-1's. We will perform the operation *y=1-2x* on each convolutional encoder output symbol to accomplish that.

AWGN channel

If we want to simulate the real characteristics of the channel, we will have to add the additive white Gaussian noise process which simulates background noise of the channel under study.

Adding noise to the transmitted channel symbols produced by the convolutional encoder, involves generating Gaussian random numbers, scaling the numbers according to the desired energy per symbol to noise density ratio, E_s/N_0, and adding the scaled Gaussian random numbers to the channel symbol values.

If we have an uncoded channel, $E_s=E_b$, since there is one symbol per bit. However, for a coded channel like in our case we have to take into consideration that we are sharing the energy between all the code word bits. We have to work with the energy per coded word or energy per symbol (E_s).



As an example, we can consider our system with the convolutional channel coding of rate ½. In this case, $E_s \neq E_b$, so to obtain the relation between E_s/N_0 we have to apply:

$$\frac{E_s}{N_0} = \frac{E_b}{N_0} + 10\log_{10}(k/n) = \frac{E_b}{N_0} + 10\log_{10}(1/2)$$

Figure 2.4: the AWGN channel

So as a result of all that, the received signal without taking into consideration any other thing, will be the sum of both, the noise and the information coming from the encoder, *r(t)= s(t)+n(t)*.

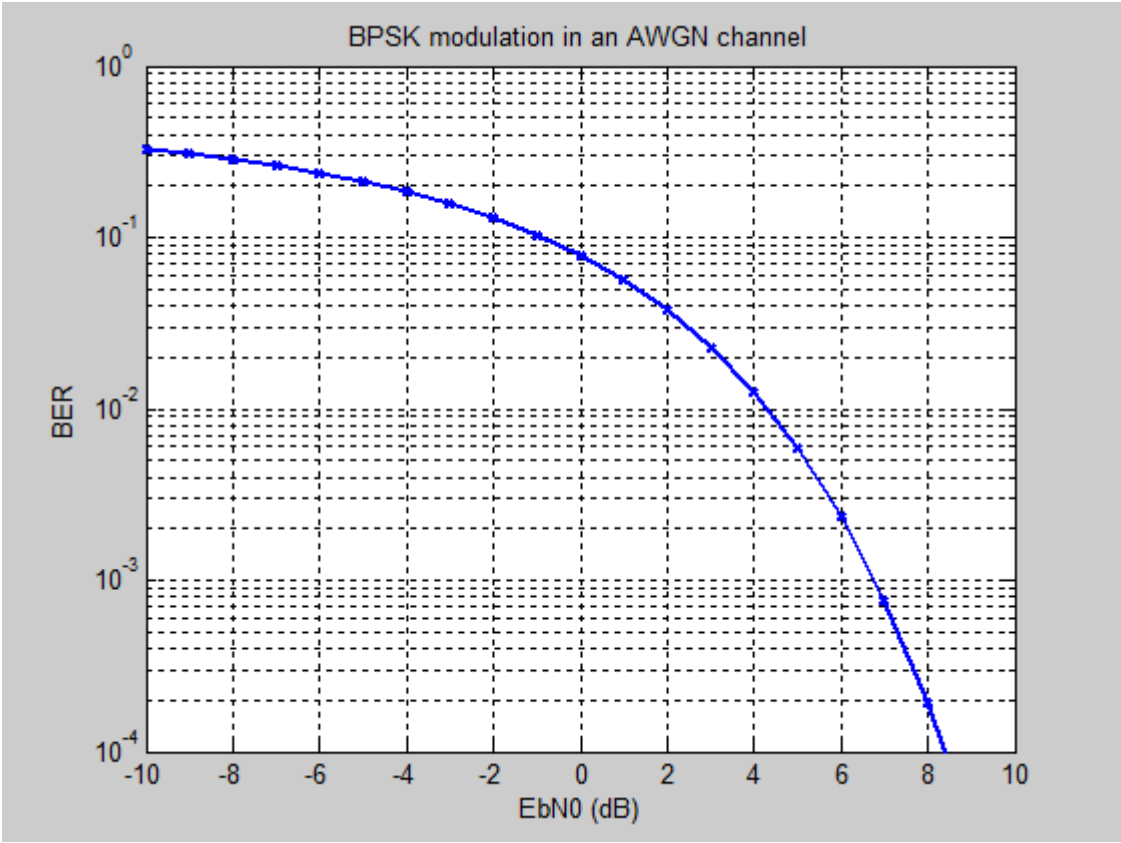


Figure 2.5: BER of BPSK modulation in AWGN channel

To see the result of the different methods of coding in our system, we will compare with the theoretical curve which we can obtain by applying the expression to obtain the BER of a BPSK modulation in AWGN channel:

$$BER = erfc\left(\sqrt{\frac{E_c}{N_0}}\right)/2$$

As we said before, in this modulation case, $E_c=E_b$, so it would not matter which energy we choose to simulate the behaviour of this modulation over an AWGN channel.

We plot the behaviour of this BPSK modulation over an AWGN channel (*Figure 2.5*). That will be our reference for the rest of the channel coding simulations that we will see further on.

Viterbi Decoder

This is the trickiest part of our system so far; it has also been the part which has needed more time to fulfil as we have faced different problems to accomplish it.

The very first step that will help us understand the Viterbi algorithm is the trellis diagram. The figure below shows the trellis for our example (remember that we had a rate of ½ and L=3).

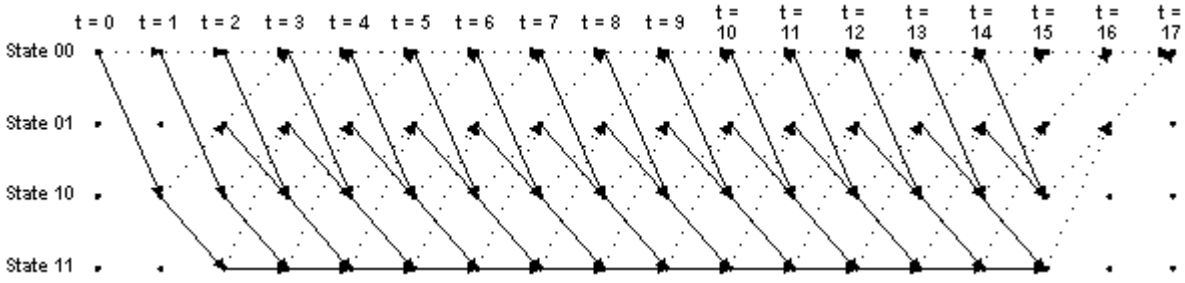
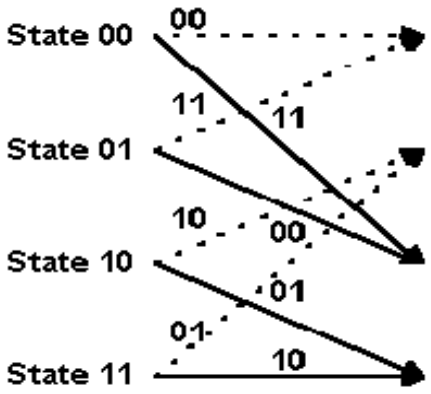


Figure 2.6: trellis diagram for a convolutional encoder of rate ½ and L=3 for a 15-bit message



We have 2 memory slots (*M=2*) so that means we can have up to 4 different possible states which are represented as a four rows of horizontals dots. There is one column of four dots for the initial state of the encoder and one for each time instant during the message.

For a 15-bits message with two encoder memory flushing bits, there are 17 time instants in addition of the initial one (t=0). The solid lines connecting dots in the diagram represent state transitions when the input bit is a one whereas the dotted lines represent state transitions when the input bit is a zero.

Figure 2.7: transition between states

The two-bit numbers labelling the lines are the corresponding convolutional encoder channels symbol outputs (remember that *n=2* in our system).

One of the most important things we have to notice is that as the initial condition of the encoder is the state '00' and the two memory flushing bits are zeroes, the arrows start and finish at this state. This is an important feature of the trellis diagram as we have to know which are the initial and final states of the encoder to proceed with the decoding process.

Let's start explaining how the Viterbi algorithm actually works. Each time we receive a pair of channel symbols, we are going to compute a metric to measure the distance between what we received and all the possible channel symbols pairs we could have received.

In our example, we are using soft decision which means that we are considering Euclidean distance to measure. The distance values we compute at each time instant for the paths between the states at the previous time instant and the states at the current time instant are called branch metrics. For the first time, we are going to save these results as 'accumulated error metric' values associated with the states.

If we take the first transition as an example, which means going from t=0 to t=1, there are only two possible channel symbol pairs we could have received, '00' and '11'. This is because we know the convolutional encoder was initialised to the all zeros state and given one input bit (that can only be a one or a zero), there are only two states we could transition to and two possible outputs for the encoder.

In each of the following states, we will be adding the branch metric values to the previous accumulated error metric values associated with each state that we came from to get to the current one. What we carry forward to the next time instant is the accumulated error metrics for each state and the predecessor states for each of the four states at the previous time instant.

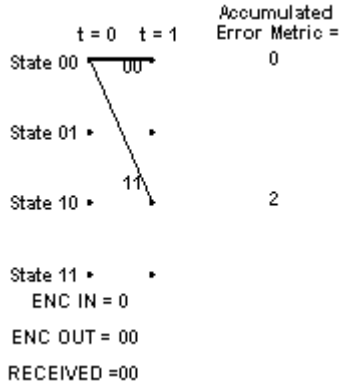


Figure 2.8: result at t=1

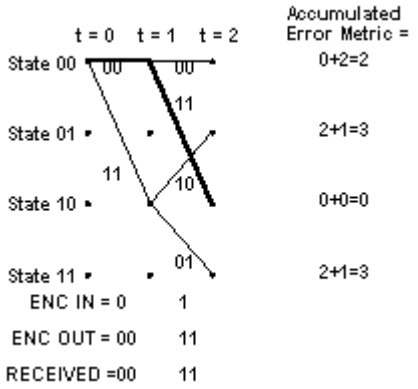


Figure 2.9: result at t=2

In the figures above, we are showing the process with hard decision which is easier due to the use of Hamming distance that only counts the number of different bits between the received channel symbol pair and the possible channel symbol pairs.

Note that the solid lines between states at t = 1 and the state at t = 0 illustrate the predecessor-successor relationship between the states at t = 1 and the state at t = 0 respectively. This information is shown graphically in the figure, but is stored numerically in the actual implementation. To be clearer, at each time instant t, we will store the number of the predecessor state that led to each of the current states at t.

From this point and starting at time instant t=3, things get a little bit more complicated since there are now two different possible ways to get to each of the four possible states from the previous ones. The way to handle this is just by comparing the accumulated error metric associated with each branch and discarding the larger one of each pair of branches leading into a given state. Dealing with Hamming distances, we can have some cases in which we have the same metric for both paths, we just save one of them, but in our case, for soft decision decoding, we will rarely have this problem.

The operation of adding the previous accumulated error metrics to the new branch metrics, comparing the results, and selecting the smallest accumulated error metric to be retained for the next time instant is called the add-compare-select operation.

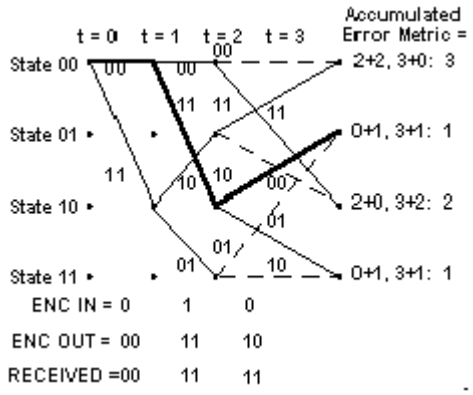


Figure 2.10: result at t=3

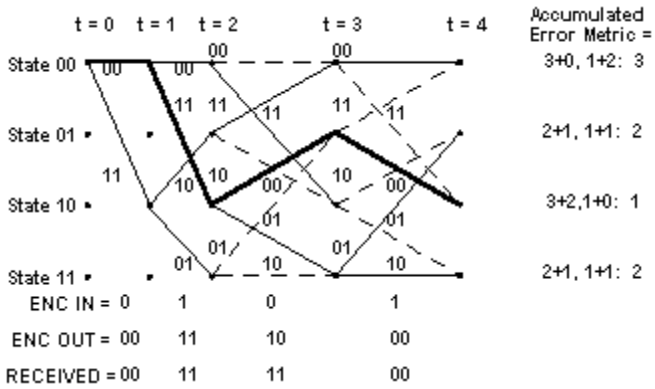


Figure 2.11: result at t=4

In the two figures above we can see the result of the process for the time instants t=3 and t=4. We have to note that in t=3 we are committing an error between the encoded word which comes out of the encoder and the received word after being transmitted through the channel. Normally after de decoding process we will be able to solve that error and we will not actually count it as so.

Notice that at t=4, the path through the trellis of the actual transmitted message, shown in bold, is again associated with the smallest accumulated error metric. This is the main idea behind the Viterbi decoder and the one that it exploits to recover the original message.

At this point of the process, we have already studied the different case scenarios that we can face in the trellis, which are the first two steps (M in a general case) and the rest, up to the end of the trellis, starting in our case at time instant t=3. So we can get to the end at time instant t=17 and the trellis look like this without the messy paths in every time instant.

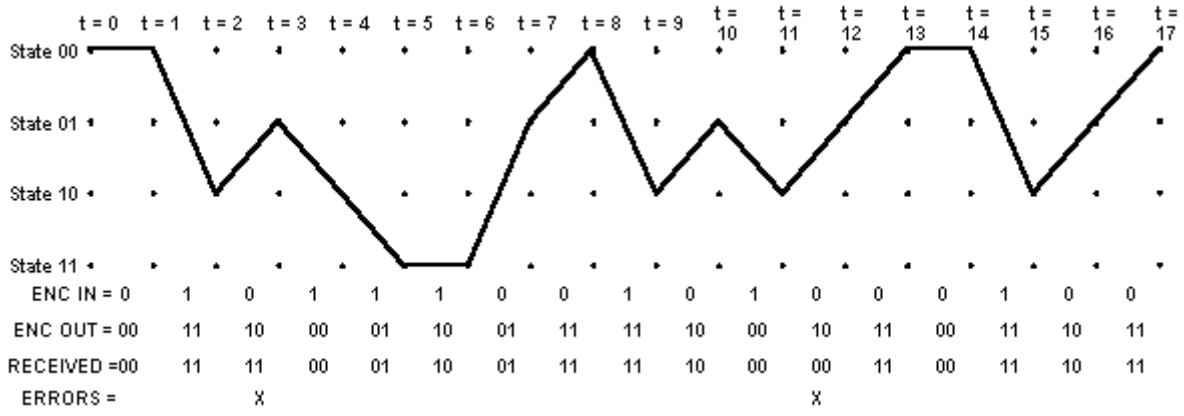


Figure 2.12: trellis optimum path after getting to the end

Once we have got to the end of the trellis we are ready to perform the decoding process, ready to generate the decoder output from the optimal path. We have to build a matrix with the accumulated error metric for every single state and for every single time instant and another one

with the history of the sates that preceded the states at instant t with the smallest accumulated error metric. Once this information is built up, the Viterbi decoder is ready to recreate the sequence of bits that were input to the convolutional encoder when the message was encoded for transmission. To accomplish this we have to follow the following steps:

- We select the state having the smallest accumulated error metric and save the state number of that state
- Then, iteratively we perform the following step until we reach the beginning of the trellis: we work backwards through the state history table (the one that shows the surviving predecessor states for each state at time instant t), for the selected state, select a new state which is listed in the state history table as being the predecessor to that state and saving it. This is the backtrack step.

t =	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	0	0	2	1	2	3	3	1	0	2	1	2	1	0	0	2	1	0

Table 2.2: table with states selected when tracing the path back through the survivor state table

- To finish, we will work forward again through the list of selected states we have saved in the previous steps. With the help of the table showing the next state given the input and the current state, we can know which is the input bit that corresponds to a transition from each predecessor state to its successor state. That is the bit that must have been encoded by the convolutional encoder.

	Input was, Given Next State =			
Current State	$00_2 = 0$	$01_2 = 1$	$10_2 = 2$	$11_2 = 3$
$00_2 = 0$	0	x	1	x
$01_2 = 1$	0	x	1	x
$10_2 = 2$	x	0	x	1
$11_2 = 3$	x	0	x	1

Table 2.3: table that maps state transitions and inputs

So after all the process, we must have the same 15 bits that were encoded at the beginning. That means that we are discarding the last 2 bits (M in a general case) which were the flushing bits.

The performance estimation block

This block, we could actually have avoided the explanation because it is a trivial thing to do to obtain the performances of a channel coding. What we will do it is to compare both, the message word which enters the convolutional encoder with the decoded word which we get from the Viterbi decoder. We will then compute the errors we are having for different Eb/N0 ratios.

3. Radio channels

Introduction

The radio channel place the basic limits on the performance of wireless communications. A radio channel can be of a variety of types and it can change with time. This random nature of radio channels make them very important in the wireless transmission systems. In traditional systems we experience additive white Gaussian noise (AWGN) channel model with thermal noise in the receiver components along with antenna temperature as the main source of signal degradation. In radio channels, this model fails due to the random nature of propagation channel.

This random nature causes rapid changes in signals amplitude, phase and frequency and is generally called fading. More importantly in a wireless channel, a signal can travel over multiple paths between transmitter and receiver and can contribute to multi path fading.

The term fading refers to the time variation of a received signal power caused by changes in the transmission medium or paths. In a fixed setup, fading is affected by changes in atmospheric conditions such as rainfall. But it is different in a mobile environment where one of the two antennas is moving relative to the other, the relative location of various obstacles changes over time, creating complex transmission effects.

Fading models

The modeling of a channel is generally based on large and short area distances. We have two big different models of case scenario we can actually simulate. One of them is large scale fading which is a model based on large area distances that covers mean signal strength of the signal and are efficient in measuring radio coverage area in broader aspects. On the other hand, we have models based on rapid changes of signal strength over short areas and time which are called small scale fading. We can make a classification of the main types of radio channel fading.

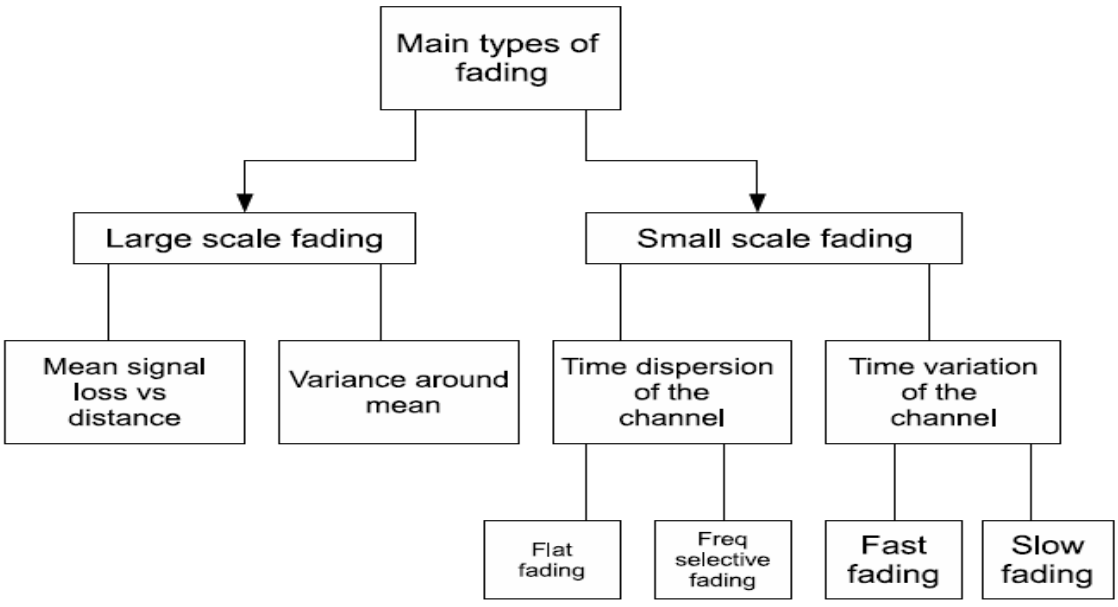


Figure 3.1: types of fading in radio channels

Large scale fading

Impact of surrounding landscape and infrastructure on received signal strength are described in large scale fading model. This is described in terms of mean path loss nth-power law and a log normally-distributed variation about the mean. Mean path loss as a function of distance‘d’ between mobile end and base station can be expressed as:

$$L_p(d) \propto (d/d_r)^n,$$

Where L_p is the path loss, d_r is reference distance generally in the range of 100 m to 1 km depending on the size of cell and value of path loss exponent n depends on propagation environment and generally lies between 2 and 4.

Measured values have shown that path loss is a random variable with a log-normal distribution. So, in more appropriate terms we can generalize large scale fading model as:

$$L_{ls} = L_p(d) + Y_\sigma,$$

Where Y_σ, is deviation about the mean.

Small scale fading

This type of fading is caused by multi path in the radio channel. It can cause fading in received signal in three main ways:

- Fast changes in the received signal over a short period of time or distance.

- Due to the time variation of the channel, various Doppler shifts can cause frequency modulation of the signal.

- Multi path delays can generate replicas of the original signal.

We can follow different models for the simulation of these different impairments that can occur to the transmitted signal regarding fading. If the received signal is composed of line of sight components, then we can make the assumption that this small scale fading can be modeled as Ricean fading. If the line of sight component in the received signal is zero then it is called Rayleigh fading. This is the model we are going to focus in because it is the most common one in wireless communications. An example of Rayleigh fading is shown below:

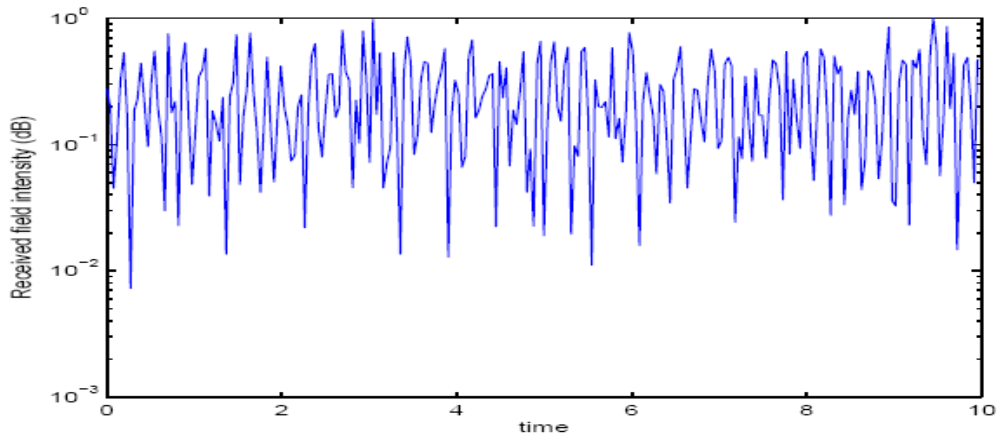


Figure 3.2: a typical Rayleigh fading envelope

Small scale fading: mechanisms, categories and effects

For the time being, we are going to focus on small scale fading because of the multi path in the radio channel. We have been speaking about the different types of fading we can have for a mostly real scenario. We could make two different classifications regarding the frequency dependence or the time variance.

Time dispersion of the channel

Time dispersion can be observed by plotting received power of the transmitted signal varying with time delay τ. We can define τ as the copy of signal component received after the arrival of the first signal. We can define maximum excess delay T_m as the time during which first and last multipath component of the transmitted signal is received. If we compare T_m with symbol time T_s, we can observe that our signal is distorted in two main ways, which we will explain subsequently.

- **Frequency selective fading**: that happens when T_m is greater than T_s so the received multipath components are arriving beyond the symbol time duration which causes channel related inter symbol interferences (ISI). We will have different types of attenuation regarding the frequency used to transmit in every case.

- **Flat fading:** in this case, $T_m<T_s$, so it means that all the received multipath components are arriving within the symbol time period. We then, can assume the same behaviour of the signal in every different frequency. For such a scenario, we will not suffer from *ISI* but still the received components can add destructively and cause variations in the received signal, in the signal to noise ratio (*SNR*) of the received signal.

Time variance of the channel

We can either define the time variance of the channel as the relative motion of transmitter and receiver with respect to each other or the motion of the objects in the channel. Due to this variation in the channel, signal amplitude and phase are varied as channel propagation paths are constantly changing. We can define a time constant T_0 as the time over which the channel remains constant. We can then compare this no-variation of the channel characteristics time with the time per symbol (T_s). We obtain two different classifications.

- **Fast fading**: that is the case when $T_s>T_0$. We have a different behaviour of the channel for every single bit we send. This is not a very realistic case scenario cause if that is so, it would be very difficult to simulate this type of channel.

- **Slow fading**: this is the opposite case, the one in which $T_s<T_0$, which means that the channel is staying constant over the symbol time period. We can assume we have the same scenario within an hour for example, it doesn't change that much and the impact on the signals transmitted will be the same.

When we want to simulate this fading phenomenon, we need to have a statistical model. We assume that there is a large number of scatters in the channel that contribute to the signal at the receiver so we can then apply the central limit theorem that leads to a Gaussian process model for the channel impulse response. If the process is zero mean, which is the case for a Rayleigh fading model, the envelope of the channel response at any time instant has a Rayleigh probability distribution and the phase is uniformly distributed $(0,2\pi)$.

For our case scenario, we will only care about the amplitude of the fading model as we are in a BPSK scenario and the shift in the phase will not affect our performances.

4. Collaborative networks

Introduction

In this part we are going to study the case of collaboration in wireless networks to achieve better performances. When we talk about collaboration we are talking about spatial diversity and when we talk about this term we should think that we actually need multiple transmit antennas at the source. Normally that is not possible in real world applications due to practical issues that force us to limit the amount of antennas we can actually have in a wireless device.

Lately, we have thought about the actual possibility of using other idle nodes in our wireless system to take advantage of them and make them help us to transmit our information to the destination. That scenario can be viewed as a virtual transmitting antenna array.

Following with our work, we will try to see the enhancement in the performances of our convolutional code when we add the help of the relay to transmit our information.

When we talk about collaborative communications we should mention the previous use of multiple-input multiple output (*MIMO*) techniques. With this technique we actually improve the capacity of the system by increasing the number of transmitting and receiving elements. We are transmitting our data through different faded channels (each of the antennas in the transmitter and the receiver) so that means we are increasing the diversity of our system. One of the channels might be affected by a bad fading coefficient but the others might not so we take advantage out of it.

As we can see, transmit diversity is actually an advantage when dealing with wireless networks. However, we have to think about the mobility in this type of systems. We need to have portable devices easily carried from one place to the other. This requirement of mobility is clearly an obstacle to implement several transmitting or receiving elements in a real wireless device. Mobility and transportability impose size constraints on the wireless terminals.

So that is the reason why collaboration has been proposed. The basic idea behind it is that in a network composed by wireless nodes a possible transmitting node with a single antenna could be added by an idle node emulating a *MIMO* scenario. That idle node will be another wireless device.

Background ideas

We are going to explain some previous ideas that have lead to the development of collaborative communications.

As we have already stated, collaborative communications were introduced to allow devices equipped with a single antenna to attain

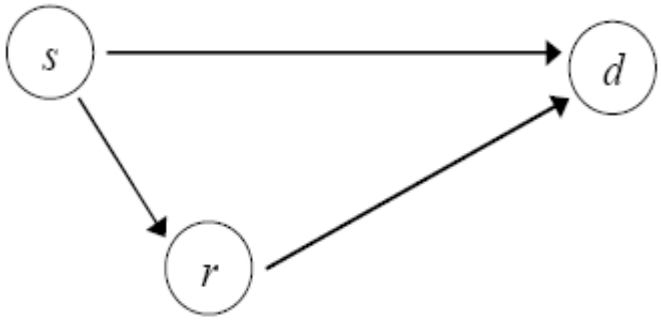


Figure 4.1: wireless collaborative scenario

higher order spatial diversity. Thus, they are capable of dealing with the fading issue which is really important in wireless communications.

At the very beginning we were supposing that the channel between the source and the relay was only affected by an additive white Gaussian noise (*AWGN*). When we introduce the fading phenomena in the system is when it is worth talking about collaborative communications as we can attain a higher degree of spatial diversity.

Basic assumptions

We are going to present the scenario we are going to work with. Actually, we will be working with the basic scenario, the most basic example of collaborative networks in which we have one source, one relay and one destination. So far, we had not considered the relay in our convolutional system but we have the same situation in which a source wants to transmit some kind of information to the destination. However, this time he will be helped by an idle node in the system considered the relay. Working with the relay gives us the opportunity to have another way of transmitting our information through a different path which will have a different fading. As a result we will be maximizing the transmit diversity.

The scenario that we have is basically the same as we saw in Figure 4.1 but in this case, we will be adding some parameters that we will be able to adjust. Different simulations will be done changing the value of these parameters.

As we can see we have three different parameters for each path, D_{ij}, H_{ij} and PG_{ij} that stand for distance between nodes, fading coefficients and path gain respectively.

When we talk about path gain we actually should be talking about path loss because it represents the long scale fading coefficient which might lead to an attenuation of the signal.

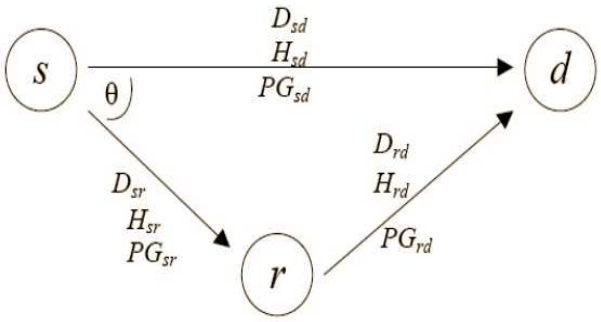


Figure 4.2: network model with parameters

What we do is we set the source-destination gain and the distance between source and destination to be 1 (PG_{sd}=D_{sd}=1) and we determine the rest of the path gain relative to this PG_{sd} following this equation:

$$PG_{ij} = PG_{sd}\left(\frac{D_{sd}}{D_{ij}}\right)^{\alpha}$$

We are going to use a path loss exponent of 2 (α=2) for all results. We also assume the distance between source and relay to be greater than zero because if D_{sr} was zero we would be having a normal *MIMO* system.

If we talk about the fading coefficients for the system, we will assume we have a small scale fading at the beginning. We can actually make some simulations assuming fast fading but at the beginning we will set them to be the same throughout the whole frame. As in the previous simulations, we also assume that the destination knows the behavior of the channel between him and the source and between him and the relay. That means he will know the value of the fading coefficients. As for the relay, it will know the behavior of the channel between him and the

source so it will also know the fading coefficient between both of them. We have to add that as we assume those fading coefficients to follow a Rayleigh pattern, they will be modeled as independent samples of a complex Gaussian random variable with zero mean and a variance of 0.5 as before.

In previous sections we did not pay attention to the large scale fading as we assumed it to be 1 but now we have to quantify the over-all attenuation of a signal suffered through any link. Thus, we have to take into consideration both, the path gain and the fading coefficient as follows:

$$G_{ij} = \sqrt{PG_{ij}} H_{ij}$$

As we can derive from what we have already assumed, in the scheme we are proposing, the source is considered blind. Basically, it is neither aware of its transmission channel's fading coefficients nor of the relay collaboration status. As we can imagine, the relay is neither aware of its transmission channel's coefficients with the destination.

Two phase communication

If we want to keep source and relay working in the same frequency we need to have a two phase protocol for the relay. It will be impossible for the relay to be transmitting and receiving at the same time if we do not want to use another frequency. If we were to use another frequency we would be using more bandwidth which is not a desired situation, it could be even not possible at some point. The thing is that due to the severe attenuation through the wireless channel any received signal will be very weak compared to the one transmitted to the source, that means that if we are transmitting at the same frequency, any transmitted signal from the node would overwhelm any received signal.

Therefore, the relay has two operations to perform; it must listen first to the source and then send its own signal to the destination. As a result of that, two-phase protocols have been proposed for collaboration.

In a two-phase protocol, the relay spends the first phase (also known as the exchange or listening phase) receiving the data from the source. In the second phase (also known as the collaborative phase) the relay transmits its own signal to the destination. Depending on the protocol under study, the destination may listen only to the collaborative phase or both phases to perform the decoding. Two-phase protocols eliminate the need for the relay to collaborate on a different wireless channel. However, they increase the bandwidth requirement. It is evident that any signal sent by the source will require twice the bandwidth for the relay to be able to collaborate because it has to listen first.

When we talk about collaboration we might assume full diversity which means that all the information sent by the source is being received at the destination through two different channels with two different gains. When we compare direct transmission with collaboration we realize that we have to send the information to the relay as soon as possible for it to be able to collaborate.

So we will have to change the rate of transmission to be able to have collaboration which means that the entire data transmitted from the source must be contained in a fraction of the channel allocation used in direct transmission.

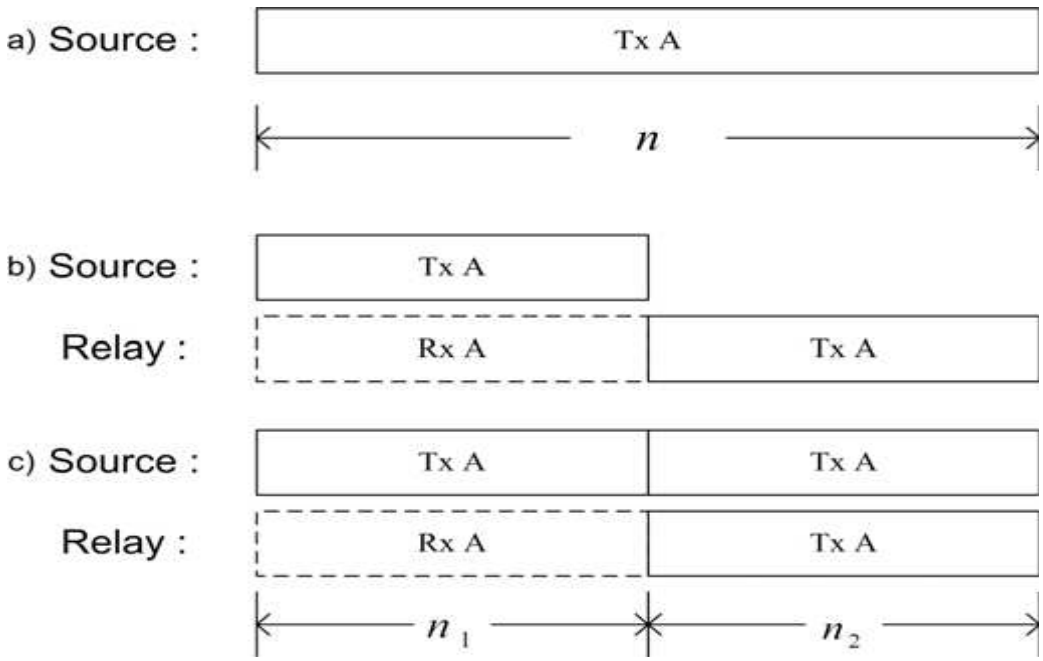


Figure 4.3: a) direct transmission, b) collaboration with source silent in second phase, c) collaboration with source transmitting in the second phase

As we can see the source might be transmitting or not in the collaboration phase. The information is transmitted in the first phase and once both, the source and the relay, have the information we can let the source retransmit it (in the form of parity or repetition).

In this case both phases have the same length which is one of the many different cases we can have. We have to notice that if the period of collaboration is shorter than the period of listening some of the information symbols might be only transmitted once which would mean that the whole system would not be considered as a full diversity system.

If we want to compare the b) and c) cases we have to take into consideration that we are transmitting more information (in the form of parity or not) in the c) case. Hence, the bits transmitted in the b) case should have higher power in comparison with those transmitted in c). As an example to understand the whole idea, if we receive a bit of information three times we will have a lower probability of error than if receive it twice but, if those two received bits have higher power they would be less likely to be in error. As a result of that we can actually make a fair comparison between those two case scenarios.

Detect and forward

This was one of the first methods proposed at the beginning when talking about two phase communication. For this protocol, the relay listens in the first phase and attempts to detect the transmitted symbols from the source and then retransmits the detected symbols in the collaborative phase. So basically, we can make two different parts, the non-collaborative part and collaborative one.

As we can see this is a simple way to achieve collaboration in a wireless channel. However, we do not have full diversity for the whole set of symbols we are transmitting. The method was proposed to increase the throughput of the system so we need to balance between the symbols that are sent with and without collaboration. That is the reason why some symbols might attain full diversity whereas those transmitted without collaboration can only achieve unit

diversity. Therefore, the overall performance of this method does not maximize diversity and hence cannot be assumed to maximize throughput either.

We could be having this example between source (S) and relay (R):

S: a_1 a_2 a_3 a_4

R: a_3 a_4

So as we can see in this example, the first two symbols are attaining unit diversity whereas the last two are actually achieving full diversity. The whole system cannot be considering as a full diversity system.

Besides, we will have to handle with other drawbacks for this method. The thing is that if the detection at the relay is poor or incorrect, we are doing nothing to recover from it, we are just sending the information we have received to the destination, so we might attain worse performances at the end. In addition to that problem, we need to review that to achieve optimal decoding, the destination needs to know the characteristics of the channel between source and relay.

Full diversity collaboration

As we have already said, one of the most important features of collaborative communications is that we can attain full diversity for the whole set of information bits. To attain that, several methods have been proposed, all of them following a two phase pattern. We are going to focus in two main protocols, Amplify and Forward (AF) and Decode and Forward (DF). To explain both of these methods we will consider that only the relay is transmitting in the collaborative phase. This is only an assumption we are making but the source could be very well transmitting at the second phase too.

Amplify and Forward

This is a simple method to achieve collaboration in the wireless channel. In the first phase, the relay listens to the information coming from the source and then simply amplifies the signal and retransmits it to the destination. We could see that method as a version of repetition encoding, where source and relay transmit basically the same thing although we have to take into consideration that the information sent by the relay is actually corrupted by the noise in the channel between source and relay.

As a basic example of this method between source (S) and relay (R):

S: a_1 a_2 a_3 a_4

R: $\beta(a_1+n)$ $\beta(a_2+n)$ $\beta(a_3+n)$ $\beta(a_4+n)$

This time, we have a gain (β) added by the relay but we are amplifying not only the information but also the noise in the source-relay channel. We could actually be sending errors which means that the decoding process would be compromised.

We are going to assume that the exchange and collaboration phases are of the same length (which means that once the source has transmitted half of his frame the relay starts to collaborate) to see how are the signals received at each phase at the relay and the destination.

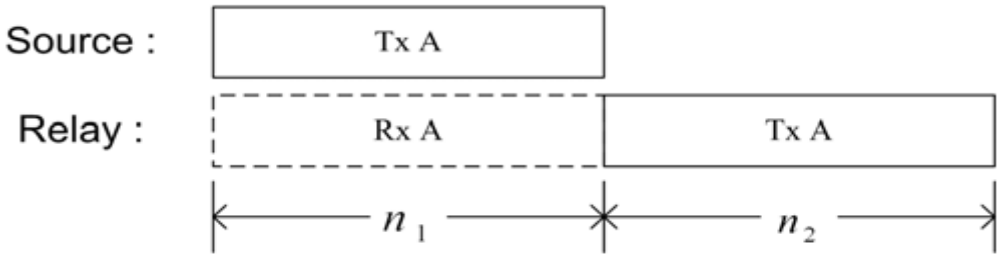


Figure 4.4: collaboration with source silent in the second phase

So $n_1=n_2$ which means that the collaboration fraction is ½ which is only an assumption due to the fact that we can actually have another rate.

During the exchange phase of the AF protocol, the received signals at the relay and the destination are respectively those:

$$y_r^e(k) = \sqrt{E_s} G_{sr} x_s^e(k) + z_r^e(k)$$

$$y_d^e(k) = \sqrt{E_s} G_{sd} x_s^e(k) + z_d^e(k)$$

The term $y_i^e(k)$ stands for the kth symbol received at node *i* during the exchange phase. As we can see the symbol transmitted by the source is the same and we have different samples of noise due to the different channels. Both of them are Gaussian random variables with a variance of No/2 per dimension. E_s is the energy per transmitted symbol and G_{ij} is the channel coefficient that we already defined before, taking into consideration the path gain and the fading coefficient.

During the exchange phase the relay processes the data and amplifies it. The received signal at the destination during the collaborative phase will be:

$$y_d^c(k) = \beta \sqrt{E_s} G_{rd} y_r^e(k - n/2) + z_d^c(k)$$

Where β is the amplifying gain at the relay.

To sum up, we have to remember the main idea of this protocol which is that even though noise is also amplified at the relay, the destination is receiving two copies of the signal through two independently different fading channels, which means that we are accomplishing our goal which is to maximize the spatial diversity.

Decode and Forward

This is another two-phase protocol in which the relay not only receive and forward the information but it also decode it and re-encode it again to transmit it to the destination. As we can imagine, this is a more robust method of collaboration as we no longer have the effect of the possible errors between the source and the relay. However, this extra step will mean more work and more complexity for the relay.

As a rough example of this method between source (S) and relay (R):

S: a_1 a_2 a_3 a_4

R_1: a_1 a_2 a_3 a_4

R_2: b_1 b_2 b_3 b_4

We are showing the possible case in which we could have two different relays listening or collaborating with the source. As we can see, for the first relay, we are assuming that the encoder is the same as the one in the source, so when we re-encode the symbols we are obtaining the same code word as in the source. For the second relay, we have decoded the information coming from the source and we have re-encoded it with a different encoder. That is the reason why we have different code words sent to the destination.

Obviously, at the destination we will need to have some kind of information about the coding process in both the source and the relay if this is not the same.

If we get into a more formal example as we did for the AF case we can say that for the first phase of the protocol, which is always the exchange phase the signals received at the relay and the destinations remains the same as in the AF protocol:

$$y_r^e(k) = \sqrt{E_s} G_{sr} x_s^e(k) + z_r^e(k)$$

$$y_d^e(k) = \sqrt{E_s} G_{sd} x_s^e(k) + z_d^e(k)$$

The real change in this method comes actually at the collaboration phase after the relay has decoded the codeword coming from the source and re-encoded it. Thus, the signal received from the relay at the destination follows this pattern:

$$y_d^c(k) = \sqrt{E_s} G_{rd} x_r^c(k) + z_d^c(k)$$

As we can see, the coded symbol $x_r^c(k)$ stands for the k^{th} symbol obtained from the encoder at the relay during the collaboration phase. It is the result of a process of decoding and re-encoding:

$$x_r^c(k) = u(w(y_r^e(k)))$$

For that expression, u(.) is an encoding function and w(.) is a decoding one.

We have to notice that if the encoder at the source and the destination is the same which means we have a sort of repetition code but through different channels, the symbols received from both should be following the same pattern so $x_r^c(k) = x_s^e(k)$.

To conclude with this section we need to make a comment on the process of decoding in the relay. As we can imagine, if we want a perfect decoding process of the information coming from the source without any error, we will need to have a high SNR between source and relay or a very favorable channel between them.

In reality, we will need to be satisfied with a certain degree of BER to start the collaboration process. Based on that certain threshold we can work with selection relaying. In this case, when the relay is unavailable to decode the information coming from the source, it does not help and the source decides to go back to the direct transmission for the remainder of the frame.

We will need to balance between attaining full diversity and being able to decode the information coming from the source with a certain degree of reliability. That means, if we want to have the information sent through different channels (full diversity) we will have to relax the decoding constraint at the relay for this to be able to re-encode the information and actually collaborates with the source.

Time fraction

This is a variable we have considered so far as being ½. The time fraction is defined to be as the ratio of time spent by the relay listening to the source versus the over-all time of the whole process of listening and collaboration.

$$\Delta = \frac{n_1}{n}$$

Variable time fraction

We might think about changing the time spent at the listening phase to increase the time the relay is actually collaborating, thus we would increase the performance at the destination. The problem is we need to ensure a minimum BER value at the relay for it to collaborate. If we shorten the listening part the relay might not get what the source is transmitting so it would eventually forward errors to the destination. No matter the two phase method used, we might not be helping the destination to decode the information properly.

One solution to this problem is to increase the power of the signal transmitted from the source but by doing so, we will be increasing the over-all energy cost which is not good.

As a result of this issue, we are presenting a protocol which deal with the time fraction selection. It will be a matter of the relay, the task to choose its own time fraction and by doing so, we can be sure to always maximize the amount of time the relay is collaborating.

The process is very simple, assuming that we divide the frame into a set of blocks; the relay listens to the source until it can decode the data with a certain error rate, then it starts collaborating. In order to accomplish this process the source must be transmitting a signal during the entire frame time and it will never become silent. Like that, we can have different time fractions possibilities.

For the destination, to achieve optimal decoding, it needs to be aware of the time fraction decided by the relay as well as the fading coefficients at the source-destination and the relay-destination path.

Collaborative coding

That is what we call to the combination of coding and collaboration. Up to now, we had assumed we did not have any coding for the information bits generated at the source but form now on we will consider some type of coding for that information. At the very last, the idea is the same because we are sending sets of encoding information through different faded channels.

This is actually the next step in our work once we have generated our convolutional encoder and decoder we integrate them with the collaboration part. We could have perfectly

worked with any other code to end up doing the same because the collaboration protocol does not rely on the channel coding method. Basically, the source encodes the information with the channel code method chosen and sends it to both the relay and the destination. The relay receives the codeword and decodes it to get the information. It will be a matter of the relay to decide whether the decoded information is reliable or not to send it to the destination. If that is the case it re-encodes the information bits to transmit them as another set of parity bits. At the destination we will combine the set of parities obtained from the source and the relay. We just have to take into consideration that, the set of parity coming from the relay will be received from the source only in the collaboration phase, whereas the set of parity coming from the source could be received in either phases.

Multiple relays

As we have seen, when we have the help of the relay we are increasing the spatial diversity of the whole system. With a single relay, we will have two different paths to the destination because the information is sent through two different faded channels. We could tend to think that, by increasing the spatial diversity degree normally we would achieve an enhancement in the performances.

The problem when we have multiple relays is that in the collaborative phase we will have more than one signal arriving to the destination while when we had only one relay there was no doubt about the signal coming from the relay. So, we have to take into consideration that whether we achieve full diversity or not when all the relays are transmitting in the second phase is not so clear.

Two different methods to achieve full diversity while having multiple relays have been proposed. One of them, is based on the sharing of the transmitting time which means that, if we have M relays, we are using $M+1$ phases to transmit all the signal from the different nodes including the source. With such a way of operation, we book a certain time for each relay to transmit and it is clear that we do not have two nodes transmitting at the same time and we can actually achieve full diversity for our system. However, if we want to transmit the same information in a shorter period of time we will have to use a higher rate which means we will be increasing the bandwidth requirements.

For the second method we will use a space-time coding technique which will allow us to use the same two phases. Thus, in the first one, all the relays will be listening to the signal coming from the source and in the second one, once they have decoded the information, they will re-encode it using a space-time code and forward to the destination. This method also achieves full spatial diversity but it requires a more complex space-time coding at the relays which means a longer time to send the parity information to the destination.

5. Results

AWGN channel

At the beginning, we are testing our AWGN system without any Rayleigh fading. We will compare the results with the theoretical curve, the curve we obtain when we have no coding and the curve for a Hamming block code with Belief propagation decoding.

The simulations are repeated until we have 100 blocks in error or until we have sent 10^7 blocks. We are making two different simulations for our system, with a block of 10 bits and with a block of 100 bits. Both of them are run with a relation *EbN0* from 0dB to 8dB.

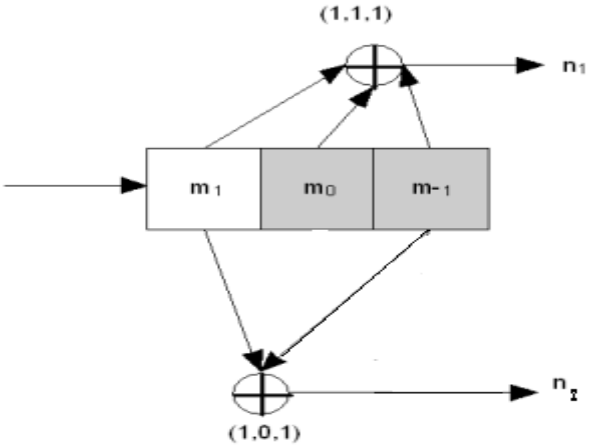


Figure 5.1: convolutional encoder

Normally, there should not be big differences between both of them due to the encoder we have chosen of 3 slots (*L=3*). We will remind the appearance of our encoder followed by the results we have obtained for this case scenario.

Figures

In this first figure we show the performances of the different channel code techniques.

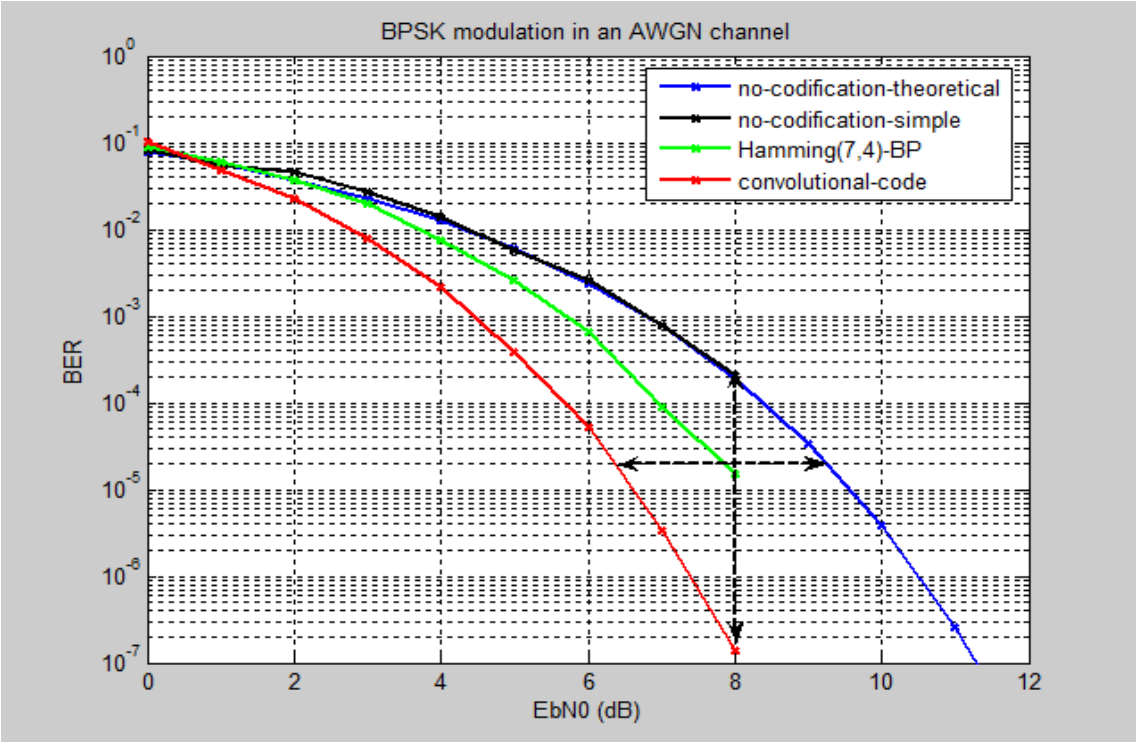


Figure 5.2: BPSK modulation in an AWGN channel for blocks of length 10 bits

As we can see, at the beginning when the ratio *EbN0* is equal to zero, which means that we have the same power of signal and noise, we do not get any advantage from the channel coding. As we increase the power of the signal (we should better say, we increase the ration *EbN0*), we obtain a lot better performances from the fact that we are coding our information and so, we can recover from previous errors. That is a very general behaviour for all channel coding.

In this graph, we are plotting four different curves, one of them, the blue one, is just showed to be compared with the others. It's a theoretical curve which stands as a reference and which is calculated out of the model for a BPSK modulation over an AWGN channel.

Then, we are simulating a BPSK modulation to which we add the Gaussian noise to actually compare with this theoretical curve. This information will be just send through the channel without any channel coding process. As we could imagine, we are following the same performance as for the theoretical one. To decode the received information we just follow a simple deciding process in which we compare the received bits to a threshold, if we are over this value we assume we have transmitted a '0', if we are below we decide a '1' was transmitted.

The other two curves are the interesting ones because we are following a coding process. We wanted to compare the performances of a block code and a convolution code so that is the reason we have chosen a Hamming code to compare to our convolutional scenario.

The only difference between a block code and a convolutional one is that for the last one, the encoded bits depend not only on the current *k* inputs but also on the past input bits. Convolutional codes do not offer more protection against noise than an equivalent block code.

In many cases, they generally offer greater simplicity of implementation over a block code of equal power.

We are using a *Belief Propagation* technique to decode our block code. We are not going to get into the details of this decoding technique; we are using this Hamming block code to make a comparison with the convolutional one that we have indeed implemented.

As we can see the performances of the convolutional code are getting better keeping with the increase of the ratio *EbN0*. We really appreciate the reduction in the BER for a ratio of 8dB where we almost attend a BER of 10^{-7} for the convolutional code whereas for the block code we hardy reach to the 10^{-5}. That means that with the same power we can get to a lot better performances using a convolutional code.

Furthermore, we can talk about power saving when we need to have a certain BER over which our system does not perform well. If that is the case we can see that if we want to attend a BER of $2x10^{-4}$ we will need an *EbN0* ratio of about 6.2dB which is almost 3 dB smaller than the amount of power we would need to transmit the same information without any coding attending the same performances. That is half of the power, so it is really here where we can see the difference of applying channel coding techniques to our information.

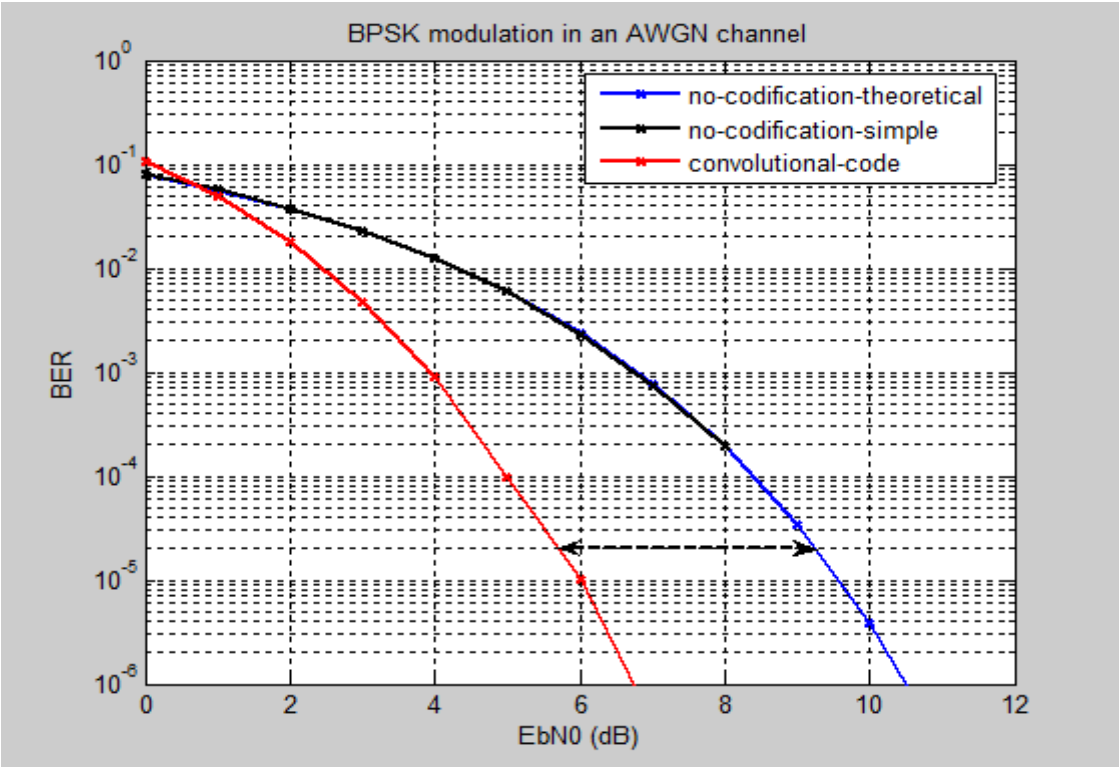


Figure 5.3: BPSK modulation in an AWGN channel for blocks of length 100 bits

For this next figure we just have the same case scenario but this time, we are sending blocks of 100 bits so it took us more time to simulate. The performances are slightly better than for the previous graph as we can see. This time, we are obtaining over 3dB for the same BER constraint. That could be normal at some point because convolutional codes are meant to be working with a continuous bit stream and as we increase the number of bits per block sent we could be obtaining better performances. That will not always be the case; there must be some point in which there are not any differences as we increase the number of bits per block.

AWGN channel with Rayleigh slow fading

Once we have simulated the AWGN channel, we are adding the Rayleigh fading due to multipath that we normally face in a radio channel. In this section, we are going to see the effect on the received data when we are facing a slow fading that affects the same way to all the bits in the block. We can assume that the channel stays the same for a certain period of time (it will change for every block).

In this case, the simulations are repeated until we have 500 blocks in error or until we have sent 10^7 blocks. We are increasing the number of blocks in error because we want to obtain softer and more precise curves. For this case scenario, we are making three different simulations for our system, with a block of 10 bits, with a block of 100 bits and with a 1000 bits block. All of them are run with a relation *EbN0* from 0dB to 30dB.

Figures

We are showing the curve for the convolutional code and the one without any coding to make a comparison between both of them.

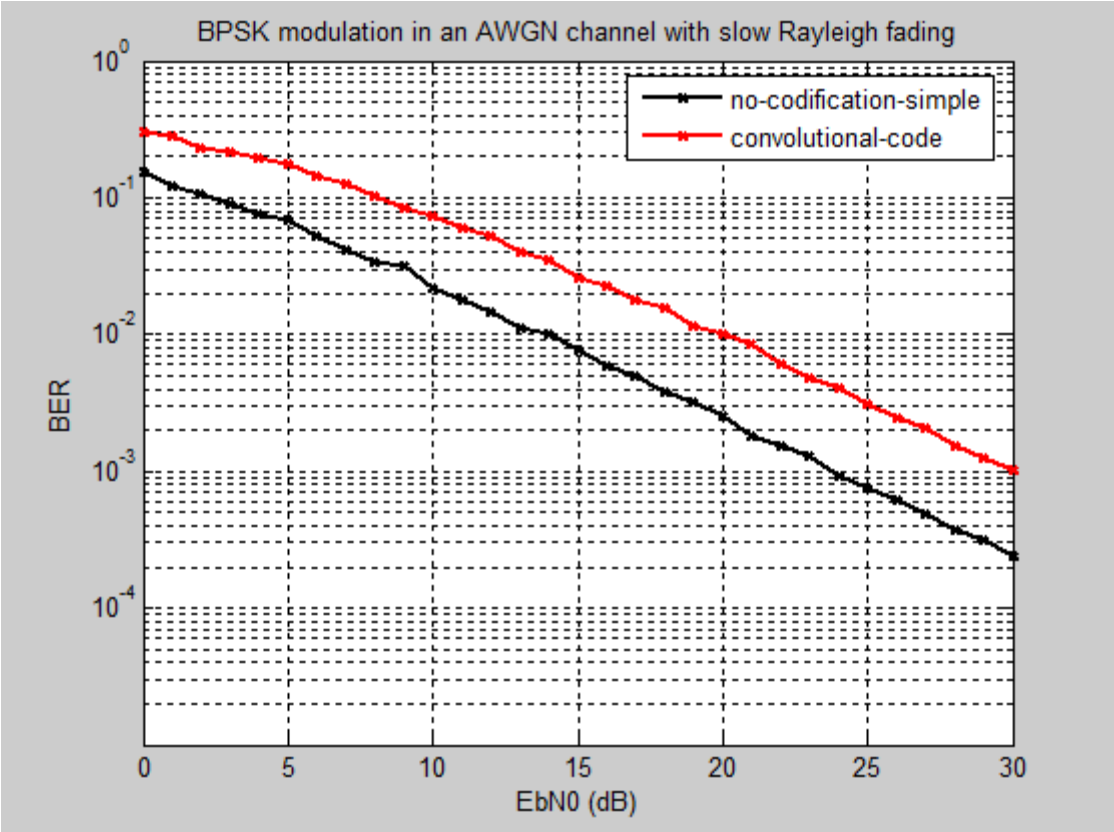


Figure 5.4: BPSK modulation in an AWGN channel with slow Rayleigh fading for blocks of length 10 bits

As we can see in the graph, the performances of the convolutional code in this case are worst than the ones obtain without coding. Both curves follow the same behaviour when we increase the ratio *EbN0*.

This behaviour is perfectly normal as we have the same fading through the whole block of bits. If the channel is bad (means that it has a bad fading characteristics) the received bits are going to be mostly wrong and we will not be able to recover from those wrong bits because the following ones will be affected by the same fading. The power of convolutional coding is no longer useful and the performances are even worse than when we have no coding.

To conclude we have to say that the difference between both curves will be random because of the fading figure chosen for the simulation in case, that means that we could be having a closer approach between both curves.

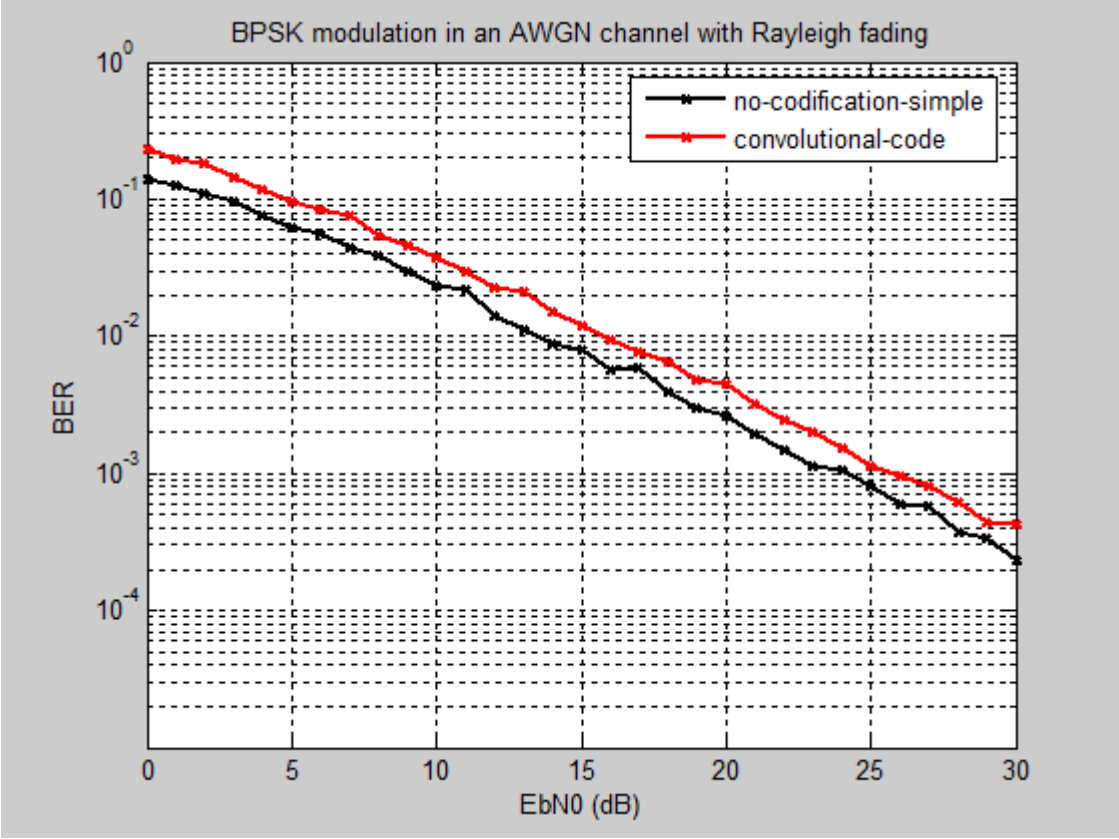


Figure 5.5: BPSK modulation in an AWGN channel with slow Rayleigh fading for blocks of length 100 bits

In this other graph we are simulating the same scenario but in this case, the length of the blocks is of 100 bits. As we can see the behaviour is the same as in the previous graph with a worse performance from the convolutional code. In this case, the distance between both curves is not that big because the length of the blocks is bigger. That means that we have more bits following the same fading value and so, we are more close to the case in which we have no coding. Normally, as we increase the number of bits per block we should be attaining closer curves to the non coded one.

To obtain softer curves, we should have increased the number of blocks in error but we think it is already enough to have 500 blocks in error, which means that we have at least 500 bits in error in total.

For this last section, we are going to show the performance for this same scenario when the length of the block is 1000 bits. Normally, we should obtain a closest pair of curves.

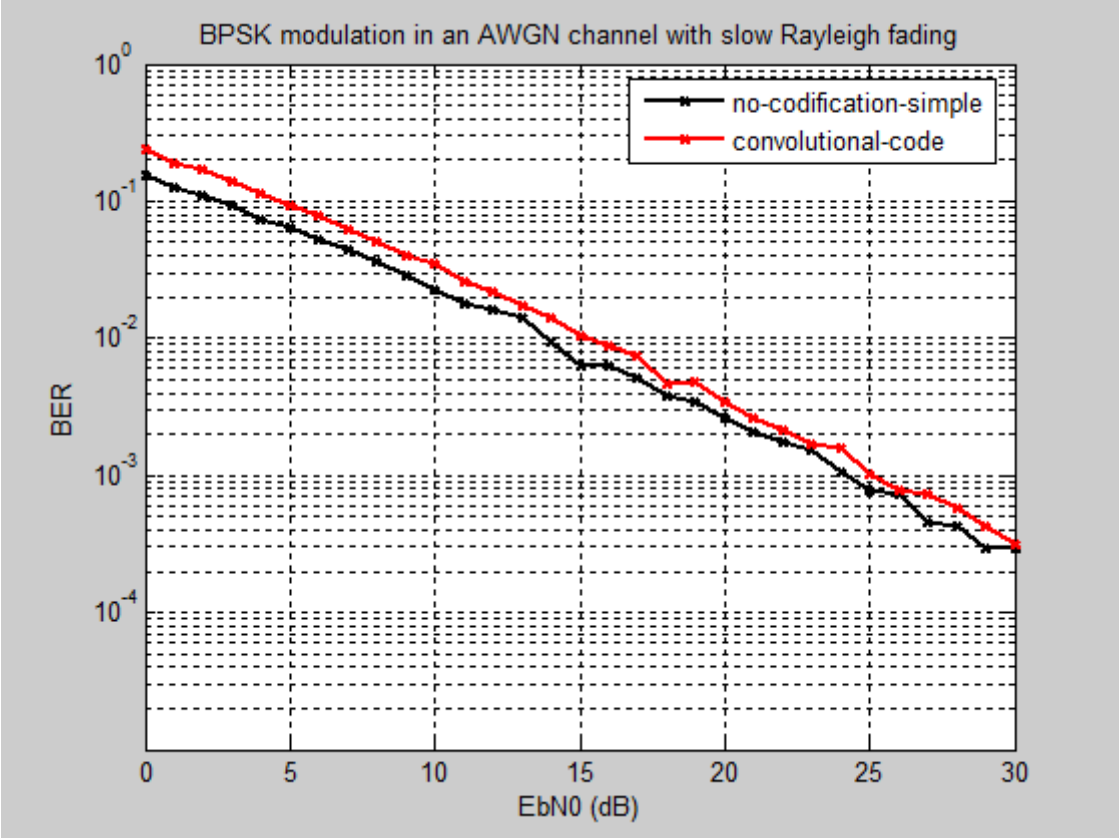


Figure 5.6: BPSK modulation in an AWGN channel with slow Rayleigh fading for blocks of length 1000 bits

As we said before, as we increase the number of bits per block we obtain closer curves, the one which is changing is the convolutional one which is actually approaching to the one with no coding.

AWGN channel with Rayleigh fast fading

We continue the simulations with a fading channel but this time we are simulating a fast fading scenario. In this section, we are going to see the effect on the received data when we are facing a fast fading. That means that we are having a different behaviour of the channel for the different bits of the block. When we had a slow fading, we were suffering a single fade for the whole block but this time we are going to see what happens for different fast fading.

For this case, we are attaining better performances so we have decided to repeat the loop to calculate the BER for the different *EbN0* ratios, 100 times instead of 500 as we did in the previous section. We are having at least 100 bits in error so that is more than enough to obtain reliable performances and the simulation time is average. So, to sum up, the simulations are repeated until we have 100 blocks in error or until we have sent 10^7 blocks. This time, we are making three different simulations for our system to see how it actually responds. The length of the blocks is more or less the same, around 9-10 bits, we could have made the length of the blocks bigger but the simulations would have taken too long.

Figures

In this first figure for this section we are showing the performance of the system when we have a different fading value for every single bit in the block.

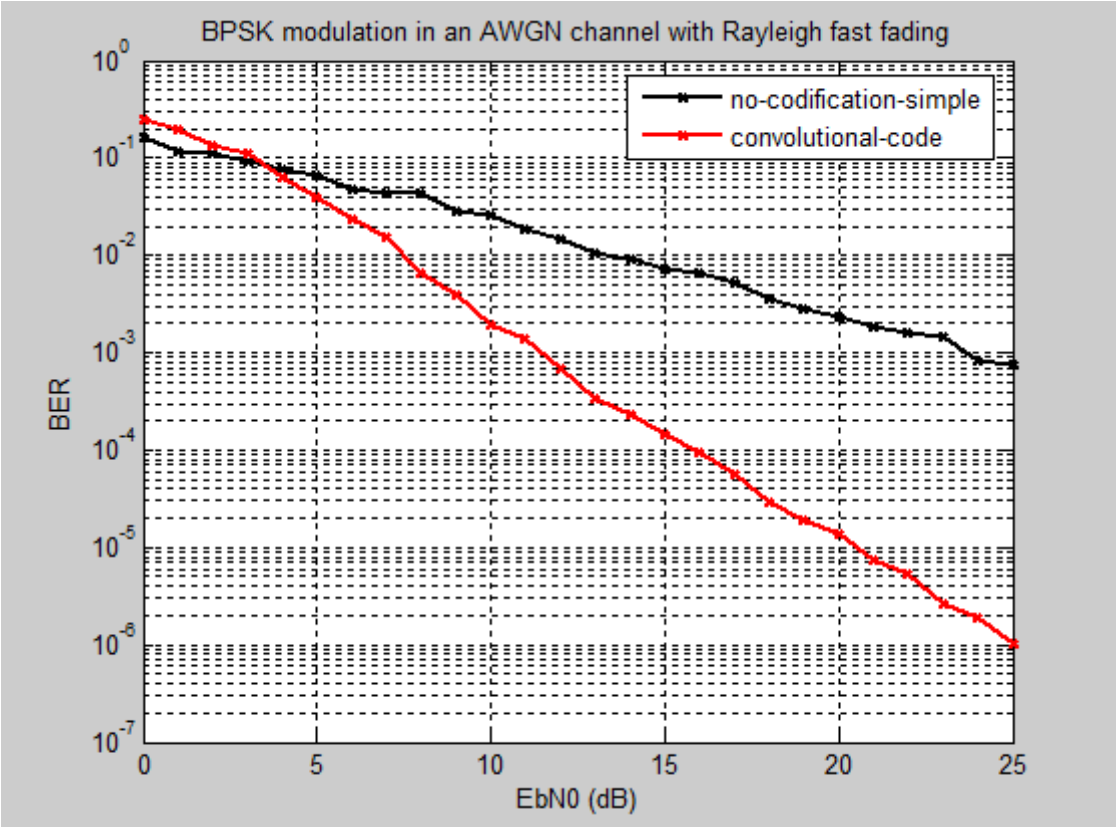


Figure 5.7: BPSK modulation in an AWGN channel with fast Rayleigh fading for blocks of length 10 bits

As we can see the performances are better than in the previous section when we had slow fading. Actually, what is happening is that as we have different fading for every bit, not every random fading value is bad and the convolutional code can recover from one bad bit with the next ones. In this case, the channel coding helps to recover some information that could have been corrupted due to the impairments offered by the channel.

If we make a comparison with the case with AWGN only, we can see that to obtain the same values of BER we need to have higher ratios of *EbN0* (up to 25dB to attain 10^{-6}).

So the main conclusion we can have from this simulation is that when we have more different fading values, we obtain better performances for our system. The thing is, this scenario is not that common because the system doesn't change that much or that quick in time.

Now, we are going to see what happens when we vary a little bit the fading pattern for our block. That means, we are going to simulate the performances of the system when we have three different fading values and five different fading values. The length of the blocks will be 10 bits and we will try to make a simulation of the system with 100 bits per block.

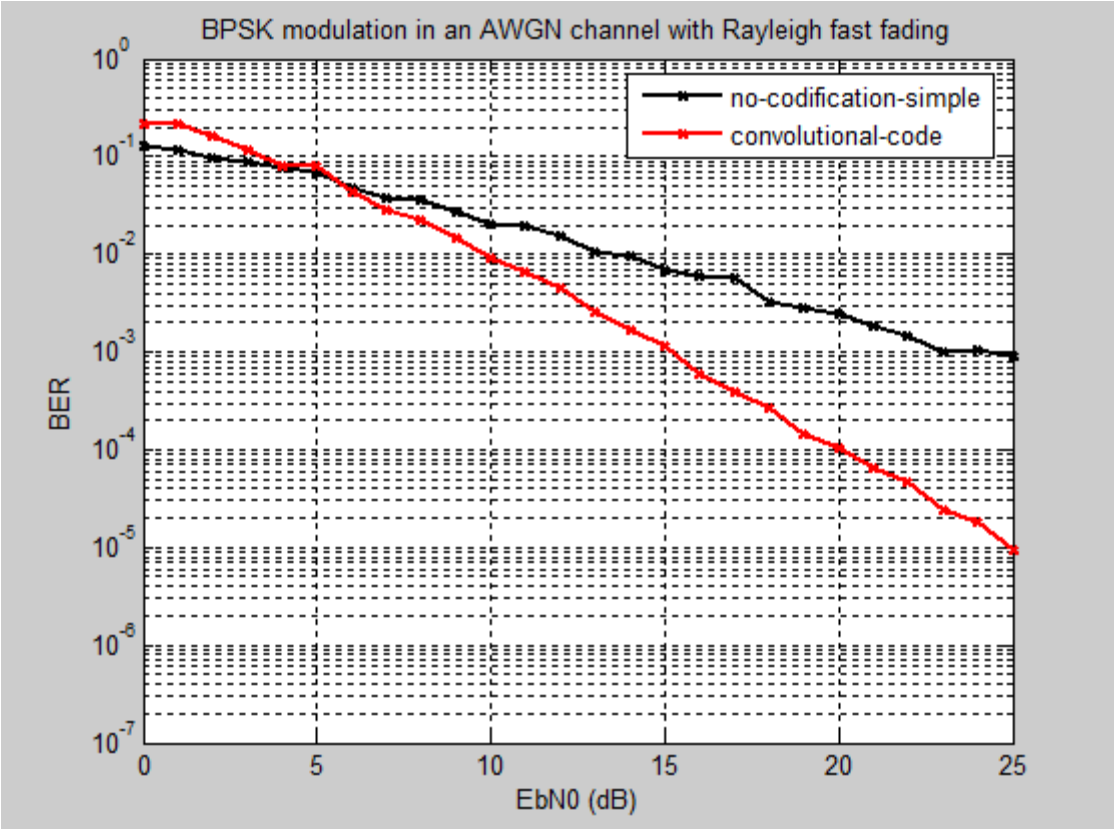


Figure 5.8: BPSK modulation in an AWGN channel with three different Rayleigh fading values for blocks of length 9 bits

As we were expecting, as we decrease the number of fading values we are actually reducing the diversity degrees. In the previous example we had a different fading value for every bit and as we already commented the channel coding can recover one wrong bit from the rest of them which are good.

This time we have a scenario in which the fading values are affecting more than one bit so we are approaching the case of slow fading. That means our performances must be worse than in the case of fast fading as it actually happens. For the same *EbN0* of 25dB we are attaining a BER of 10^{-5} whereas in the previous case in which we had fast fading we were having a BER of 10^{-6} for the same ratio of *EbN0*.

Next, we have made a simulation of a system in which we have five different fading values. It is important to notice that we have blocks of 10 bits so if we have five different fading values we will be approaching the case in which we have a fading value for every single bit (fast fading). Consequently, the performances of this system will be better.

The last simulation will be as well with five different fading values but the length of the block will be 100 bits. As we increase the number of bits per block, we will obtain better performances as the convolutional code will be working more efficiently so we will be able to recover from more errors.

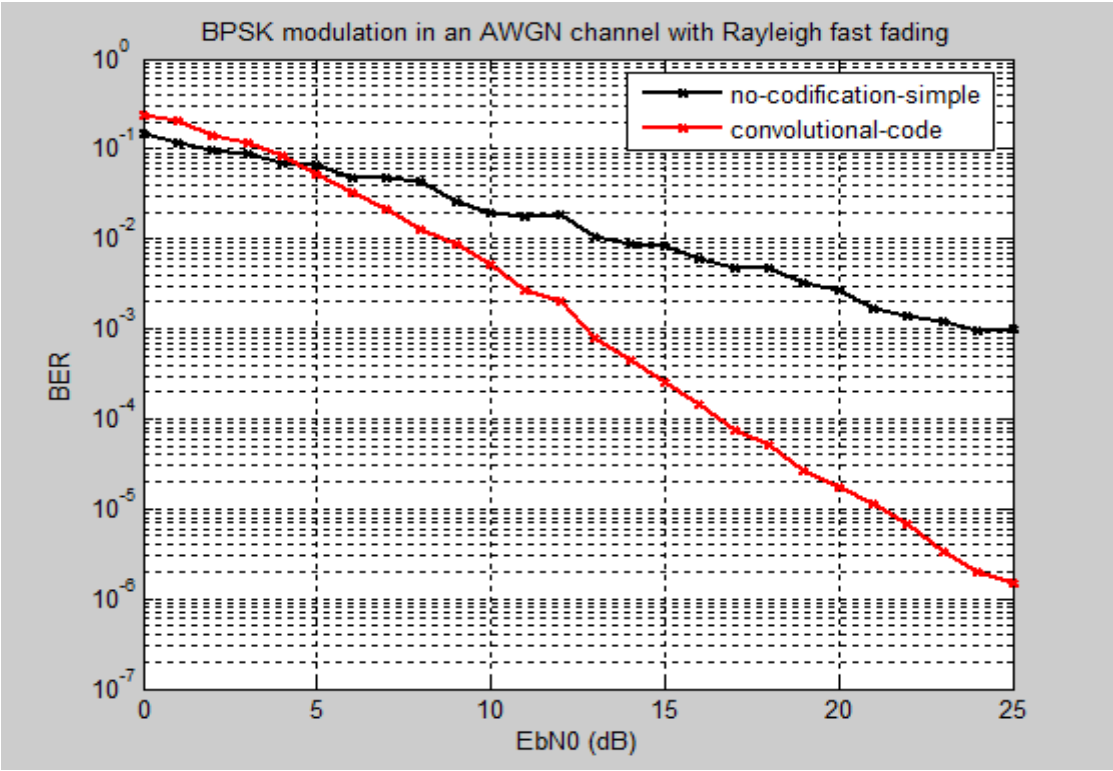


Figure 5.9: BPSK modulation in an AWGN channel with five different Rayleigh fading values for blocks of length 10 bits

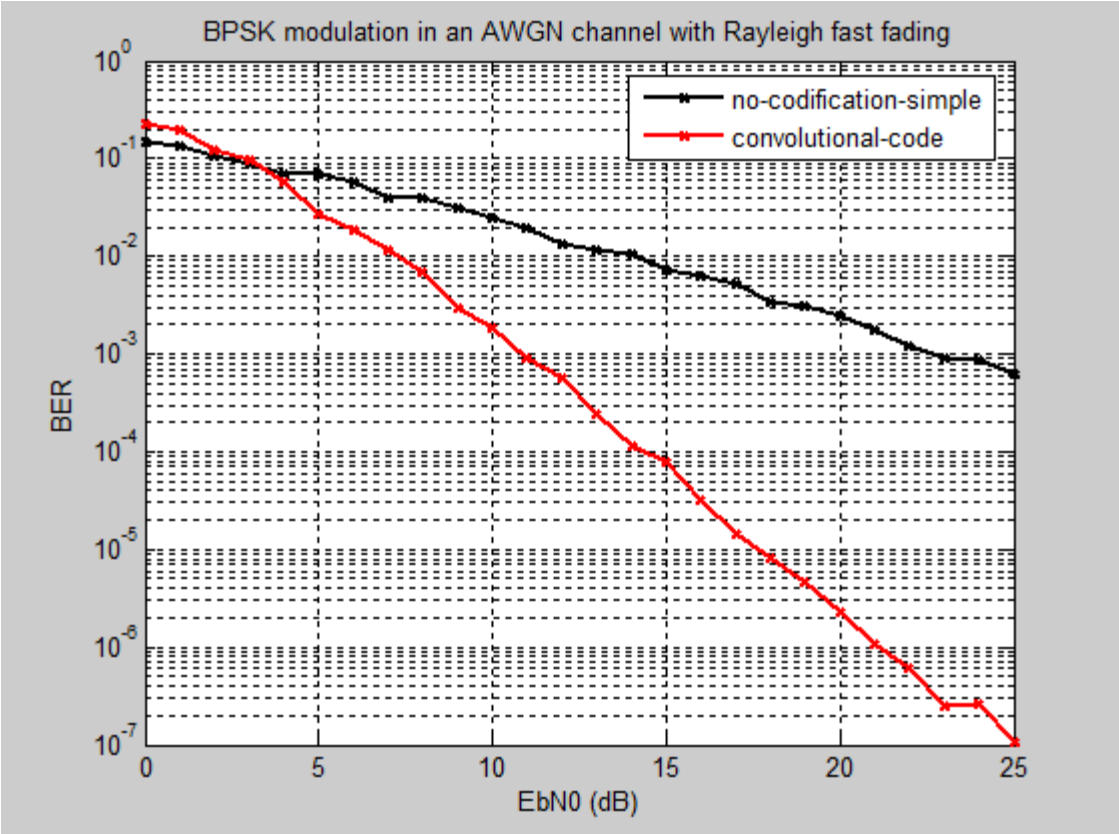


Figure 5.10: BPSK modulation in an AWGN channel with five different Rayleigh fading values for blocks of length 100 bits

As we said, for the case in which we have five different fading coefficients with 10 bits per block, we are attaining almost the same performances for *EbN0* of 25dB than in the first case of this section where we had fast fading. Of course, we have better performance than in the case of three different fading values.

For the 100 bits per block, we are attaining a better BER for every single value of *EbN0* as we can see for example for the ratio of 25dB in which we are having a BER if 10^{-7}.

AWGN channel with Rayleigh slow fading and collaboration

After we have simulated a normal Rayleigh channel between a source and a destination we are going to simulate now the performances of a scenario in which we add the presence of another node acting as a relay.

We are going to take advantage of the fact that we have already simulated the case in which the relay is not collaborating so we have a normal case scenario with a source and a destination. Then, we will compare the performances we have obtained in that case with those obtained in the case that the relay is collaborating. We are going to assume that the relay is decoding the information perfectly so we have no errors transmitted from the relay.

After having done that, we could see the different performances when the relay actually decodes the information coming from the source. In this case, we could have some errors as a result of a poor decoding in the relay. Finally we could define a threshold to differ the case the relay can collaborate from the case the relay actually cannot. That will be a matter of the power of transmission used at the source and the characteristics of the channel between the source and the relay. These two other phases could be part of a future work.

Simulated scenario

To obtain the performances of a collaborative network, we will follow a series of steps in the simulation process.

First, we have to notice that we can make the process of simulation a lot easier by assuming that the relay is able to decode properly and dealing only with the signals at the receiver. Instead, as we have already simulated the convolutional encoder and decoder, we could use them at the relay as well.

So we are simulating a full diversity collaboration scenario following the ‘Decode and Forward’ method that we already explained in the ‘Collaborative Networks’ section. Just to remind some little features about this method, the main difference with other method was that, at the relay, we decoded the information coming from the source to re-encode it again. For our simulations, we are going to consider that we have the same encoder at the source and the relay so it makes things easier.

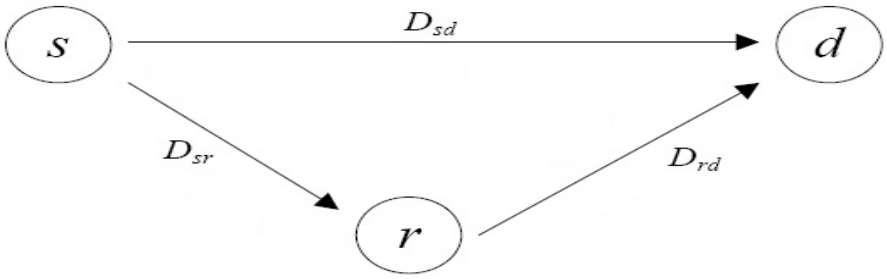


Figure 5.11: simulated scenario

As we can see in the figure, our variables will be the distances between the different nodes of the system. We will make different simulations regarding those variables. As we established in the corresponding chapter, we are making some basic assumptions. The most important ones are that the distance between source and relay is one (D_{sd}=1) and that the path gain between those two nodes is also one (PG_{sd}=1).

So that is the initial step to set the distances between the nodes, basically D_{sr} and D_{rd}. Once we have this, we can calculate the path gain using the following equation:

$$PG_{ij} = PG_{sd}\left(\frac{D_{sd}}{D_{ij}}\right)^{\alpha},$$

As we said we set α=2.

After we have set all the variables, we can start the real process of simulation. First of all, we generate the information bits:

S: a_1 a_2 a_3 a_4

Once we have generated them, we use our convolutional encoder to generate the coded words. We have a convolutional encoder of rate ½ which means that for every single bit of information we will be obtaining two parity bits.

S_d: c_1 c_2 c_3 c_4 c_5 c_6 c_7 c_8

As we can see we have an odd set of parity and an even set of parity after the information has passed the encoder. This is the set of bits we are going to send to the destination. For the relay, we will send only the odd bits generated by the encoder so that means we will have something like:

S_r: c_1 0 c_3 0 c_5 0 c_7 0

The decoding process at the destination will be the same that for the previous case in which we had the source and the destination.

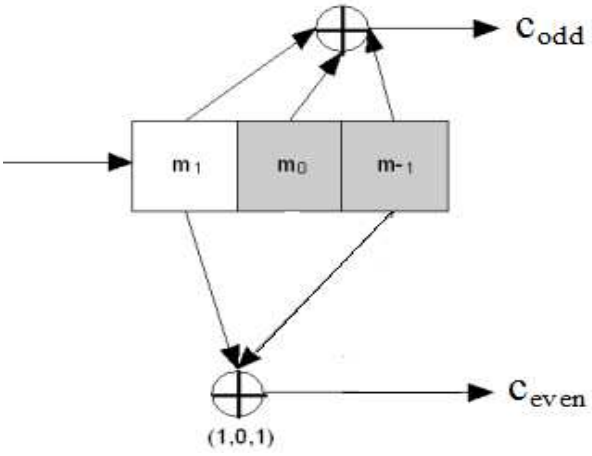


Figure 5.12: convolutional encoder for source and relay

The main difference will be at the relay but, we have to deal with the signal coming from the source (S_r), as if the relay was indeed the destination. Once we have received the signal from the source, we use our Viterbi decoder to obtain the information out of it. As we can imagine we could be obtaining something different to what we have transmitted because of the impairments of the channel between the source and the relay and because of the power of the signal.

Therefore, we obtain our estimated information and we will re-encode it to send it to the destination through a different path which is really the point of all this process. Again, we use our convolutional encoder to generate a set of parity bits. In our simulations, we will assume as we said before, that we are decoding properly so we do not have any errors.

If we remember, for the 'Decode and Forward' method we had two phases, the one in which the relay is listening and receiving the information from the source in a parity way and the collaborative phase in which we were helping the source by sending the information from the relay to the destination. We have already explained the simulation of the listening phase. To simulate the collaborative phase, we generate another set of parity bits.

R: d_1 d_2 d_3 d_4 d_5 d_6 d_7 d_8

However, we have to take into consideration that we have assumed both phases to last the same time (each ½) which means that we will be sending half of the information; in this case we will be sending the even part of the bits generated.

$\mathbf{R_d}$: 0 d_2 0 d_4 0 d_6 0 d_8

Once we have transmitted the information through the relay-destination path we have accomplished our goal of attaining full diversity because we are sending the information bits through two different channels. If we think about it we are sending twice the information from the source and once from the relay so we are actually simulating the case in which the source is transmitting in the second phase.

Now, all we have to do is decoding the information at the destination gathering all the sets of parity that we have received through the different channels. We have to be careful with that because we have to take into consideration the actual fading coefficients and path gains of the channels to properly apply the Viterbi decoder. So at the destination we will have something like that:

D: c_1 c_2+d_2 c_3 c_4+d_4 c_5 c_6+d_6 c_7 c_8+d_8

That will be our received coded word (we cannot forget about the Rayleigh fading, the path gain and the noise but this is a rough example, not a detailed one). Once we have the result of both transmissions added we can apply, once again, our Viterbi decoder to obtain the estimated information bits we transmitted at the very beginning. Hopefully we will obtain better results than the no-collaboration case scenario because we have a more robust system as a result of the help provided by the relay.

At the beginning of the simulations, we will be assuming that we have a slow fading scenario which means that we have the same fading coefficient for the whole block or frame. Basically, we are assuming that the channel stays the same for a certain period of time. Obviously, we will generate a different random coefficient for all the three different channels that we have.

For this case, the simulations are repeated until we have 200 blocks in error or until we have sent 10^7 blocks. We are increasing the number of blocks in error because we want to obtain softer and more precise curves. We will try to make different simulations with different distances between all the nodes of the system to see the different performances. The length of the block or the frame will be 260 bits. All of the simulations are run with a relation *EbN0* from 0dB to 30dB.

Figures

We are going to plot different graphs to show the different performances of the system under different case scenarios.

For the first graph we are assuming the following distances between elements:

D_{sd}=1 (it is always equal to one); D_{sr}=2; D_{rd}=3.

So, as we can see we have the relay at almost the same distance from the source and the destination. However, the relay is farther from the destination than the actual source so this is not a very illustrative real case scenario. The purpose of this simulation is to show that there is no point to have the relay in that position because it will not be helping the source at all. We should expect pretty much the same performances from the case in which we have collaboration and the case in which we do not have the relay helping the source.

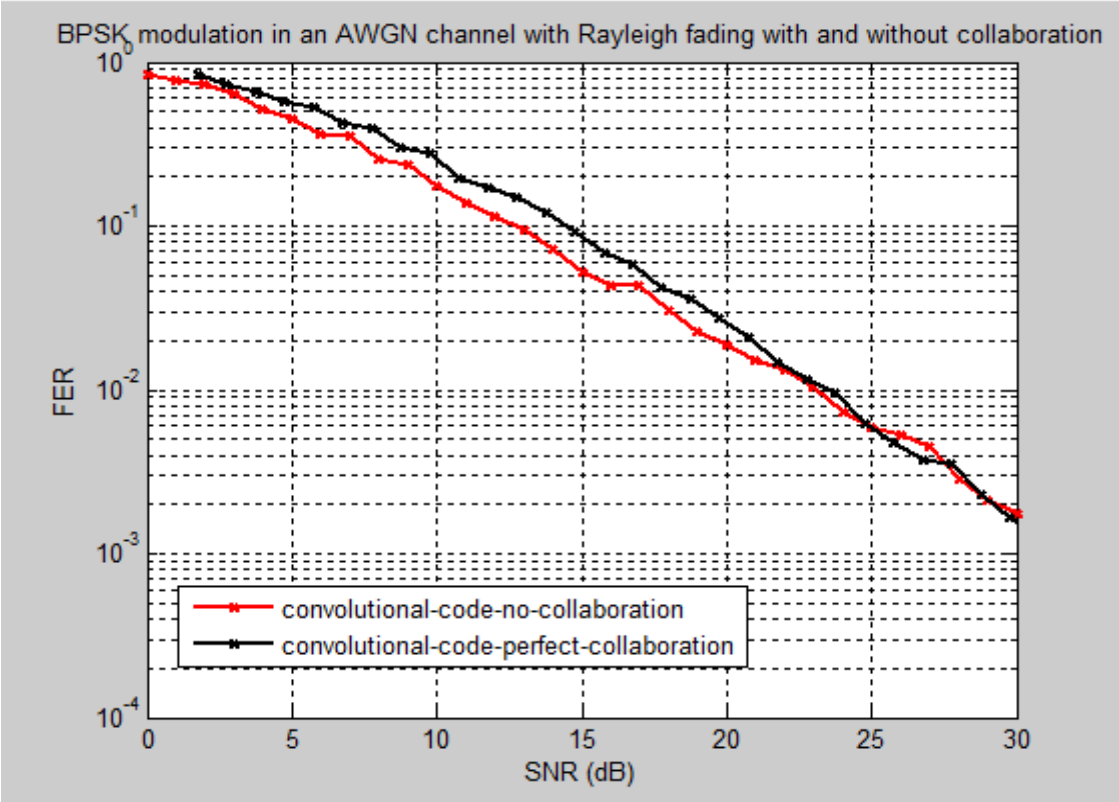


Figure 5.13: BPSK modulation in an AWGN channel with Rayleigh fading and collaboration of the relay far from the source and the destination, for blocks of length 260 bits

As we had expected, there is no big difference between the case in which we are collaborating and the case in which we are not because the relay is further than the source and is not helping that much.

From now on, we are going to place the relay in between the source and the destination and we will make three different simulations, placing the relay near the source, near the destination and in the middle of both to see how it behaves. We will be plotting in the same graph the behaviour of the system when the relay is not helping (no collaboration) and when we assume that the decoding process at the relay is perfect so we are sending data from the relay with no errors (perfect collaboration).

In the first graph we are showing the case in which the relay is closest to the source than to the destination. We are assuming the following distances:

D_{sd}=1 (it is always equal to one); D_{sr}=0.2; D_{rd}=0.8

We can see that scenario like if we had two sources sending the same information to the destination. This same information will travel through two different fading channels with different channel coefficients. Just to remember the channel coefficient was the combination of the path gain and the fading coefficient:

$$G_{ij} = \sqrt{PG_{ij}} H_{ij}$$

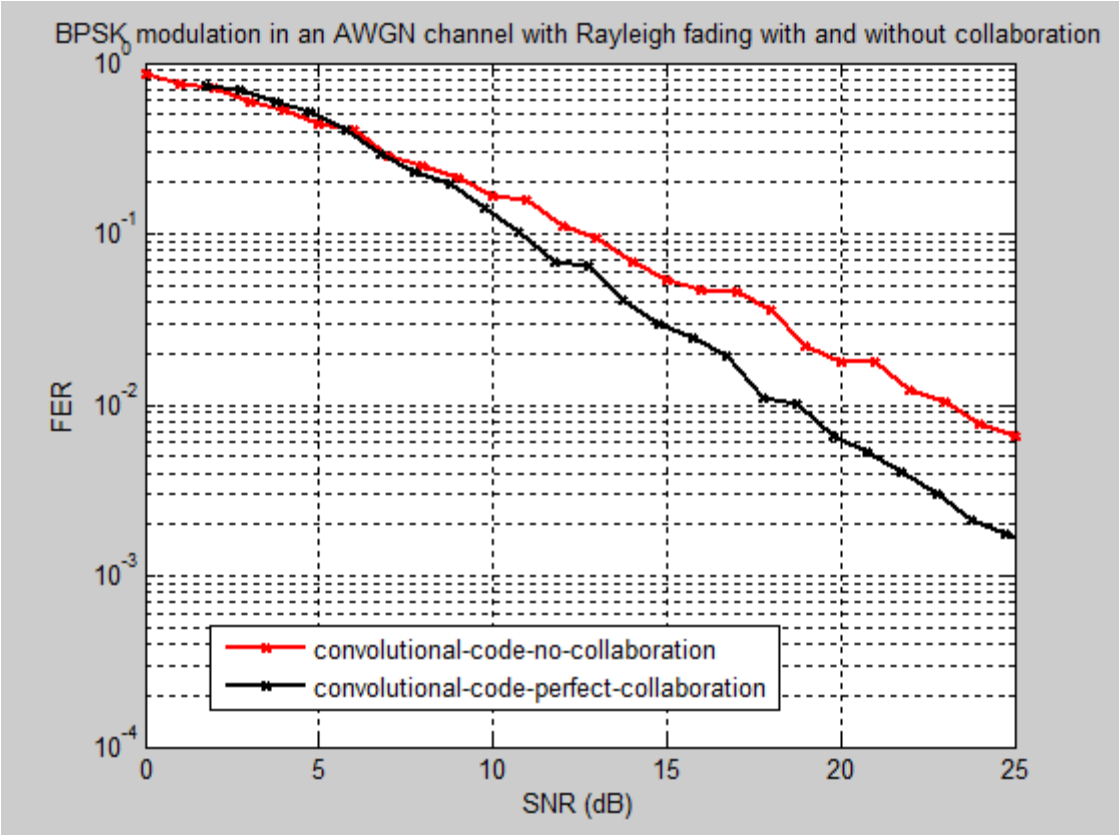


Figure 5.14: BPSK modulation in an AWGN channel with Rayleigh fading and collaboration of the relay near the source, for blocks of length 260 bits

Again as we were expecting, the performance of the system is better than any other previous cases. This time the relay is actually helping the source to get better results at the destination. As we increase the ratio SNR we have bigger differences between both cases.

Now, we are going to place the relay near the destination, so we will have the source transmitting from a further position than the relay but both in different fading channels. We are assuming the following distances:

D_{sd}=1 (it is always equal to one); D_{sr}=0.8; D_{rd}=0.2

In advance, we could think that we will get better performances than in the previous case when we had the relay close to the source.

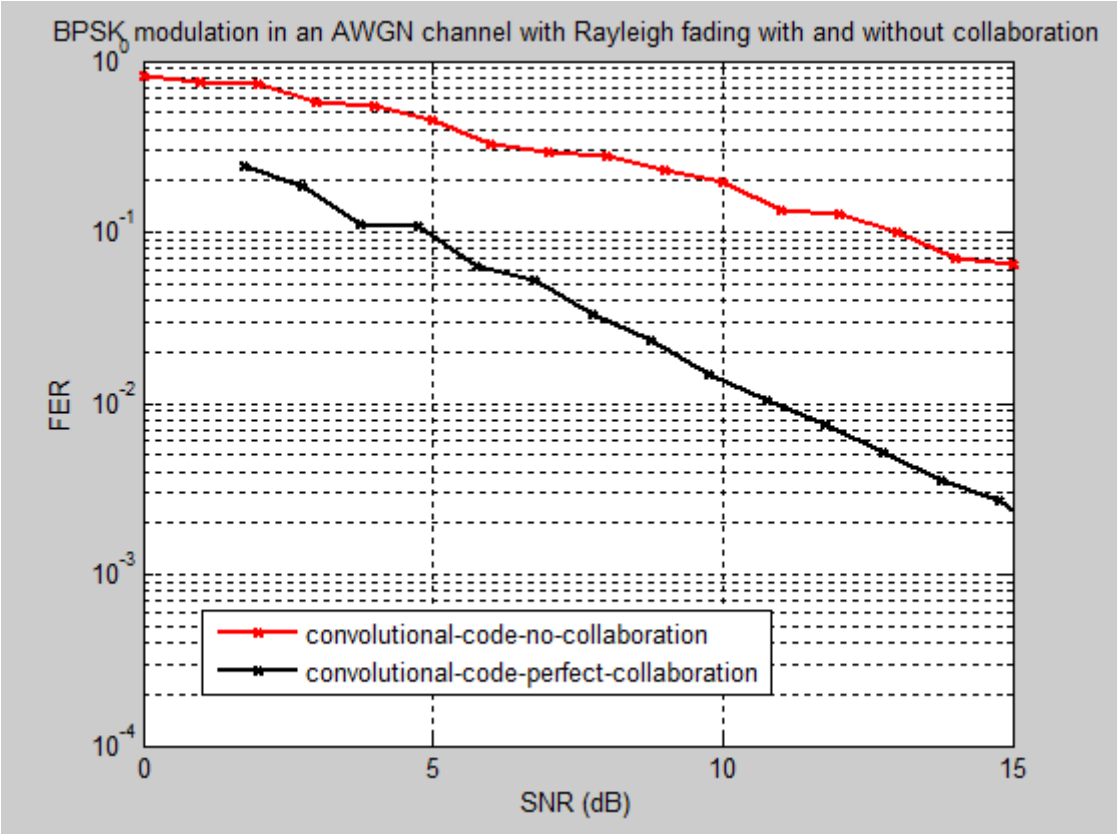


Figure 5.15: BPSK modulation in an AWGN channel with Rayleigh fading and collaboration of the relay near the destination, for blocks of length 260 bits

As we were expecting the results for this simulation are better than for the previous one. The relay is really helping the source because it is very close to the destination. The information coming from the relay is very reliable because of the proximity of the node, the attenuation is very small. We make the simulation up to a SNR of 15dB because it would have taken too long to do it for higher ratios.

The last figure of this set is with the relay place in the middle of the source and the destination. We are assuming the following distances:

D_{sd}=1 (it is always equal to one); D_{sr}=0.5; D_{rd}=0.5

Again, as we can expect in advance, we will be attaining better performances than in the case in which we had the relay close to the source but not better than the previous case in which we had the relay close to the destination.

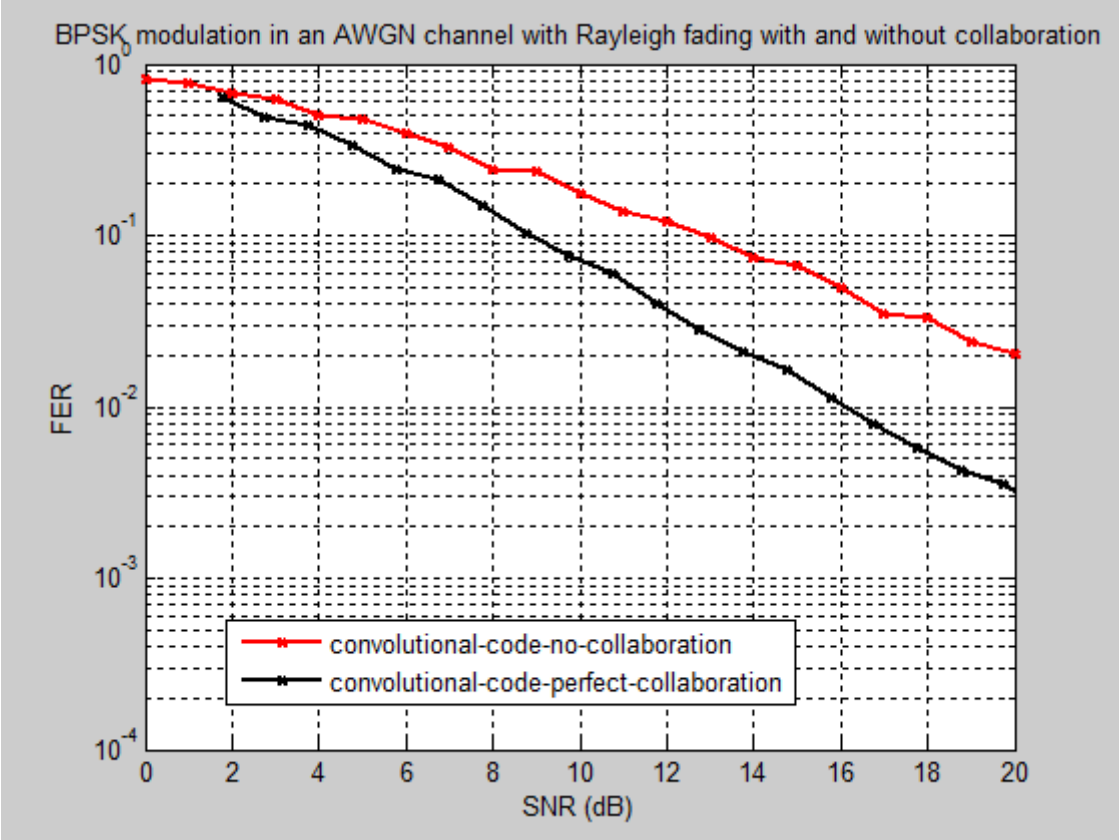


Figure 5.16: BPSK modulation in an AWGN channel with Rayleigh fading and collaboration of the relay in the middle of source and destination, for blocks of length 260 bits

If we compare this graph with the two previous ones, we can notice that the performances are better than the case when we have the relay close to the source but worse than when we have it close to the destination.

We have to take into consideration that once we have collaboration we are using more power, the power to transmit the signal from the source and the power to transmit the signal from the relay. To make a fair comparison with the case in which we do not have collaboration, we have to take that fact into consideration.

For the simulations we are performing, we are sending another set of parity from the relay, in the collaboration phase so the relay is only working half of the time. We will then assume that he is using half of the power the source is using.

```
EbN0_values_linear=10.^(EbN0_values/10);
SNR=3/2*EbN0_values_linear;
SNR_db=10*log10(SNR);
```

As we can see, we create another variable that will be the one we will use in the graphs that take into consideration that we are actually using more power in the whole system (unity for the source and half for the relay so that makes 3/2).

To conclude, we have made a simulation of a system in which we have some collaboration, we are not collaborating all the time anymore. The collaboration of the relay will depend on the channel coefficient between the source and the relay. This last one knows this coefficient and can decide whether to collaborate or not.

In advance we could think the performances are going to be worse than in the case in which we are always collaborating which is what is actually happening.

We are assuming the following distances:

D_{sd}=1 (it is always equal to one); D_{sr}=0.5; D_{rd}=0.5

We want to compare this new graph with the previous one in which we had the same placement of the nodes but we were collaborating all the time.

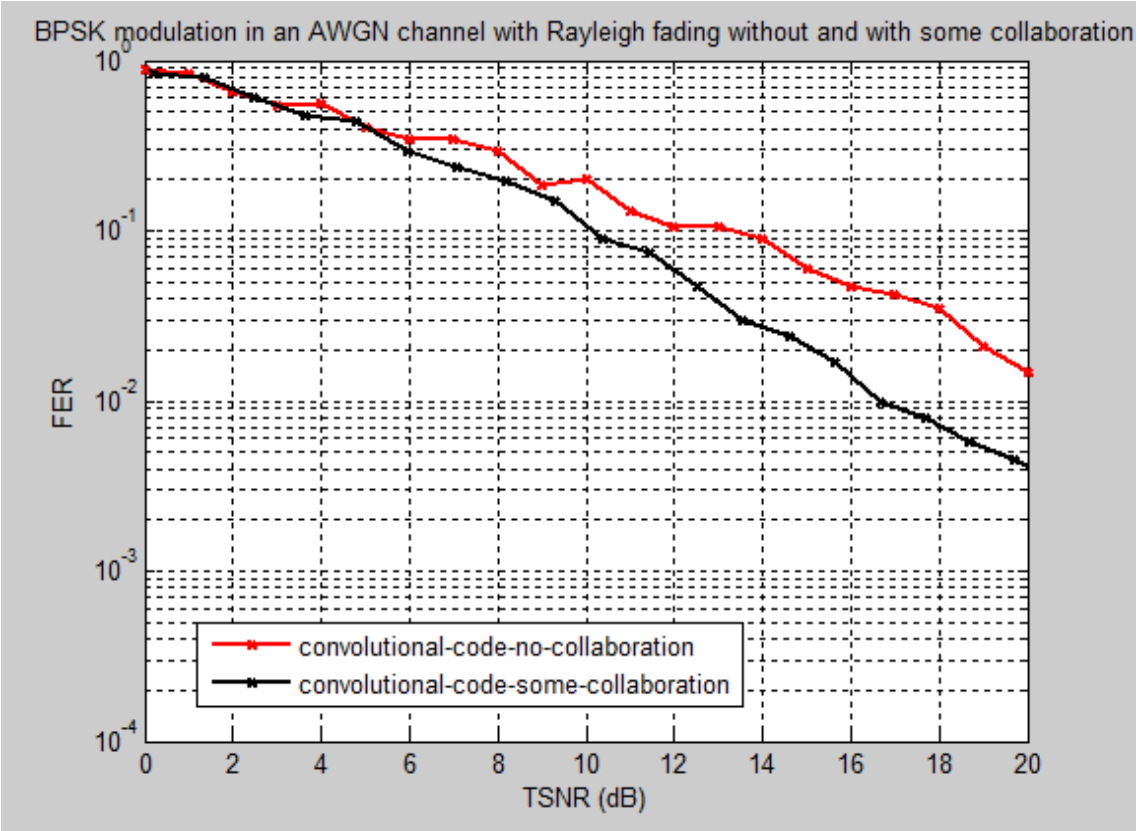


Figure 5.17: BPSK modulation in an AWGN channel with Rayleigh fading and some collaboration of the relay in the middle of source and destination, for blocks of length 260 bits

If we compare it with the previous graph the performances are slightly worse due to the reason that the relay is not always collaborating. We have to take this fact into consideration by defining a threshold (τ) above which we will be considering collaboration.

```
pg_sr = (d_sd/d_sr)^2;
tau = 9.12;
P_no_collab = 1 - exp(-tau./(pg_sr*ebn0));
P_collab = exp(-tau./(pg_sr*ebn0));

FER_some_collab = P_no_collab.*FER_no_collab +
P_collab.*FER_collab;

TSNR = ebn0.*(P_no_collab+ (3/2).*P_collab);
TSNR_db = 10*log10(TSNR);
```

As we can see the probability of collaborating or not only depends on the channel coefficient between source and relay and the actual power used to transmit the signal. Then to make a fair comparison with the case in which we do not have collaboration we define a TSNR which is balanced by the probabilities calculated before

This graph is also rougher because we have simulated with up to 100 blocks in error instead of 200 blocks like we have done for the previous graphs. This is just because we wanted some faster results.

6. Discussion

We have studied the convenience of using collaborative communications in wireless scenarios that are limited in some practical ways. As we have seen, we cannot have the number of antennas that we want in a single device due to the need of portability and movement of this type of wireless devices. This is the main reason we have decided to work in a collaborative way with the other nodes that belong to the system. We can actually use the resources of other idle nodes to help us obtain better performances in the reception of the signal by increasing the spatial diversity for our transmitted signal.

As we have shown in the results section of this work through some simulations of different case scenarios, the use of collaborative communications is something worth implementing in a real world application because of the enhancement of the performances.

Finally, as we have said, it is a very promising topic in wireless communication and it may well be the future of the development in such an area of study.

7. Future work

The collaborative communications topic has been one of the most popular ones for the wireless communication field in the past few years. As a result of that amount of research devoted to that topic, many results have been obtained. However, we have a lot of issues still to unveil. One of the most important ones is the actual practical implementation of those methods simulated in paper. In practice, we no longer have the control over certain variables that we assume to have a certain value in a theoretical scenario.

Another important issue is the operation of the relays. We are assuming all of them have the same timing for the collaborative phase but in practice every single relay will have data knowledge in a different time which means the will finish its listening phase in a different way.

We have considered the collaboration topic in terms of a physical layer method to increase spatial diversity but it can be applied to other layers to achieve other goals. As an example, some recent works have studied the possibility of using collaboration methods to improve secrecy in wireless channels used for the transmission of sensitive information.

Due to a lack of time in the execution of this work we have not simulated the case in which the relay is performing the decoding of the signal coming from the source. We have tried to implement it but the results are not good. We should try to think about the procedure more thoroughly in order to make that part work. Hence, we have made our simulations of the collaborative part considering that the decoding process at the relay was perfect and with no errors. Normally, if we had simulated the decoding process in the relay, we would be making some errors so the performances at the destinations would be worse than in the case of assuming perfect decoding but better than not having collaborations of the relay.

Another step we could implement out of this work would be the time of collaboration for the relay. We could have compared the differences between the case in which the relay is always collaborating in the second phase and the case in which the relay is not collaborating in the second phase due to the behaviour of the channel in that particular moment (the channel coefficient). We have simulated the case in which the relay is always collaborating but in the real life this is not always the case because of the behaviour of the channel. The relay knows the fading coefficient between itself and the source and could chose not to collaborate because it finds the channel is not good to do so. We will assume that above some channel coefficient threshold, there is no point for the relay to collaborate. We made a very simple simulation of this case scenario.

8. References

[1] Tooher, P., & Soleymani, M.R. *Wireless Collaboration: Maximizing Diversity Through Relaying.* Chapter for a paper

[2] Tooher,P., Khoshneviss, H., & Soleymani, M.R. (2007). *Design of collaborative codes achieving space-time diversity*. Paper presented at the IEEE International Conference on Communication (ICC), Glasgow,UK.

[3] Laneman, J.N., Tse, D., & Wornell, G. W. (2004). *Cooperative diversity in wireless networks: Efficient protocols and outage behavior*. IEEE Transactions on Information Theory, 50(12), 3062-3080.

[4] Sendonaris, A., Erkip, E., & Aazhang, B. (2003a). *User cooperation diversity, part I: System description*. IEEE Transactions on Communications, 51(11), 1927-1938.

[5] Sendonaris, A., Erkip, E., & Aazhang, B. (2003b). *User cooperation diversity, part II: System description*. IEEE Transactions on Communications, 51(11), 1939-1948.

[6] Stefanov, A., & Erkip, E. (2004). *Cooperative coding for wireless networks*. IEEE Transactions on Communications, 52(9), 1470-1476.

[7] Hunter, T. E., &Nosratinia, A. (2006). *Diversity through coded cooperation*. IEEE Transactions on Wireless Communications, 5(2), 283-289.

[8] Foschini, G., & Gans, M. (1998). *On limits of wireless communication in a fading environment when using multiple antennas*. Wireless Personal Communications, 6(3), 311-335.

[9] Tarokh, V., Seshadri, N., & Calderbank, A. R. (1998). *Space-time codes for high data rate wireless communication: Performance criterion and code construction*. IEEE Transactions on Information Theory, 44(2), 744-765.

[10] Proakis, J. G. (2000). *Digital Communications*, 4th ed. New York: McGraw-Hill.

[11] Viterbi, A. J. (1971). *Convolutional codes and their performances in communication systems*. IEEE Transactions on Communications, COM-19, 751-772.

[12] Fleming, C. (2003). *A Tutorial on Convolutional Coding with Viterbi Decoding*; Spectrum Application.

[13] Malkamäki, E., Leib, H. (1999). *Evaluating the performance of convolutional codes over block fading channels.* IEEE Transactions on Information Theory, 45(5). 1643-1646.

[14] Liu, E. (2004). *Convolutional coding & Viterbi algorithm*. Postgraduate seminar on radio communications. Helsinki University of technology.

9. Appendix

Matlab files

For the different steps of this work we have used different files but we have followed a similar pattern to accomplish all the simulations.

graph.m

```
function graph
%we are going to be plotting in this graph the different scenarios

%we will be sending a BPSK signal(basically 1's and -1's) that
%goes through an additive gaussian noise channel (AWGN)

%we will design the ideal channel with which we'll make the
%comparaisons values of EbN0 in dBs
EbN0_values = 0:12;


%we plot the BER in channel without any coding (to calculate de BER
%we need to have the amounts or figures in linear units)
%we are plotting the theoretical curve with which we can make a
%comparison to the practical one
ebn0_values = 10.^(EbN0_values/10);
BER = erfc(sqrt(ebn0_values))/2;
semilogy(EbN0_values,BER,'bx-','linewidth', 2)
hold on;


%we plot the BER in an AWGN channel with a BPSK modulation and
%a simple method of decoding
[EbN0_values,BER] = scenario('no_codification','simple','no_fading');
semilogy(EbN0_values,BER,'kx-','linewidth', 2)
hold on;


%we plot the BER in an AWGN channel with a BPSK modulation when we
%have an exact decoding method with a Hamming code(7,4)
[EbN0_values,BER] = scenario('H','BP','no_fading');
semilogy(EbN0_values,BER,'gx-','linewidth', 2)
hold on;


%we plot the BER in an AWGN channel with a BPSK modulation and a
%convolutional encoder and a viterbi decoder
[EbN0_values,BER] =
scenario('convolutional_code','viterbi','no_fading');
semilogy(EbN0_values,BER,'rx-','linewidth', 2)
hold on;

grid on
axis([ 0 12 1e-7 1])
title('BPSK modulation in an AWGN channel')
```

```
legend('no-codification-theoretical', 'no-codification-
simple','Hamming(7,4)-BP', 'convolutional-code');
xlabel('EbN0 (dB)')
ylabel('BER')
```

scenario.m

```
function [EbN0_values,total_BER] =
scenario(codification,decodification,fading)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%codification: indicates the type of codification against errors that
%we have
    %no codification
    %H->Hamming(7,4)
    %convolutional code
%decodification:
    %exact: we suppose we havent coded the bits in the transmitter)
    %BP->Belief Propagation(BP)
    %viterbi
%fading: we're transmitting in a AWGN channel in which we will
%be adding or not fadding)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%we are going to calculate the BER for different values of SNR
%(relation express in dBs)
EbN0_values = 0:12;

%index for the BER vectors and the Pe
index = 0;

%we obtein all the possible cases for the different EbN0
for EbN0 = EbN0_values

    %to calculate the BER we'll be calling the function channel_cod as
    %many times as necessary to calculate it as accurate as possible
    block_Err_sum = 0;
    BER_sum = 0;
    N_blocks = 0;

    %for each value of EbN0 we send as many blocks as necessary until
    %we have 100 blocks in error or until the number of blocks sent is
    %higher than 10^7
    while (block_Err_sum<100 && N_blocks<1e7)
        [BER, block_Err] =
channel_cod(EbN0,codification,channel,decodification,fading);

        %if we have an error in the block, Err_block will be equal to
        %1, which indicates that we have another more block incorrect
        block_Err_sum = block_Err_sum + block_Err;
        BER_sum = BER_sum + BER;
        N_blocks = N_blocks + 1;
    end

    index = index + 1;

    %once we've finished the iterations we'll calculate the values for
    %the probabilities of error in a block and the probability of a
    %error bit (BER)
```

```
    total_N_blocks(index) = N_blocks;
    block_error_Prob(index) = block_Err_sum / N_blocks;

    %we save the different values of the BER in a vector, for each
    %value of the EbN0
    total_BER(index) = BER_sum / N_blocks;
end
```

channel_cod.m

```
function [BER,block_Err] =
channel_cod(EbN0,codification,decodification,fading)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%EbN0: ratio between signal power and noise power
%codification: indicates the type of codification against errors that
%we have
    %no codification
    %H->Hamming(7,4)
    %convolutional code
%decodification:
    %exact: we suppose we havent coded the bits in the transmitter)
    %BP->Belief Propagation(BP)
    %viterbi
%fading: we're transmitting in a AWGN channel in which we will
%be adding or not fadding)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%convolutional code of rate 1/2-->(n,k,L)=(2,1,3)
if strcmp(codification,'convolutional_code')
    %we generate a sequence (a block) of bits to transmit
    %(that will apply to every single method of codification)
    %we actually could be considering that we are only transmitting
    %one bit at a time if we're not introducing any coding
    length_block = 100;
    m = round(rand(1,length_block));

    %generator matrix that we have chosen (we could define another
    %one)
    G = [1 1 1;
        1 0 1];
    %input for the encoder (k bits/sec)
    k=1;

    %we obtein the output of the encoder out of the generator matrix
    n = size(G,1);
    L = size(G,2);

    %we encode the bits of the message through the convolutional
    %encoder that we have already implemented
    c = cnv_encd(G,k,m);

%Hamming coding (7,4)
elseif strcmp(codification,'H')
    %generator matrix
    G = [1 0 0 0 1 1 0;
         0 1 0 0 1 0 1;
         0 0 1 0 0 1 1;
         0 0 0 1 1 1 1];
```

```
    %we obtein the dimensions of the generator matrix that indicates
    %both the length of the message and the length of the code word
    %and we generate a block of bits to transmit
    [length_block,n] = size(G);
    m = round(rand(1,length_block));

    %we code the bits of the message through the code generator matrix
    c = mod(m*G,2);

elseif strcmp(codification,'no_codification')
    %we generate a sequence (a block) of bits to transmit
    length_block = 100;
    m = round(rand(1,length_block));

    %if we are not coding, the random bits and the code word will
    %be the same, we are not applying any matrix or anything
    c = m;
    n = length_block;
end

%we make an antipodal transmission (we transmit 1's and -1's)
%be careful cause, when we're tx a '1' the equivalent in antipodal
%we'll be a '-1' and when we tx a '0' the equivalent will be '1'
%we get to take that into consideration once we're decoding
ct = 1-2*c;

%to compare all the different types of coding we'll have to make that
%the probability of error depends on the rate of codification
%(we take into consideration the sharing of the energy between all
%the code word bits);we work with the energy per coded word
ebn0 = 10^(EbN0/10);
ecn0 = ebn0* (length(m)/length(ct));


%n=sqrt(n0/2)*randn(1,length(ct))
%n=sqrt(1/(2*ecn0))*randn(1,length(ct)
%if Ec=1--> ecn0=Ec/N0--> N0=1/ecn0
sigma = sqrt(1/(2*ecn0));

%we have to distinguish between a normal AWGN channel and a more real
%channel with Rayleigh fading
if strcmp(fading,'fading')
    %we simulate the effects of the Rayleight slow fadding
    %that is what the destination will receive when we take into
    %consideration both, the noise and the fadding (lets say slow
    %fading that affects to each block)
    %take into consideration, that the most general expression for the
    %effects of the channel is rt=sqrt(ec)*h*ct+noise where ec is the
    %energy per bit code
    n = sigma*randn(1,length(ct));
    %the fading has real and imaginary part but we only need the real
    %part, thats the reason we take the module of it; Rayleigh fading
    %has nothing to do with the AWGN
    h = sqrt(0.5)*randn + j*sqrt(0.5)*randn;

    alpha = abs(h);
    rt = alpha.*ct+n;
```

```
elseif strcmp(fading,'no_fading')
    %we simulate the effects of the channel (gaussian channel)
    %that is what we will be receiving, adding the noise to our
    %transmitted signal
    n = sigma*randn(1,length(ct));
    rt = ct + n;
end


%simple method of decodification
if strcmp(decodification,'simple')
    m_est = simple_decoding(rt,length_block,n);

%codification Hamming with BP decodification
elseif strcmp(decodification,'BP')
    m_est = decod_BeliefPropagationHamming(rt,G,EbN0);

%viterbi decodification, we have two different scenarios depending
%on whether we consider a fading channel or not
elseif strcmp(decodification,'viterbi')
    if strcmp(fading,'no_fading')
        m_est = viterbi_soft(G,k,rt);
    elseif  strcmp(fading,'fading')
        m_est = viterbi_soft_fading(G,k,rt,h);
    end
end


%after having decoded the received code word, we compare it with the
%one we had transmited and we'll see if there is any error; if that
%is so, we'll be counting the BER in the frame or the block
%(that means we have to divide the number of errors into the length
%of the block) and we will be counting another block in error
%(we dont care if one or more errors have occured as
%long as we have one error, the block is considered as an error block)
BER = sum(m~=m_est)/length_block;

%if BER>0 is because we have at least one bit in error, which means
%that this block is also in error, so we consider it as a block error
block_Err = BER>0;
```

conv_enc.m

```
function encoder_out=cnv_encd(G,k,encoder_in)
%determines the output sequence of a binary convolutional encoder
%the program assumes zero initial state for the encoder
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%G: generator matrix of the convolutional code with n rows and k*L
%columns (remember L=M+1)
%k: number of bits entering the encoder at each clock cycle
%(number of inputs)
%encoder_in: the binary input sequence; this input sequence
%starts with the first information bit that enters the encoder

%encoder_out: binary output sequence
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%check to see if extra zero-padding is necessary; if the result of
```

```
%the division is different to 0, that means that we have to fill
%with zeros at the end (that applies for k>1 cause if k=1 we wont
%ever have this problem)
if rem(length(encoder_in),k) > 0
    encoder_in = [encoder_in,zeros(size(1:k-
rem(length(encoder_in),k)))];
end

%number of times well have to repeat the procedure to obtein our
%matrix uu that we will be multiplying by the generator matrix at
%the end to obtein the coded word
num_steps = length(encoder_in)/k;

%check the size of matrix G, we check the number of columns that
%must be a multiple of k (remember that the number of columns is
%k*(L+1)). If we have any reminder, that will mean that the size
%of the matrix is not what we were expecting
if rem(size(G,2),k) > 0
    error('Error, G is not of the right size.');
end

%we determine L and n0; remember that the number of columns are
%the number of states multiplied by the number of inputs
%(n=k(M+1)--> M=(n/k)-1); this L is the number of states +1 (L=M+1)
%cause we are using the first memory unit to store the input bits
L = (size(G,2)/k);
n = size(G,1);

%add extra 0's due to assumption of zero initial state for the
%encoder, that means we have 0's in all the state slots
u = [zeros(size(1:(L-1)*k)),encoder_in,zeros(size(1:(L-1)*k))];

%we generate u1, a row vector which has all the different states
%of the encoder (including the input) at every clock cycle
u1 = u(L*k:-1:1);
i=0;
for i=1:num_steps+L-2
    %we concatenate the diferent groups of bits with the ones we
    %had from before
    u1 = [u1,u((i+L)*k:-1:i*k+1)];
end

%matrix whose columns are the contents of the
%convolutional encoder at various clock cycles
%it will have the same number of rows that the number of columns
%of G (that means L*k rows where L=M+1)
uu = reshape(u1,L*k,num_steps+L-1);
%determine the output
encoder_out = reshape(rem(G*uu,2),1,n*(L+num_steps-1));
```

simple_decoding.m

```
function [m_est] = simple_decoding(r,k,n)
%we're gonna be doing a more simple case scenario for the decoder,
%we'll be deciding that we've received a '0' when the correpondant
%bit is below 0 and a '1' with that correspondant bit is over 0
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%r: received wordcode
%k: number of bits per block
%n: number of bits per coded word that number will be the same as
%k if we dont have any coding

%canal: tenemos demodulacion blanda(gaussiano)
%o demodulacion dura(binario simetrico)

%m_est: we get as result the estimated word decoded
%(we will be comparing this word code with the one we transmitted
%to see if an error has occured)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

m_est= zeros(1,k);
%we make a simple loop in which we obtein the estimation of the
message
%transmitted
for i = 1: k
    if r(i)>0
        m_est(i)=0;
    else
        m_est(i)=1;
    end
end
```

decod_BeliefPropagationHamming.m

```
function [m_est] = decod_BeliefPropagationHamming(r,G,EbN0)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%r: received code word
%G: generator matrix
%EbN0: ratio signal to noise

%m_est: palabra codigo estimada
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[K,N] = size(G);

%take into consideration antipodal transmision: 1->-1 and 0->1
pi_aux = logsig(r*4*10^(EbN0/10));
pi_m = [transpose(pi_aux<0.5).*transpose(pi_aux)
transpose(pi_aux<0.5).*transpose(1-pi_aux)] + [transpose(1-
(pi_aux<0.5)).*transpose(pi_aux) transpose(1-
(pi_aux<0.5)).*transpose(1-pi_aux)];

%initial values
%variable to check the convergence of the algorithm
mu_m1_A_aux = [0 0];
mu_m1_A = [1 1];

mu_A_m1 = [0.5 0.5];
mu_A_m2 = [0.5 0.5];
mu_A_m4 = [0.5 0.5];

mu_B_m1 = [0.5 0.5];
mu_B_m3 = [0.5 0.5];
mu_B_m4 = [0.5 0.5];
```

```
mu_C_m2 = [0.5 0.5];
mu_C_m3 = [0.5 0.5];
mu_C_m4 = [0.5 0.5];

mu_c5_A = pi_m(5,:);
mu_c6_B = pi_m(6,:);
mu_c7_C = pi_m(7,:);

for index=1:200

    %we check if the algorithm has already converged
    if (sum(mu_m1_A - mu_m1_A_aux)==0)
        break;
    end
    mu_m1_A_aux = mu_m1_A;

    %information sent from m1 to A
    mu_m1_A = pi_m(1,:).*mu_B_m1;
    mu_m1_A = mu_m1_A./sum(mu_m1_A);

    %information sent from m1 to B
    mu_m1_B = pi_m(1,:).*mu_A_m1;
    mu_m1_B = mu_m1_B./sum(mu_m1_B);

    %information sent from m12 to A
    mu_m2_A = pi_m(2,:).*mu_C_m2;
    mu_m2_A = mu_m2_A./sum(mu_m2_A);

    %information sent from m2 to C
    mu_m2_C = pi_m(2,:).*mu_A_m2;
    mu_m2_C = mu_m2_C./sum(mu_m2_C);

    %information sent from m3 to B
    mu_m3_B = pi_m(3,:).*mu_C_m3;
    mu_m3_B = mu_m3_B./sum(mu_m3_B);

    %information sent from m3 to C
    mu_m3_C = pi_m(3,:).*mu_B_m3;
    mu_m3_C = mu_m3_C./sum(mu_m3_C);

    %information sent from m4 to A
    mu_m4_A = pi_m(4,:).*mu_B_m4.*mu_C_m4;
    mu_m4_A = mu_m4_A./sum(mu_m4_A);

    %information sent from m4 to B
    mu_m4_B = pi_m(4,:).*mu_A_m4.*mu_C_m4;
    mu_m4_B = mu_m4_B./sum(mu_m4_B);

    %information sent from m4 to C
    mu_m4_C = pi_m(4,:).*mu_A_m4.*mu_B_m4;
    mu_m4_C = mu_m4_C./sum(mu_m4_C);

    %we upgrade the values of the different factors
    inc_mu_A_m1 = (mu_m2_A(1,1) - mu_m2_A(1,2))*(mu_m4_A(1,1) -
mu_m4_A(1,2))*(mu_c5_A(1,1) - mu_c5_A(1,2));
    mu_A_m1 = [(1+inc_mu_A_m1)/2 (1-inc_mu_A_m1)/2];
    inc_mu_A_m2 = (mu_m1_A(1,1) - mu_m1_A(1,2))*(mu_m4_A(1,1) -
mu_m4_A(1,2))*(mu_c5_A(1,1) - mu_c5_A(1,2));
```

```
    mu_A_m2 = [(1+inc_mu_A_m2)/2 (1-inc_mu_A_m2)/2];
    inc_mu_A_m4 = (mu_m1_A(1,1) - mu_m1_A(1,2))*(mu_m2_A(1,1) -
mu_m2_A(1,2))*(mu_c5_A(1,1) - mu_c5_A(1,2));
    mu_A_m4 = [(1+inc_mu_A_m4)/2 (1-inc_mu_A_m4)/2];

    inc_mu_B_m1 = (mu_m3_B(1,1) - mu_m3_B(1,2))*(mu_m4_B(1,1) -
mu_m4_B(1,2))*(mu_c6_B(1,1) - mu_c6_B(1,2));
    mu_B_m1 = [(1+inc_mu_B_m1)/2 (1-inc_mu_B_m1)/2];
    inc_mu_B_m3 = (mu_m1_B(1,1) - mu_m1_B(1,2))*(mu_m4_B(1,1) -
mu_m4_B(1,2))*(mu_c6_B(1,1) - mu_c6_B(1,2));
    mu_B_m3 = [(1+inc_mu_B_m3)/2 (1-inc_mu_B_m3)/2];
    inc_mu_B_m4 = (mu_m1_B(1,1) - mu_m1_B(1,2))*(mu_m3_B(1,1) -
mu_m3_B(1,2))*(mu_c6_B(1,1) - mu_c6_B(1,2));
    mu_B_m4 = [(1+inc_mu_B_m4)/2 (1-inc_mu_B_m4)/2];

    inc_mu_C_m2 = (mu_m3_C(1,1) - mu_m3_C(1,2))*(mu_m4_C(1,1) -
mu_m4_C(1,2))*(mu_c7_C(1,1) - mu_c7_C(1,2));
    mu_C_m2 = [(1+inc_mu_C_m2)/2 (1-inc_mu_C_m2)/2];
    inc_mu_C_m3 = (mu_m2_C(1,1) - mu_m2_C(1,2))*(mu_m4_C(1,1) -
mu_m4_C(1,2))*(mu_c7_C(1,1) - mu_c7_C(1,2));
    mu_C_m3 = [(1+inc_mu_C_m3)/2 (1-inc_mu_C_m3)/2];
    inc_mu_C_m4 = (mu_m2_C(1,1) - mu_m2_C(1,2))*(mu_m3_C(1,1) -
mu_m3_C(1,2))*(mu_c7_C(1,1) - mu_c7_C(1,2));
    mu_C_m4 = [(1+inc_mu_C_m4)/2 (1-inc_mu_C_m4)/2];

end

[value,index]=max(mu_A_m1.*mu_B_m1.*pi_m(1,:));
m_est(1) = index-1;
[value,index]=max(mu_A_m2.*mu_C_m2.*pi_m(2,:));
m_est(2) = index-1;
[value,index]=max(mu_B_m3.*mu_C_m3.*pi_m(3,:));
m_est(3) = index-1;
[value,index]=max(mu_A_m4.*mu_B_m4.*mu_C_m4.*pi_m(4,:));
m_est(4) = index-1;
```

viterbi_soft.m

```
function decoder_out=viterbi_soft(G,k,decoder_in)
%the Viterbi decoder for convolutional codes

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%G: generator matrix of the convolutional code with n rows and k*L
%columns
%k: number of bits entering the encoder at each clock cycle (number
%of inputs)
%decoder_in: the binary input sequence; this input sequence
%starts with the first information bit that enters the encoder

%decoder_out: the binary output sequence
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%we obtein the number of bits that will be entering the decoder
%(remember k/n is the rate of the code)
n=size(G,1);

%check the sizes (if the number of columns of G is not a multiple
%of k, that means, the G matrix doesnt correspond to a valid one
```

```
%for this value of k)
if rem(size(G,2),k) ~=0
    error('Size of G and k do not agree');
end

%we check the size of the row vector which has the information
%coming from the channel; it has to be a multiple of n
if rem(size(decoder_in,2),n) ~=0
    error('channel output not of the right size');
end

%L is the number of states +1 (L=M+1) cause we are using the
%first memory unit to store the input bits (see Fig 1)
L=size(G,2)/k;

%different posibilities for the state machine, it will depend
%on the number of states and the number of bits entering the
%encoder
number_of_states=2^((L-1)*k);

%we build some data structures around which the decoder algorithm
%will be implemented we generate state transition matrix, output
%matrix, and input matrix
for j=0:number_of_states-1
    for l=0:2^k-1
        %we call the function nxt_stat (go to the function to see
        %what it does) we have the next state and the actual one
        %in 'memory contents' (it includes the input)
        [next_state,memory_contents]=nxt_stat(j,l,L,k);

        %matrix that shows for each convolutional encoder current
        %state and next state and what input value (0 or 1) would
        %produce the next state, given the current one
        input(j+1,next_state+1)=l;

        %copy of the convolutional encoder 'next state' table
        %(gives the next state given the current state and the
        %input; the dimensions of this table are 2^(L-1)x2^k
        nextstate(j+1,l+1)=next_state;

        %we obtein the output given the current state and the
        %input data (which is stored in memory_contents)
        %(this will be stored in the output table)
        branch_output=rem(memory_contents*G',2);

        %copy of the convolutional encoder output table;
        %the dimensions of this table are 2^(L-1)x2^k
        output(j+1,l+1)=bin2deci(branch_output);
    end
end

%matrix to store the accumulated error metrics for each state
%computed using the add-compare select operation; the dimension
%of this array will be 2^(L-1)x2
state_metric=zeros(number_of_states,2);

%we save the number of sections of our Trellis
depth_of_trellis=length(decoder_in)/n;
```

```
%we create a matrix with the elements of the information getting
%into our system (decoder_in)(thats the info coming from the
%channel, the received coded word)
decoder_in_matrix=reshape(decoder_in,n,depth_of_trellis);

%matrix to store state predecesor history for each encoder state
%for up to depth of Trellis +1 symbol pairs (pairs cause n=2);
%the dimensions are 2^(L-1)xdepth_Trellis+1
survivor_state=zeros(number_of_states,depth_of_trellis+1);

%we'll be dividing the decoding process in to parts thats because
%the initial condition of the encoder is state 0 and the path must
%end up at this same state (the number of steps of the algorithm
%will be the number of bits of our message plus the number of
%encoder memory plus the initial state t=0 which represents the
%initial condition of the encoder)

%start decoding of non-tail channel outputs
for i=1:depth_of_trellis-L+1
    flag=zeros(1,number_of_states);

    %at the beginning, going for example from t=0 to t=1, there
    %are only two possible channel symbols pairs (if k=1) we could
    %have received; thats because we know the convolutional encoder
    %was initialialized to the all zeros state and giving one input
    %bit (1 or 0), there are only two states we could transition
    %to and therefore, two possible outputs of the encoder; thats
    %the reason we make this two different steps, one for the
    %fisrts time instants and the other for the following ones
    if i <= L
        step=2^((L-i)*k);
    else
        step=1;
    end

    %each time that we receive a pair of channel symbols,
    %we're going to compute a metric to measure the distance
    %between what we received and all the possible channel symbols
    %we could have received
    for j=0:step:number_of_states-1

        %we have as many different posibble paths getting to a
        %node as the numbers of bits at the input (if k=1, we will
        %have two different posiible paths getting to each node,
        %we'll have to sa ve the one that has less distance)
        for l=0:2^k-1

            %we save the distance values we compute at each time
            %instant for the paths between the states at the
            %previous time instant and the states at the current
            %time instant
            branch_metric=0;

            %we obtein the binary code for the posible outputs
            %considering the state in which we are
            binary_output=deci2bin(output(j+1,l+1),n);

            %we make an antipodal transmission (we transmit 1's
            %and -1's) be careful cause, when we're tx a '1' the
            %equivalent in antipodal we'll be a '-1' and
```

```
            %when we tx a '0' the equivalent will be '1'we get
            %to take that into consideration once we're decoding
            binary_output_antipodal = 1-2*binary_output;

            %we save the distance between the received channel
            %symbols and the possible channel symbols
            for ll=1:n

branch_metric=branch_metric+abs(decoder_in_matrix(ll,i)-
binary_output_antipodal(ll));
            end

            %if the metric of that state is higher than the
            %cumulated metric of the previous state plus the branch
            %metric we save the lower metric and we set the new
            %survivor state, otherwise we dont get into this if
            %we keep the same survivor state
            if((state_metric(nextstate(j+1,l+1)+1,2) >
state_metric(j+1,1)+branch_metric) || flag(nextstate(j+1,l+1)+1)==0)
                state_metric(nextstate(j+1,l+1)+1,2) =
state_metric(j+1,1)+branch_metric;
                survivor_state(nextstate(j+1,l+1)+1,i+1)=j;
                flag(nextstate(j+1,l+1)+1)=1;
            end

        end
    end
    %we switch the order of the columns in our state_metric matrix;
    %actually what we would be switching is the second row which
    %before that change was the first one
    state_metric=state_metric(:,2:-1:1);
end

%start decoding of the tail channel-outputs
%we do the same procedure for the last time instants of the Trellis
for i=depth_of_trellis-L+2:depth_of_trellis
    flag=zeros(1,number_of_states);
    last_stop=number_of_states/(2^((i-depth_of_trellis+L-2)*k));
    for j=0:last_stop-1
        branch_metric=0;
        binary_output=deci2bin(output(j+1,1),n);
        binary_output_antipodal = 1-2*binary_output;
        for ll=1:n
            branch_metric=branch_metric+abs(decoder_in_matrix(ll,i)-
binary_output_antipodal(ll));
        end
        if((state_metric(nextstate(j+1,1)+1,2) >
state_metric(j+1,1)+branch_metric) || flag(nextstate(j+1,1)+1)==0)
            state_metric(nextstate(j+1,1)+1,2) =
state_metric(j+1,1)+branch_metric;
            survivor_state(nextstate(j+1,1)+1,i+1)=j;
            flag(nextstate(j+1,1)+1)=1;
        end
    end
    state_metric=state_metric(:,2:-1:1);
end

%generate the decoder output from the optimal path; we work backwards
%through the state history table (survivor_state)(shows the surviving
%predecessor states for each state at each time t), for the selected
```

```
%state and for that particular time instant, we select a new state
%which is listed in the state history table as being the best
%predecesor to thatstate (we save the state number of each
%selected state)

%array to store a list of states determined during traceback;
%its a line array which dimensions are 1xdepth_Trellis+1
state_sequence=zeros(1,depth_of_trellis+1);

%the last element of this array must be the state 0, so the
%penultimun will be the state which path leads to that 0 state
%(in our example that path comes from the state 1)
state_sequence(1,depth_of_trellis)=survivor_state(1,depth_of_trellis+1
);

%we obtein the states selected when tracing the path back through
%the survivor state table
for i=1:depth_of_trellis
    state_sequence(1,depth_of_trellis-
i+1)=survivor_state((state_sequence(1,depth_of_trellis+2-
i)+1),depth_of_trellis-i+2);
end

%once we've obteined the path back, using a table that maps state
%transitions to the imputs that caused them, we can recreate the
%original message; the flushig bits at the end are discarded so
%the dimensions of this line vetor are 1xdepth_Trellis-L+1
%(=1xdepth_Trellis-M)
decoder_out_matrix=zeros(k,depth_of_trellis-L+1);
for i=1:depth_of_trellis-L+1

dec_output_deci=input(state_sequence(1,i)+1,state_sequence(1,i+1)+1);
    dec_output_bin=deci2bin(dec_output_deci,k);

    %if k=1 this matrix will be equal to de final result, if not,
    %we'll be having the estimated bits of the message in each time
    %instant per columns, afterwards, we will have to organise them
    %in a single row
    decoder_out_matrix(:,i)=dec_output_bin(k:-1:1)';
end

%we give the result in a row vector
decoder_out=reshape(decoder_out_matrix,1,k*(depth_of_trellis-L+1));
```

nxt_stat.m

```
function
[next_state,memory_contents]=nxt_stat(current_state,input,L,k)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%current_state: those are the states of our encoder (M=L-1)
%input: this is the input, we'll be simulating a simple
%convolutional code with k=1 but if we had a more complex case
%scenario with k=2 for ex. this input would be (0,1,2,3-->0-2^2-1)
%L: number of states plus 1
%k: number of inputs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%we call 'deci2bin' function to obtein the actual state in a binary
%format and the input
binary_state=deci2bin(current_state,k*(L-1));
binary_input=deci2bin(input,k);

%we create the next state from the actual input and all the memory
%slots excepts the last one and we use the 'bin2deci' function to
%obtein it in a decimal format
next_state_binary=[binary_input,binary_state(1:(L-2)*k)];
next_state=bin2deci(next_state_binary);

%we also save the content of the state machine in this particular
%moment(including the input)
memory_contents=[binary_input,binary_state];
```

deci2bin.m

```
function y=deci2bin(x,l)
%decimal to binary
y = zeros(1,l);
i = 1;
while x>=0 & i<=l
y(i)=rem(x,2);
x=(x-y(i))/2;
i=i+1;
end
y=y(l:-1:1);
```

bin2dec.m

```
function y=bin2deci(x)
% binary to decimal
l=length(x);
y=(l-1:-1:0);
y=2.^y;
y=x*y';
```

graph_collaboration

```
function graph_collaboration
%we are going to be plotting in this graph the different scenarios

%we will be sending a BPSK signal(basically 1's and -1's) that
%goes through an additive gaussian noise channel (AWGN) with
%slow Rayleight fading and with collaboration from a relay

%we will design the ideal channel with which we'll make the
%comparaisons values of EbN0 in dBs
EbN0_values = 0:25;


%we plot the FER in an AWGN channel with a BPSK modulation and a
%convolutional encoder and a viterbi decoder when we have no help
%from the relay
[EbN0_values,BER,FER] = scenario_collaboration('no_collaboration');
```

```
semilogy(EbN0_values,FER,'rx-','linewidth', 2)
hold on;


%we plot the FER in an AWGN channel with a BPSK modulation and a
%convolutional encoder and a viterbi decoder when we have perfect
%help from the relay (we assume a perfect decoding process at the
%relay)
[EbN0_values,BER,FER] = scenario_collaboration('collaboration');
EbN0_values_linear = 10.^(EbN0_values/10);
%to make a fair comparaison we have to take into consideration the
%power of the transmission from the relay
SNR = 3/2*EbN0_values_linear;
SNR_db = 10*log10(SNR);
semilogy(SNR_db,FER,'kx-','linewidth', 2)
hold on;


%we plot the FER in an AWGN channel with a BPSK modulation and a
%convolutional encoder and a viterbi decoder when we have perfect
%help from the relay (we assume a perfect decoding process at the
%relay)
[EbN0_values,BER,FER] =
scenario_collaboration('perfect_collaboration');
EbN0_values_linear = 10.^(EbN0_values/10);
%to make a fair comparaison we have to take into consideration the
%power of the transmission from the relay
SNR = 3/2*EbN0_values_linear;
SNR_db = 10*log10(SNR);
semilogy(SNR_db,FER,'kx-','linewidth', 2)
hold on;

grid on
axis([ 0 25 1e-4 1])
title('BPSK modulation in an AWGN channel with Rayleigh fading with
and without collaboration')
legend('convolutional-code-no-collaboration', 'convolutional-code-
perfect-collaboration');
xlabel('SNR (dB)')
ylabel('FER')
```

channel_cod_collaboration.m

```
function [BER,FER] = channel_cod_collaboration(EbN0,collaboration)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%EbN0: ratio between power of signal and power of noise
%collaboration: indicates wheter we have or we do not have
%collaboration of the relay in our system

%BER: probabilidad de error de bit
%FER: boolean that tells you wether the transmitted block or frame
%is in error
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%we define the different variables for the collaborative part in
%here (the different distances between all the devices in the system)
d_sd = 1;
d_sr = 0.5;
```

```
d_rd = 0.5;

%we calculate the path gain for the the source-relay path and
%for the relay-destination path (we have set the path gain for
%the source-destination path to 1)
%the formula to calculate the path gain for the source-relay
%path for example would be: pg_sr=(d_sd/d_rd)^2*pg_sd but the
%pg_sd=1 so we dont need to take it into consideration
pg_sr = (d_sd/d_sr)^2;
pg_rd = (d_sd/d_rd)^2;

%we generate a sequence (a frame) of bits to transmit (that will
%apply to every single method of codification)
%we actually could be considering that we are only transmitting
%one bit at a time if we're not introducing any coding
length_block = 260;
m = round(rand(1,length_block));

%convolutional code of rate 1/2-->(n,k,L)=(2,1,3)
%generator matrix that we have chosen (we could define another one)
G = [1 1 1;
    1 0 1];

%input for the encoder (k bits/sec)
k=1;

%we obtein the output of the encoder out of the generator matrix
n = size(G,1);
L = size(G,2);

%we encode the bits of the message through the convolutional encoder
%that we have already implemented
c = cnv_encd(G,k,m);

%we make an antipodal transmission (we transmit 1's and -1's)
%be careful cause, when we're tx a '1' the equivalent in antipodal
%we'll be a '-1' and when we tx a '0' the equivalent will be '1'
%we get to take that into consideration once we're decoding
ct = 1-2*c;

%to compare all the different types of coding we'll have to make
%that the probability of error depends on the rate of codification
%(we take into consideration the sharing of the energy between all
%the code word bits);we work with the energy per coded word
ebn0 = 10^(EbN0/10);
ecn0 = ebn0* (length(m)/length(ct));

%n=sqrt(n0/2)*randn(1,length(ct))
%n=sqrt(1/(2*ecn0))*randn(1,length(ct)
%if Ec=1--> ecn0=Ec/N0--> n0=1/ecn0
sigma = sqrt(1/(2*ecn0));

%we are simulating a real channel with Rayleigh fading
%we simulate the effects of the Rayleight slow fadding
%that is what the destination will receive when we take into
%consideration both, the noise and the fadding (lets say slow
%fading that affects to each block)
%take into consideration, that the most general expression for the
%effects of the channel is rt=sqrt(ec)*h*ct+noise where ec is the
```

```
%energy per bit code
n_sd = sigma*randn(1,length(ct));

%the fading has real and imaginary part but we only need the real
%part, thats the reason we take the module of it; Rayleigh fading
%has nothing to do with the AWGN
h_sd = sqrt(0.5)*randn+j*sqrt(0.5)*randn;
alpha_sd = abs(h_sd);
rt_dest = alpha_sd*ct+n_sd;

%if we have collaboration we also have to send the information to
%the relay for it to decode it, re-encode it and send it to the
%destination as well
if strcmp(collaboration,'collaboration')

    %we generate the noise samples for relay (the noise coefficient
    %is particular of every reception node)
    n_sr = sigma*randn(1,length(ct));

    %we generate the fading coefficient for the source-relay path
    h_sr = sqrt(0.5)*randn+j*sqrt(0.5)*randn;
    alpha_sr = abs(h_sr);

    %our information sent to the relay would have this shape but we
    %are actually sending only the odd set of bits to the relay;
    %remember to add the path gain factor to the information sent
    rt_sr = (sqrt(pg_sr)*alpha_sr)*ct+n_sr;

    %we want to transmit the odd set of parity to the relay so we
    %create another set of rt's (we need to have 0's in the even
    %positions)
    rt_rel = zeros(1,length(rt_sr));
    for i=1:length(rt_rel)
        if(mod(i,2)==1)
            rt_rel(i) = rt_sr(i);
        end
    end
end


%we start the decoding process

%viterbi decodification, we are considering we have a fading
%channel
%we decode the information for the destination
m_est = viterbi_soft_fading(G,k,rt_dest,h_sd);

%if we have collaboration we have to decode the information
%transmited from the source, re-encode it and send it to the
%destination
if strcmp(collaboration, 'collaboration') || strcmp(collaboration,
'perfect_collaboration')

    %we check if we are assuming that we have a perfect decoding
    %process at the relay,if thats the case we do not need to decode
    %the information, we just assume that the relay decode it
    %perfectly so we take 'm' again to encode
    if strcmp(collaboration, 'perfect_collaboration')
        c_rel = cnv_encd(G,k,m);
```

```
    else
        %we decode the information at the relay (we put 0's for the
rest of
        %coefficients which are not necessary in this case but for the
p_sd=1
        %always)
        m_est_rel =
viterbi_soft_fading_collaboration(G,k,rt_rel,'relay',h_sr,0,0,pg_sr,1,
0);


        %once we have it decode it, we have to reencode it again
        %(we assume we have the same encoder at the source and at
        %the relay);we encode the bits of the message through the
        %convolutional encoder that we have already implemented
        c_rel = cnv_encd(G,k,m_est_rel);
    end

    %remember antipodal transmission
    ct_rel = 1-2*c_rel;

    %we generate the fading coefficient for the relay-destination path
    h_rd = sqrt(0.5)*randn+j*sqrt(0.5)*randn;

    %once we have the fading coefficient for the relay-destination
    %path we have to take into consideration that for the recived code
    %word at the destination we have to create another fading
    %coefficient which will be the sum of both (we have to be careful
    %because that fading coefficient is complex and we cannot add the
    %modules like that)

    %we are going to create the simulated code word received at the
    %destination when the relay is collaborating (for the odd bits
    %we will have like no collaboration and for the even bits we will
    %create another structure, taking into consideration that we have
    %to add the fading coefficients and not the modules)
    rt_dest_2 = zeros(1,length(rt_dest));
    for i=1:length(rt_dest)
        %we have to distinguish between the odd and the even bits
        if(mod(i,2)==1)
            rt_dest_2(i) = rt_dest(i);
        %for the even coefficients (be careful with the fading
        %coeficient)
        elseif(mod(i,2==0))

            if ct(i)==1 && ct_rel(i)==1
                alpha_srd_ct = abs(h_sd + sqrt(pg_rd)*h_rd);
            elseif ct(i)==1 && ct_rel(i)==-1
                alpha_srd_ct = h_sd - sqrt(pg_rd)*h_rd;
                if(-pi/2<angle(alpha_srd_ct)<pi/2)
                    alpha_srd_ct = abs(alpha_srd_ct);
                else
                    alpha_srd_ct = -abs(alpha_srd_ct);
                end
            elseif ct(i)==-1 && ct_rel(i)==1
                alpha_srd_ct = -h_sd + sqrt(pg_rd)*h_rd;
                if(-pi/2<angle(alpha_srd_ct)<pi/2)
                    alpha_srd_ct = abs(alpha_srd_ct);
                else
```

```
                    alpha_srd_ct = -abs(alpha_srd_ct);
                end
            elseif ct(i)==-1 && ct_rel(i)==-1
                alpha_srd_ct = -abs(-h_sd - sqrt(pg_rd)*h_rd);
            end

            rt_dest_2(i) = alpha_srd_ct + n_sd(i);
        end
    end

    %we have to decode the resulting codeword at the destination
    m_est_2 =
viterbi_soft_fading_collaboration(G,k,rt_dest_2,'destination',0,h_sd,h
_rd,0,1,pg_rd);
end

%after having decoded the received code word, we compare it with the
%one we had transmited and we'll see if there is any error; if that
%is so, we'll be counting the BER in the frame or the block
%(that means we have to divide the number of errors into the length
%of the block) and we will be counting another block in error
%(we dont care if one or more errors have occured as long as we
%have one error, the block is considered as an error block)
if strcmp(collaboration, 'no_collaboration')
    BER = sum(m~=m_est)/length_block;

    %if BER>0 is because we have at least one bit in error, which
means that
    %this frame is also in error, so we consider it as a block error
    FER = BER>0;

%if the relay was indeed collaborating, we will have another BER and
%another FER
elseif strcmp(collaboration, 'collaboration') || strcmp(collaboration,
'perfect_collaboration')
    BER = sum(m~=m_est_2)/length_block;
    FER = BER>0;
end
```

viterbi_soft_fading.m

```
function decoder_out=viterbi_soft_fading(G,k,decoder_in,h)
%the Viterbi decoder for convolutional codes

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%G: generator matrix of the convolutional code with n rows and k*L
%columns
%k: number of bits entering the encoder at each clock cycle (number
%of inputs)
%decoder_in: the binary input sequence; this input sequence
%starts with the first information bit that enters the encoder
%h: Rayleigt fading which is known for the receiver

%decoder_out: the binary output sequence
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%we obtein the number of bits that will be entering the decoder
%(remember k/n is the rate of the code)
```

```
n=size(G,1);

%check the sizes (if the number of columns of G is not a multiple
%of k, that means, the G matrix doesnt correspond to a valid one
%for this value of k)
if rem(size(G,2),k) ~=0
    error('Size of G and k do not agree');
end

%we check the size of the row vector which has the information
%coming from the channel; it has to be a multiple of n
if rem(size(decoder_in,2),n) ~=0
    error('channel output not of the right size');
end

%L is the number of states +1 (L=M+1) cause we are using the
%first memory unit to store the input bits (see Fig 1)
L=size(G,2)/k;

%different posibilities for the state machine, it will depend
%on the number of states and the number of bits entering the
%encoder
number_of_states=2^((L-1)*k);

%we build some data structures around which the decoder algorithm
%will be implemented we generate state transition matrix, output
%matrix, and input matrix
for j=0:number_of_states-1
    for l=0:2^k-1
        %we call the function nxt_stat (go to the function to see
        %what it does) we have the next state and the actual one
        %in 'memory contents' (it includes the input)
        [next_state,memory_contents]=nxt_stat(j,l,L,k);

        %matrix that shows for each convolutional encoder current
        %state and next state and what input value (0 or 1) would
        %produce the next state, given the current one
        input(j+1,next_state+1)=l;

        %copy of the convolutional encoder 'next state' table
        %(gives the next state given the current state and the
        %input; the dimensions of this table are 2^(L-1)x2^k
        nextstate(j+1,l+1)=next_state;

        %we obtein the output given the current state and the
        %input data (which is stored in memory_contents)
        %(this will be stored in the output table)
        branch_output=rem(memory_contents*G',2);

        %copy of the convolutional encoder output table;
        %the dimensions of this table are 2^(L-1)x2^k
        output(j+1,l+1)=bin2deci(branch_output);
    end
end

%matrix to store the accumulated error metrics for each state
%computed using the add-compare select operation; the dimension
%of this array will be 2^(L-1)x2
state_metric=zeros(number_of_states,2);
```

```
%we save the number of sections of our Trellis
depth_of_trellis=length(decoder_in)/n;

%we create a matrix with the elements of the information getting
%into our system (decoder_in)(thats the info coming from the
%channel, the received coded word)
decoder_in_matrix=reshape(decoder_in,n,depth_of_trellis);

%matrix to store state predecesor history for each encoder state
%for up to depth of Trellis +1 symbol pairs (pairs cause n=2);
%the dimensions are 2^(L-1)xdepth_Trellis+1
survivor_state=zeros(number_of_states,depth_of_trellis+1);

%we'll be dividing the decoding process in to parts thats because
%the initial condition of the encoder is state 0 and the path must
%end up at this same state (the number of steps of the algorithm
%will be the number of bits of our message plus the number of
%encoder memory plus the initial state t=0 which represents the
%initial condition of the encoder)

%start decoding of non-tail channel outputs
for i=1:depth_of_trellis-L+1
    flag=zeros(1,number_of_states);

    %at the beginning, going for example from t=0 to t=1, there
    %are only two possible channel symbols pairs (if k=1) we could
    %have received; thats because we know the convolutional encoder
    %was initialialized to the all zeros state and giving one input
    %bit (1 or 0), there are only two states we could transition
    %to and therefore, two possible outputs of the encoder; thats
    %the reason we make this two different steps, one for the
    %fisrts time instants and the other for the following ones
    if i <= L
        step=2^((L-i)*k);
    else
        step=1;
    end

    %each time that we receive a pair of channel symbols,
    %we're going to compute a metric to measure the distance
    %between what we received and all the possible channel symbols
    %we could have received
    for j=0:step:number_of_states-1

        %we have as many different posibble paths getting to a
        %node as the numbers of bits at the input (if k=1, we will
        %have two different posiible paths getting to each node,
        %we'll have to sa ve the one that has less distance)
        for l=0:2^k-1

            %we save the distance values we compute at each time
            %instant for the paths between the states at the
            %previous time instant and the states at the current
            %time instant
            branch_metric=0;

            %we obtein the binary code for the posible outputs
            %considering the state in which we are
```

```
            binary_output=deci2bin(output(j+1,l+1),n);

            %we make an antipodal transmission (we transmit 1's
            %and -1's) be careful cause, when we're tx a '1' the
            %equivalent in antipodal we'll be a '-1' and
            %when we tx a '0' the equivalent will be '1'we get
            %to take that into consideration once we're decoding
            binary_output_antipodal = 1-2*binary_output;

            %we save the distance between the received channel
            %symbols and the possible channel symbols
            for ll=1:n
                %we know how is the channel in terms of fading so
                %we can compensate it somehow by comparing the
                %received signal with the different posibilities
                %scaled by the Rayleigh fading

branch_metric=branch_metric+abs(decoder_in_matrix(ll,i)-
abs(h)*binary_output_antipodal(ll));
            end

            %if the metric of that state is higher than the
            %cumulated metric of the previous state plus the branch
            %metric we save the lower metric and we set the new
            %survivor state, otherwise we dont get into this if
            %we keep the same survivor state
            if((state_metric(nextstate(j+1,l+1)+1,2) >
state_metric(j+1,1)+branch_metric) || flag(nextstate(j+1,l+1)+1)==0)
                state_metric(nextstate(j+1,l+1)+1,2) =
state_metric(j+1,1)+branch_metric;
                survivor_state(nextstate(j+1,l+1)+1,i+1)=j;
                flag(nextstate(j+1,l+1)+1)=1;
            end

        end
    end
    %we switch the order of the columns in our state_metric matrix;
    %actually what we would be switching is the second row which
    %before that change was the first one
    state_metric=state_metric(:,2:-1:1);
end

%start decoding of the tail channel-outputs
%we do the same procedure for the last time instants of the Trellis
for i=depth_of_trellis-L+2:depth_of_trellis
    flag=zeros(1,number_of_states);
    last_stop=number_of_states/(2^((i-depth_of_trellis+L-2)*k));
    for j=0:last_stop-1
        branch_metric=0;
        binary_output=deci2bin(output(j+1,1),n);
        binary_output_antipodal = 1-2*binary_output;
        for ll=1:n
            branch_metric=branch_metric+abs(decoder_in_matrix(ll,i)-
abs(h)*binary_output_antipodal(ll));
        end
        if((state_metric(nextstate(j+1,1)+1,2) >
state_metric(j+1,1)+branch_metric) || flag(nextstate(j+1,1)+1)==0)
            state_metric(nextstate(j+1,1)+1,2) =
state_metric(j+1,1)+branch_metric;
            survivor_state(nextstate(j+1,1)+1,i+1)=j;
```

```
            flag(nextstate(j+1,1)+1)=1;
        end
    end
    state_metric=state_metric(:,2:-1:1);
end

%generate the decoder output from the optimal path; we work backwards
%through the state history table (survivor_state)(shows the surviving
%predecessor states for each state at each time t), for the selected
%state and for that particular time instant, we select a new state
%which is listed in the state history table as being the best
%predecesor to thatstate (we save the state number of each
%selected state)

%array to store a list of states determined during traceback;
%its a line array which dimensions are 1xdepth_Trellis+1
state_sequence=zeros(1,depth_of_trellis+1);

%the last element of this array must be the state 0, so the
%penultimun will be the state which path leads to that 0 state
%(in our example that path comes from the state 1)
state_sequence(1,depth_of_trellis)=survivor_state(1,depth_of_trellis+1
);

%we obtein the states selected when tracing the path back through
%the survivor state table
for i=1:depth_of_trellis
    state_sequence(1,depth_of_trellis-
i+1)=survivor_state((state_sequence(1,depth_of_trellis+2-
i)+1),depth_of_trellis-i+2);
end

%once we've obteined the path back, using a table that maps state
%transitions to the imputs that caused them, we can recreate the
%original message; the flushig bits at the end are discarded so
%the dimensions of this line vetor are 1xdepth_Trellis-L+1
%(=1xdepth_Trellis-M)
decoder_out_matrix=zeros(k,depth_of_trellis-L+1);
for i=1:depth_of_trellis-L+1

dec_output_deci=input(state_sequence(1,i)+1,state_sequence(1,i+1)+1);
    dec_output_bin=deci2bin(dec_output_deci,k);

    %if k=1 this matrix will be equal to de final result, if not,
    %we'll be having the estimated bits of the message in each time
    %instant per columns, afterwards, we will have to organise them
    %in a single row
    decoder_out_matrix(:,i)=dec_output_bin(k:-1:1)';
end

%we give the result in a row vector
decoder_out=reshape(decoder_out_matrix,1,k*(depth_of_trellis-L+1));
```

graph_some_collab.m

```
function graph_some_collab
%we are going to be plotting in this graph the different scenarios

%we will be sending a BPSK signal(basically 1's and -1's) that
%goes through an additive gaussian noise channel (AWGN) with
%slow Rayleight fading; in this case we are going to see the
%differences when we always have collaboration from the relay
%and when we have some collaboration from the relay

%we will design the ideal channel with which we'll make the
%comparaisons values of EbN0 in dBs
EbN0_values = 0:25;
ebn0 = 10.^(EbN0_values/10);

%we define the different variables for the collaborative part
%in here (the different distances between all the devices in
%the system)
d_sd = 1;
d_sr = 0.5;
d_rd = 0.5;

%we calculate the path gain for the the source-relay path and
%for the relay-destination path (we have set the path gain for
%the source-destination path to 1)
%the formula to calculate the path gain for the source-relay
%path for example would be: pg_sr=(d_sd/d_rd)^2*pg_sd but the
%pg_sd=1 so we dont need to take it into consideration
pg_sr = (d_sd/d_sr)^2;
pg_rd = (d_sd/d_rd)^2;

tau = 9.12;
P_no_collab = 1 - exp(-tau./(pg_sr*ebn0));
P_collab = exp(-tau./(pg_sr*ebn0));


%we plot the FER in an AWGN channel with a BPSK modulation and a
%convolutional encoder and a viterbi decoder without collaboration
[EbN0_values,BER,FER_no_collab] =
scenario_collaboration('no_collaboration');
BER
semilogy(EbN0_values,FER_no_collab,'rx-','linewidth', 2)
hold on;


%we plot the BER in an AWGN channel with a BPSK modulation and a
%convolutional encoder and a viterbi decoder when we have some
%collaboration
[EbN0_values,BER,FER_collab] =
scenario_collaboration('perfect_collaboration');
BER
FER_some_collab = FER_no_collab*P_no_collab + FER_collab*P_collab;
TSNR = ebn0*(P_no_collab+ (3/2)*P_collab);
TSNR_db = 10*log10(TSNR);
semilogy(TSNR_db,FER_some_collab,'bx-','linewidth', 2)
hold on;
```

```
grid on
axis([ 0 25 1e-4 1])
title('BPSK modulation in an AWGN channel with Rayleigh fading without
and with some collaboration')
legend('convolutional-code-no-collaboration', 'convolutional-code-
some-collaboration');
xlabel('TSNR (dB)')
ylabel('FER')
```