POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Convolutional codes for short-packet communication over non-coherent channels

Tesi di Laurea Magistrale in Telecommunication Engineering - Ingegneria delle telecomunicazioni

Author: **Matteo Ferro**

Student ID: 217765
Advisor: Prof. Maurizio Magarini
Co-advisors: Gianluigi Liva, Riccardo Schiavone
Academic Year: 2023-24

Abstract

The design of error-correcting codes for short information blocks is becoming an increasingly relevant research area, driven by the demand for efficient transmission in emerging wireless applications. These include machine-type communications, smart metering networks, the Internet of Things, remote command links, and messaging services, all of which require a reliable and efficient transmission of small data units. For short information blocks in coherent communication systems, it has been proved that convolutional codes are able to approach performance bounds. However, synchronization using known symbols (pilot symbols) adds significant overhead when dealing with short packets. In this Master's thesis we try to overcome this limit by analyzing convolutional codes over noncoherent channels, focusing on the use of noncoherent receivers that eliminate the need for pilot symbols. We examine this approach with both BPSK and QPSK modulation schemes, evaluating the performance of zero-tail and tail-biting convolutional codes with different memory values. Finally, we assess the complexity and performance of the proposed solutions, comparing them to known approximations of the finite-length bounds for the best possible codes in coherent channels.

Keywords: convolutional codes, noncoherent channel, short-packet transmission, Viterbi algorithm, Viterbi tracking

Abstract in lingua italiana

La progettazione di codici di correzione d'errore per blocchi di informazione corti sta diventando un'area di ricerca sempre più rilevante, guidata dalla necessità di una trasmissione efficiente in numerose nuove applicazioni wireless. Queste includono comunicazioni tra macchine, reti di misurazione intelligente, l'Internet of Things, collegamenti di comando remoto e servizi di messaggistica, che richiedono una trasmissione affidabile di piccole unità di dati. Per blocchi di informazione brevi in sistemi di comunicazione coerenti, è stato dimostrato che i codici convoluzionali sono in grado di avvicinarsi ai bound teorici. Tuttavia, la sincronizzazione tramite simboli noti (piloti) introduce un notevole overhead quando si trasmettono pacchetti corti. In questa tesi magistrale proponiamo delle possibili soluzioni per superare questo limite analizzando i codici convoluzionali su canali non coerenti, concentrandoci sull'uso di ricevitori che eliminano la necessità di trasmettere piloti. Valutiamo questo approccio usando una modulazione BPSK e QPSK, confrotando le prestazioni dei codici convoluzionali zero-tail e tailbiting a diversi valori di memoria. Infine, analizziamo la complessità e le prestazioni delle soluzioni proposte, confrontandole con i bound noti sui pacchetti corti per i migliori codici possibili in canali coerenti.

Parole chiave: codici convoluzionali, canale non coerente, pacchetti corti, algoritmo di Viterbi, Viterbi tracking

Contents

Abstract **i**

Abstract in lingua italiana **iii**

Contents **v**

Introduction **1**

1 Convolutional codes **3**
1.1 Encoder . . . 3
1.1.1 Free distance . . . 4
1.1.2 Finite state machine representation of the code . . . 5
1.1.3 Distance spectrum . . . 7
1.1.4 Code termination . . . 8
1.1.5 Trellis diagram . . . 8
1.2 Decoder . . . 9
1.2.1 Viterbi algorithm . . . 10
1.2.2 Wrap-around Viterbi algorithm . . . 11

2 Convolutional codes: decoding algorithms for noncoherent channels **13**
2.1 Channel definition . . . 13
2.2 ML decoder with perfect CSI . . . 14
2.3 Pilot-assisted transmission . . . 15
2.4 ML decoder without CSI . . . 16
2.5 Blind Viterbi decoding . . . 17
2.6 Performance analysis . . . 18

3 Viterbi tracking **21**
3.1 Phase estimation error . . . 21
3.2 Viterbi tracking algorithm and extra loop . . . 24

3.3 CRC codes 26
3.4 Threshold on cumulative metric 29
3.5 Tailbiting terminated codes 31
3.5.1 Suboptimal TBCCs 32
3.5.2 Additional bit check 35
3.6 QPSK modulation 36
3.7 Memory 8 codes 39

4 Conclusions **43**

Bibliography **45**

List of Figures **47**

List of Tables **49**

Acronyms **51**

Introduction

Countless emerging applications require devices to sporadically transmit short data units. This is the case for example of machine-type communications, smart metering networks, the Internet of Things, remote command links, and messaging services. In this framework, limited complexity and low power consumption are key parameters to guarantee an effective operation. This means that simplicity is a fundamental feature, as well as robustness, in ensuring an efficient and reliable communication, and, to do so, a powerful channel coding technique has to be implemented.

In 1948 Claude Shannon showed that, when transmitting through a noisy channel, it is possible to achieve an infinitely small error probability by using a code with rate R smaller than the capacity C of the channel. Though we know this is possible by using a code that is sufficiently long, his work did not tell how such code has to be constructed. Since then, theory has made significant progress, and several well-established methods now exist for approaching channel capacity with reasonable complexity.

However, as Shannon's theorem demonstrates, large block lengths are necessary to approach capacity. In our scenario though, we are constrained to transmit only a small amount of information bits at a time (e.g. 64 information bits per packet).
In this setting a whole range of new possibilities arises: amongst them, the purpose of this thesis will be to understand the performance of convolutional codes in this framework, proving that, given their simplicity, they could be extremely powerful under said constraints.

For short information blocks, convolutional codes have been shown to approach the performance bounds of the best possible codes in coherent communications. However, synchronization using known symbols (pilot symbols) introduces significant overhead when transmitting short packets. To address this issue, we analyzed the performance of convolutional codes over noncoherent channels, exploring the use of noncoherent receivers that eliminate the need for pilot symbols. By exploring a few different approaches, we were

able to consistently obtain a significant coding gain, up to 0.4 dB, with respect to the pilot-aided decoder, all of this with a manageable increase in complexity.

In Chapter 1 we give an overview on convolutional codes, highlighting their main features and properties and introducing some fundamental definitions. We also detail the working principle of the Viterbi algorithm and its modified version for tailbiting codes, i.e. the wrap-around Viterbi algorithm.

In Chapter 2 we first give a description of the noncoherent channel and its properties, then, we introduce the classical pilot-aided decoder and the blind one that does not require any channel knowledge.

In Chapter 3 we propose a Viterbi tracking algorithm to enhance the performance of the noncoherent decoder, examining two different approaches. We then show and comment the performance of the proposed implementations.

1 | Convolutional codes

1.1. Encoder

As stated in the Introduction, currently known theory has proved that there exist several well-developed ways to encode and decode packets with arbitrarily low error probability whilst approaching capacity. When there is no limitation on the block length, other solutions are generally preferred to convolutional codes as the latter do not typically ensure optimal performance.
However, if we are constrained to transmit very short packets (e.g. 64 information bits), recent developments showed that convolutional codes [2] may actually prove as a very simple and consistent way to approach this problem [1, 4, 5, 9, 12, 14].

A **binary convolutional code (CC)** can be easily described through its encoder, which corresponds to the following equation:

$$\mathbf{x}_t = \mathbf{u}_t\mathbf{G}_0 + \mathbf{u}_{t-1}\mathbf{G}_1 + ... + \mathbf{u}_{t-\nu}\mathbf{G}_\nu = \sum_{i=0}^{\nu} \mathbf{u}_{t-i}\mathbf{G}_i. \tag{1.1}$$

At each time instant the encoder receives a vector $\mathbf{u}_t$ of k bits and outputs a vector $\mathbf{x}_t$ of n bits, such that it is said to have *rate* $R = \frac{k}{n}$.
$\mathbf{G}_i$, $i = 0, ..., \nu$ are $k \times n$ binary matrices. Since usually $k = 1$, they reduce to a single row. Throughout all the discussion we will focus on $(n = 2,\ k = 1,\ \nu)$ codes, thus $R = \frac{1}{2}$ codes.

The *memory* of the CC is specified by the parameter ν, which also describes the error correction capabilities of the code itself: a longer memory allows for a greater error correction power, which is paid by means of increased complexity.
This could be very easily justified by acknowledging that the farther the words are from one another, the lower is the probability of wrongly decoding the received codeword. As we will highlight in the next section, a greater memory allows the encoder to generate words that have a higher number of different bits one from the other.

In the following, we will have a look at some properties of CCs that will be paramount to understand how to handle them and thus optimize their performance.

1.1.1. Free distance

As previously stated, the distance between two different words strongly influences the overall error probability of the system. For a better understanding, we will now give some fundamental definitions:

Definition 1.1.1. *Given a vector* $\boldsymbol{x}$ *its* ***Hamming weight*** $w_H(\boldsymbol{x})$ *is defined as its number of ones, i.e.*

$$w_H(\boldsymbol{x}) = \{i : x_i = 1\}.$$

Definition 1.1.2. *Given two vectors* $\boldsymbol{x}$ *and* $\boldsymbol{x'}$, x_i, $x'_i \in \{0, 1\}$, *the* ***Hamming distance*** *is defined as the number of positions in which* $\boldsymbol{x}$ *and* $\boldsymbol{x'}$ *differ, i.e.*

$$d_H(\boldsymbol{x}, \boldsymbol{x'}) = \{i : x_i \neq x'_i\}.$$

Definition 1.1.3. *The* ***free distance*** *of a code is defined as the minimum Hamming distance between two distinct codewords generated by its encoder:*

$$d_{free} = \min_{\substack{\boldsymbol{c},\boldsymbol{c'} \in \mathcal{C} \\ \boldsymbol{c} \neq \boldsymbol{c'}}} d_H(\boldsymbol{c}, \boldsymbol{c'}).$$

This is to be intended as the minimum number of different bits between two codewords in the codebook $\mathcal{C}$, which is the collection of all possible codewords that can be generated by a given encoder.
Since CCs are linear, the all zero codeword always belongs to the codebook. Moreover, it is possible to show that, thanks to the linearity, the spectrum of distances from a reference codeword $\mathbf{c}$ is invariant with respect to the choice of $\mathbf{c}$. Thanks to that, it is possible to compute the free distance by looking at the non-zero word in the codebook with the minimum amount of bits equal to one.

The free distance of a code, once we take care to select the best polynomial available, is dependent on its memory, such that, normally, the higher the memory, the higher the free distance (Table 1.1). However, higher memory also equals increased complexity and this trade-off will accompany us throughout all the discussion.

1.1.2. Finite state machine representation of the code

The convolutional encoder is in fact a finite state machine (FSM): at a certain time instant t, it is possible to derive the output $\mathbf{x}_t$ given the input u_t and the state of the registers.

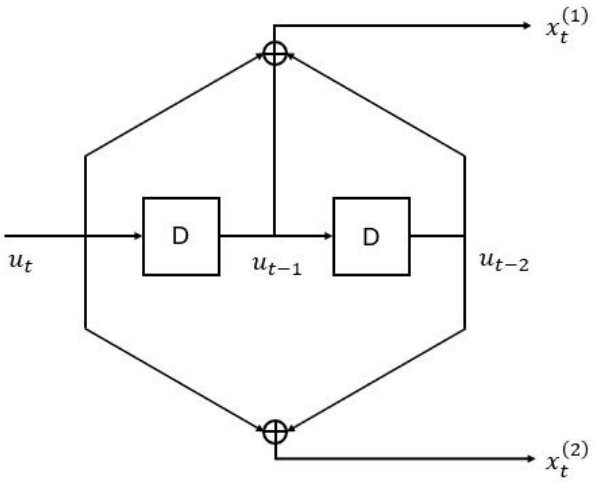

Figure 1.1: Encoder of [7, 5] CC.

The output is directly dependent on the input values at previous time instants, i.e. u_{t-1} and u_{t-2}, which also correspond to the register content, and hence the state, at time t. The number of states is given by the the memory of the code as 2^{ν}. In the example shown in Figure 1.1, it is very simple to derive the relation between the input bit and the output ones:

- $x_t^{(1)} = u_t + u_{t-1} + u_{t-2}$
- $x_t^{(2)} = u_t + u_{t-2}$

$x_t^{(1)}$ being the first output bit and $x_t^{(2)}$ the second one.
Such equations can be rewritten using the D-transform, which, given a sequence $\mathbf{x} = (\dots, x_{-2}, x_{-1}, x_0, x_1, x_2, \dots)$, is defined as:

$$x(\mathrm{D}) = \sum_t x_t \mathrm{D}^t. \tag{1.2}$$

By developing the D-transform of the input-output relation we obtain the following:

- $x^{(1)}(\mathrm{D}) = u(\mathrm{D})(1 + \mathrm{D} + \mathrm{D}^2)$
- $x^{(2)}(\mathrm{D}) = u(\mathrm{D})(1 + \mathrm{D}^2)$

in which D^k represents a delay of k samples. It is possible to write this compactly with a polynomial matrix representation:

$$\left[x^{(1)}(\mathrm{D}), x^{(2)}(\mathrm{D})\right] = u(\mathrm{D})\left[G^{(1)}(\mathrm{D}),\ G^{(2)}(\mathrm{D})\right] \tag{1.3}$$

where $G(\mathrm{D}) = \left[G^{(1)}(\mathrm{D}),\ G^{(2)}(\mathrm{D})\right] = [1 + \mathrm{D} + \mathrm{D}^2,\ 1 + \mathrm{D}^2]$ is the polynomial generator matrix of the CC.

The polynomial is typically given in octal form, such that the encoder of Figure 1.1 corresponds to the [7, 5] CC.
Throughout the discussion, we will make use of the codes listed in Table 1.1: those are the best in terms of minimum distance amongst all possible ones at fixed values of memory ν.

	$\nu = 4$	$\nu = 5$	$\nu = 6$	$\nu = 7$	$\nu = 8$
G(D)	[27, 31]	[53, 75]	[133, 171]	[247, 371]	[561, 753]
$\mathbf{d}_{\min}$	7	8	10	10	12

Table 1.1: Optimum CCs for different memory values ν.

Given the polynomial generator matrix, we have the input-output correspondence of the code, which we can summarize with a state transition table, as shown in Table 1.2:

Current State (u_{t-1}, u_{t-2})	**Input** u_t	**Output** $\left(x_t^{(1)}, x_t^{(2)}\right)$	**Next State** (u_t, u_{t-1})
$S_0 = (0, 0)$	0	(0, 0)	$S_0 = (0, 0)$
$S_0 = (0, 0)$	1	(1, 1)	$S_2 = (1, 0)$
$S_1 = (0, 1)$	0	(1, 1)	$S_0 = (0, 0)$
$S_1 = (0, 1)$	1	(0, 0)	$S_2 = (1, 0)$
$S_2 = (1, 0)$	0	(1, 0)	$S_1 = (0, 1)$
$S_2 = (1, 0)$	1	(0, 1)	$S_3 = (1, 1)$
$S_3 = (1, 1)$	0	(0, 1)	$S_1 = (0, 1)$
$S_3 = (1, 1)$	1	(1, 0)	$S_3 = (1, 1)$

Table 1.2: State transition table of [7, 5] code.

The couples (u_t, u_{t-1}) and (u_{t-1}, u_{t-2}) indicate respectively the next state and the current state, given the input bit u_t. This is equivalent to the FSM diagram representation in Figure 1.2:

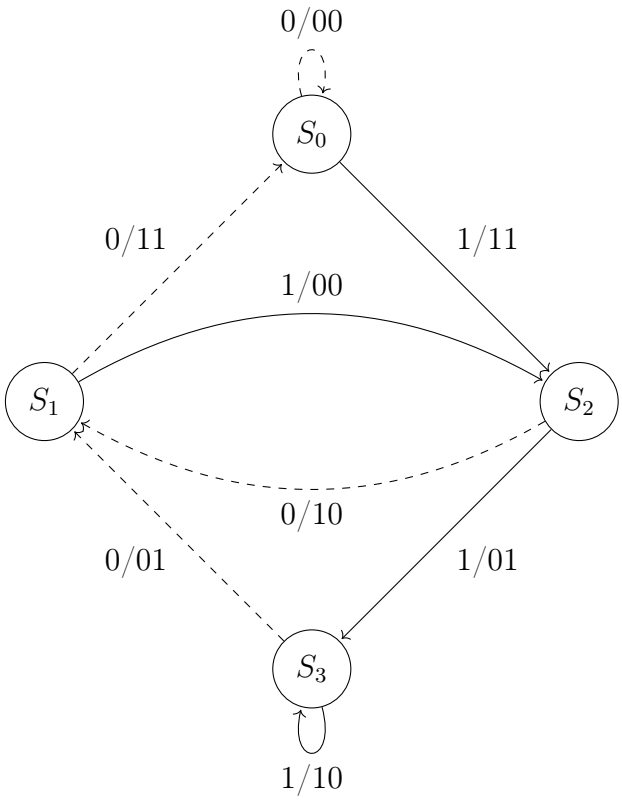

Figure 1.2: FSM diagram representation of [7, 5] code.

1.1.3. Distance spectrum

To have a better understanding of the codebook and its "quality", it is possible to give a description of it through its *distance spectrum*. This basically represents, for each possible Hamming weight, spanning from zero to the codeword length, the multiplicity of the codewords with that Hamming weight.

Definition 1.1.4. *Given an (n, k) binary code $\mathcal{C}$, its* ***weight enumerator*** *or* ***distance spectrum*** *is the number of codewords with Hamming weight i, i.e.*

$$A_i = \{\boldsymbol{c} \in \mathcal{C} : w_H(\boldsymbol{c}) = i\}.$$

Together with the free distance, the distance spectrum gives a more detailed description of the codebook, that could help us understand the properties and behavior of the CC. All this information will be fundamental in the choices operated afterwards in later stages of the discussion.

1.1.4. Code termination

When we make use of a convolutional encoder, it could be important that, at some point, once we run out of information bits to transmit, we terminate the ongoing encoding process. Among different options, for this discussion we will focus on these two alternatives:

- **Zero-tail** termination.

  We assume S_0, the all zero state, to be the initial one and, after K clock times, we constrain the code to reset back to S_0. This takes ν steps, corresponding to $n \cdot \nu$ output bits, which means the rate decreases to $R = \frac{K}{(K+\nu)\cdot n}$.

- **Tailbiting** termination.

  We do not make any assumption on the starting state but we constrain the ending state to be the same as the initial one. This means we do not need to add further bits at the end to terminate the code and the rate remains the same of the non-terminated encoder, thus $R = \frac{1}{2}$.

1.1.5. Trellis diagram

When the FSM diagram is unwrapped in time, highlighting all possible paths that generate the words in the codebook, we get a new representation of the code called *trellis diagram* [3].

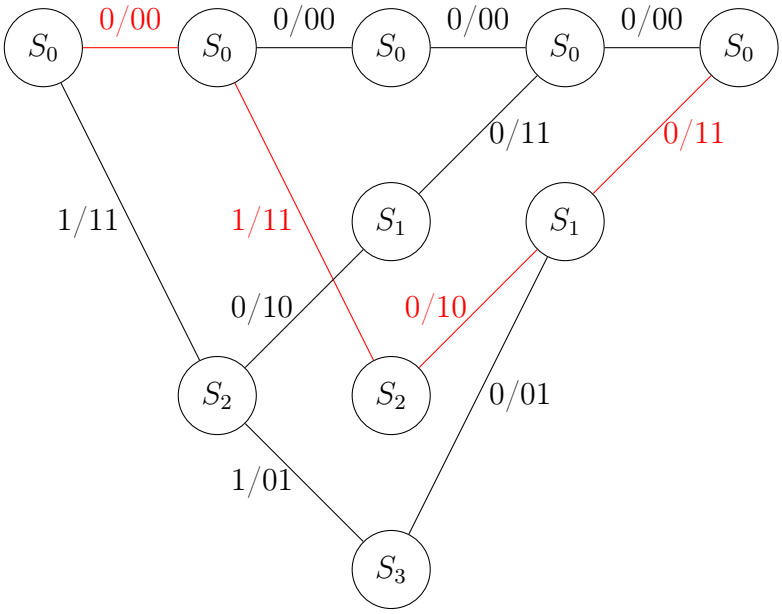

Figure 1.3: Trellis diagram of the [7, 5] zero-tail terminated convolutional code (ZTCC), $K = 2$ information bits.

In a ZTCC the diagram usually starts from the state S_0 and at each clock time a new section is appended, which includes all new possible states and edges. By following an existing path from start to end it is possible to derive one of the codewords belonging to the codebook.
For example, in Figure 1.3 the red lines highlight one amongst all possible paths that generate a valid codeword. By selecting 01 as input, the corresponding codeword generated by the encoder is 00111011. For tailbiting terminated convolutional codes (TBCCs) instead, it is possible to build a number of subtrellises equal to the total number of states (i.e. 2^{ν}), where each subtrellis starts and ends in the same state.

1.2. Decoder

We consider next the transmission over an additive white Gaussian noise (AWGN) channel, with

$$y_i = x_i(c_i) + n_i \tag{1.4}$$

where $x_i(c_i) \in \{-1,\ +1\}$ is the binary phase-shift keying (BPSK) mapping of the codeword bit c_i and $n_i \sim \mathcal{N}(0,\ \sigma^2)$ is the noise component with variance σ^2. For the sake of simplicity, from now on we will simply use the notation $\mathbf{x}$ to indicate the transmitted sequence.
Whenever a codeword is received, the maximum likelihood (ML) decoding criterion is made such that it decides in favour of the word that maximizes the likelihood function $p(\mathbf{y}|\mathbf{x})$:

$$\hat{\mathbf{x}}_{ML} = \arg\max_{\mathbf{x}\in\mathcal{C}} p(\mathbf{y}|\mathbf{x}) \tag{1.5}$$

where with a slight abuse of notation we denoted the set of modulated codewords as $\mathcal{C}$.
By developing the log-likelihood function over the AWGN channel and getting rid of terms that are not of interest for the maximization problem we end up with:

$$\begin{aligned}
\hat{\mathbf{x}}_{ML} &= \arg\max_{\mathbf{x}\in\mathcal{C}} p(\mathbf{y}|\mathbf{x}) \\
&= \arg\max_{\mathbf{x}\in\mathcal{C}} \ln \prod_{i=1}^{N} p(y_i|x_i) \\
&= \arg\max_{\mathbf{x}\in\mathcal{C}} \sum_{i=1}^{N} \ln \left\{ \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left(-\frac{(y_i - x_i)^2}{2\sigma^2} \right) \right\} \\
&= \arg\min_{\mathbf{x}\in\mathcal{C}} d_E^2(\mathbf{y}, \mathbf{x})
\end{aligned} \tag{1.6}$$

where N indicates the length of the codeword $\mathbf{c}$ and $d_E^2(\mathbf{y}, \mathbf{x})$ is the squared Euclidean distance between $\mathbf{y}$ and $\mathbf{x}$.

$$d_E^2(\mathbf{y}, \mathbf{x}) = ||\mathbf{y} - \mathbf{x}||_2^2 = \sum_{i=1}^{N} (y_i - x_i)^2. \tag{1.7}$$

1.2.1. Viterbi algorithm

Ideally, the search has to be performed over all possible codewords belonging to the codebook. Clearly, for very large codebooks, it is impossible to compute the likelihood for every possible word and select the best one, as it would take a huge amount of time. An efficient way to do so without resorting to any approximation is through the **Viterbi algorithm** [11], which exploits the property that the squared Euclidean distance can be computed as in Eq. (1.7).

The algorithm makes use of two different metrics:

- We denote as Λ_t^i the **cumulative metric** for state S_i at time t; the cumulative metric is the minimum squared Euclidean distance amongst all possible paths reaching state S_i at time t.
- We define then $\lambda_t^{i,j}$ the **branch metric** at time t for the state transition between state S_i and S_j.

The algorithm scans the trellis diagram starting from the first time instant and every time two paths join in the same state it selects the one with the best metric and ignores the others, according to the following rules:

1. At the first time step, the cumulative metric is initialized such that: $\Lambda_0^j = 0 \ \forall j$.

2. For all valid transitions, it computes the *branch metric*

   $$\lambda_t^{i,j} = d_E^2(\mathbf{y}_t, \mathbf{x}_t^{i,j}).$$

   Where $\mathbf{x}_t^{i,j}$ is the output of the transition from state S_i to state S_j and $\mathbf{y}_t$ is the received signal, both at time instant t.

3. For all available states at time t, it calculates the best *cumulative metric* as follows:

   $$\Lambda_t^j = \min_i (\Lambda_{t-1}^i + \lambda_t^{i,j}).$$

4. For each state, keep in memory only the path with the minimum squared Euclidean distance, which is called *survivor* path, such that there are in memory as many survivors as there are possible states.

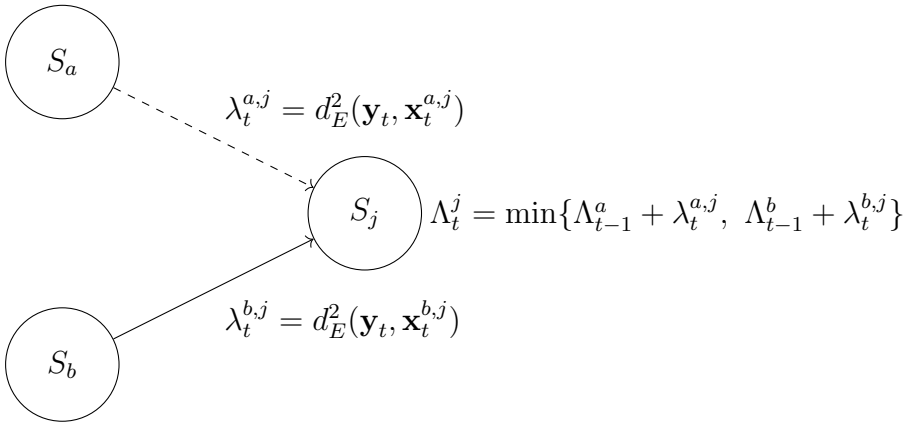

Figure 1.4: Viterbi update rule over the code trellis.

Thanks to the property of the squared Euclidean distance we can be sure that the paths cancelled are in fact worse than the ones we keep without any doubt, as they can only lead to overall higher values of the cumulative metric. If the code is terminated, at the last time instant there will be just a single path remaining and this is the one associated to the ML codeword.

1.2.2. Wrap-around Viterbi algorithm

If the code is instead tailbiting, a proper ML decoder should run the Viterbi algorithm (VA) on each possible subtrellis and then select the best word among the resulting ones. However, this is highly inefficient, especially if the code has large memory and thus a large number of subtrellises.
A valid alternative would be using a slightly modified version of the algorithm, such as the **wrap-around Viterbi algorithm (WAVA)**, that works as follows: at the start, every state is initialized with the same cumulative metric equal to zero. After running the algorithm through all time steps, at the last time instant there will still be as many paths left as there are states. Therefore what we need to do is, while scanning the trellis, to step by step keep track of the starting state of the survivor paths: once we reach the end, it could be that not all the paths are actually tailbiting, which means that initial and ending state of a path could differ.

If the path that presents the overall best metric is not tailbiting, we restart from the beginning, setting the initial cumulative metric at each state equal to the value of the cumulative metric at the same state, computed at the end of the first iteration. We continue until we either get a tailbiting word or we reach a maximum number of iterations that was previously set.

Algorithm 1.1 Wrap-around Viterbi algorithm.

$I = 0$
$\Lambda_i^{(0)} = 0, \quad \forall i = 0, ..., 2^{m-1}$
stop = false
nack = false
$[\hat{\mathbf{x}}_{ML}, S^{(0)}, S^{(K)}, \Lambda_i^{(K)}] = \text{Viterbi}(\mathbf{y}, \Lambda_i^{(0)})$
$I = I + 1$
if $S^{(K)} = S^{(0)}$ **then**
  return $\hat{\mathbf{x}}_{ML}$;
else
  $\Lambda_i^{(0)} = \Lambda_i^{(K)}$
end if
while not stop **do**
  $[\hat{\mathbf{x}}_{ML}, S^{(0)}, S^{(K)}, \Lambda_i^{(K)}] = \text{Viterbi\_TB}(\mathbf{y}, \Lambda_i^{(0)})$
  $I = I + 1$
  if $S^{(K)} = S^{(0)}$ **then**
    return $\hat{\mathbf{x}}_{ML}$;
  else
    $\Lambda_i^{(0)} = \Lambda_i^{(K)}$
    if $I > I_{\max}$ **then**
      stop = true;
      nack = true;
      return nack
    end if
  end if
end while

Where I is the number of iterations, and $I_{\max}$ is the fixed maximum iterations number, $S^{(0)}$ is the starting state and $S^{(K)}$ is the ending state, $\Lambda_i^{(t)}$ is the cumulative metric of state i at time t, $\mathbf{y}$ is the received signal, $\hat{\mathbf{x}}_{ML}$ is the word presenting the ML value.
At the first iteration the algorithm returns a word only if the ML one is also tailbiting. From the second iteration on, it returns the best tailbiting word, if there is at least one. If after $I_{\max}$ iterations no tailbiting word is found, the algorithm returns a not acknowledged flag.

2 | Convolutional codes: decoding algorithms for noncoherent channels

In satellite communications, especially when transmitting short messages, noncoherent channels present unique challenges and opportunities. Unlike coherent channels, where precise phase tracking is essential, noncoherent systems operate without requiring exact phase alignment, which can simplify the receiver design. This becomes particularly important in scenarios where the Doppler shift remains relatively constant over the transmission of a packet, which is the case with short satellite communications.
The phase rotation caused by the Doppler component can be considered stable, allowing noncoherent techniques to efficiently mitigate its effects. Understanding how to handle these noncoherent channels is crucial to ensure a reliable communication in frameworks where simplicity and robustness are key, such as in low-complexity or power-constrained satellite links.
In this chapter we analyze different decoding solutions for convolutional codes over the noncoherent communication scenario.

2.1. Channel definition

To better characterize the performance of CCs in this setting, we considered a noncoherent AWGN channel model, where the transmitted word is rotated by a (random) phase that is constant over the whole code block. This model captures, for instance, bursty transmissions of short packets in satellite links.
This means that the received signal has the same power of the original one but is rotated of a phase ϕ, which is set to be uniformly distributed between $-\pi$ and π. The channel model yields:

$$y_i = hx_i + n_i \tag{2.1}$$

where $h = e^{j\phi}$ and $\phi \sim U[-\pi, \pi)$. Since we considered a BPSK modulation, $x_i(c_i) \in \{-1, 1\}$. Furthermore, the noise component is a circular symmetric complex Gaussian random variable with variance $2\sigma^2$, i.e. $n_i \sim \mathcal{CN}(0, 2\sigma^2)$.

The signal-to-noise ratio (SNR) is defined as $\frac{E_s}{N_0} = \frac{1}{2\sigma^2}$ where E_s is the energy of a modulated symbol and N_0 is the single-sided noise power spectral density. Alternatively, the SNR can be expressed in $\frac{E_b}{N_0}$, where $E_b = \frac{E_s}{R}$ is the energy per information bit.
As the packet length is constrained to be significantly short (e.g. 64 information bits), it is reasonable to assume that the channel does not change importantly during the transmission of a single packet, as the coherence time is much larger than the transmission time.

2.2. ML decoder with perfect CSI

As stated in Section 1.2, the general approach is to use the ML criterion to decide the codeword, which works by maximizing the likelihood function.
In our scenario, supposing we have knowledge of the channel h, we obtain:

$$\begin{aligned}
\hat{\mathbf{x}}_{ML} &= \arg\max_{x \in \mathcal{C}} p(\mathbf{y}|\mathbf{x}, h) \\
&= \arg\max_{\mathbf{x} \in \mathcal{C}} \prod_{i=1}^{N} p(y_i|x_i, h) \\
&= \arg\max_{x \in \mathcal{C}} \prod_{i=1}^{N} \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2\sigma^2}|y_i - hx_i|^2} \\
&= \arg\max_{x \in \mathcal{C}} e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{N}|y_i - hx_i|^2} \\
&= \arg\min_{x \in \mathcal{C}} \sum_{i=1}^{N} |y_i - hx_i|^2 \\
&= \arg\min_{x \in \mathcal{C}} d_E^2(\mathbf{y}, h\mathbf{x}).
\end{aligned} \tag{2.2}$$

As pointed out in the previous chapter, we end up with a minimization of a squared Euclidean distance, where the received signal is now rotated by a phase ϕ with respect to the transmitted one.

2.3. Pilot-assisted transmission

If we do not have any information about the channel, the phase rotation alters completely the received signal and we are not able to perform the decoding using a classical decoder. In order to successfully deal with this, a typical approach is to send, along with the information bits, a certain amount of symbols that are known both at the receiver and the transmitter, which are called *pilot symbols*.
Through them, the decoder is able to compute an estimate of the channel and then compensate for the rotation when running the decoding algorithm. This clearly has a cost in terms of rate, especially when we are dealing with short block-lengths, and there is a trade-off between the number of pilot symbols and the accuracy of the phase rotation estimate. A longer pilot sequence would allow for a better channel estimate, but reduces significantly the rate.
In particular, when a length-L pilot sequence is appended, the rate becomes

$$R = \frac{K}{(K+\nu)\cdot n + L} \tag{2.3}$$

where K is the number of information bits, ν is the memory of the code (we suppose in this case to use ZTCCs), n is the number of encoded bits per input bit (note that we always suppose to use a rate-$\frac{1}{2}$ encoder).

To sum up, if we decide to use $L = 14$ pilot symbols while transmitting 64 information bits using a memory 6 code, the overall rate will be $R = \frac{64}{(64+6)\cdot 2+14} \simeq 0.416$. Note that, without the pilot field, the code rate would be $R = \frac{64}{(64+6)\cdot 2} \simeq 0.457$. For a fixed $\frac{E_s}{N_0}$, the change in rate results in a $\frac{E_b}{N_0}$ loss of $10\log_{10}\frac{(64+6)\cdot 2+14}{(64+6)\cdot 2} \simeq 0.41$dB.

Denoting by $\mathbf{x}_p = (x_p^1, x_p^2, \ldots, x_p^L)$ the pilot sequence and $\mathbf{y}_p = (y_p^1, y_p^2, \ldots, y_p^L)$ its observation at the channel output, we compute the ML channel estimate as

$$\hat{h} = \frac{\mathbf{x}_p^H \mathbf{y}_p}{||\mathbf{x}_p||^2} = \frac{\sum_{i=1}^{L} y_p^i}{L} \tag{2.4}$$

as in our system we fixed $x_p^i = 1$, $i = 1, \ldots, L$.

Once we have our channel estimate, we can run the Viterbi algorithm with the following metric

$$\lambda_t^{i,j} = d_E^2(\hat{h}^* \mathbf{y}_t, \mathbf{x}_t^{i,j}). \tag{2.5}$$

2.4. ML decoder without CSI

As we pointed out, adding pilot symbols to accurately estimate the channel is paid through a decrease in rate, which results in an energy loss.
This in practice creates a gap between the opitmum performance, where we suppose to have knowledge of the channel in absence of pilots, and the actual one, where we need to introduce an overhead to obtain that knowledge.
With this in mind, we looked for a solution that allows limiting (or even removing) the overhead introduced by pilots, such that we could possibly close the previously mentioned gap.

To do so, it is necessary to understand the behaviour of the likelihood function of the channel whenever the channel state information is not available.
In the following, we will give a complete and thorough mathematical description of how the ML metric was derived, starting again from the ML decision rule:

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}\in\mathcal{C}} p(\mathbf{y}|\mathbf{x}).$$

As the phase is unknown, we can rewrite the likelihood function adding the dependence on ϕ:

$$p(\mathbf{y}|\mathbf{x}) = \int_{-\pi}^{\pi} p(\mathbf{y}|\mathbf{x},\phi)\cdot p(\phi)\mathrm{d}\phi \quad \text{with} \quad p(\phi) = \frac{1}{2\pi}.$$

This expression can be further developed by considering the bit by bit likelihood function:

$$p(\mathbf{y}|\mathbf{x},\phi) = \prod_{i=1}^{N} p(y_i|x_i,\phi) = \prod_{i=1}^{N} \frac{1}{2\pi\sigma^2}\cdot e^{-\frac{1}{2\sigma^2}|y_i - e^{j\phi}x_i|^2} = \prod_{i=1}^{N} C_0 C_1(y_i) C_2 \cdot e^{\frac{x_i\,\mathrm{Re}\{y_i e^{-j\phi}\}}{\sigma^2}}$$

such that the constants C_0, C_1 and C_2 are defined as follows:

$$C_0 = \tfrac{1}{2\pi\sigma^2}; \quad C_1 = e^{-\frac{|y_i|^2}{2\sigma^2}}; \quad C_2 = e^{-\frac{1}{2\sigma^2}}.$$

The numerator of C_2 is equal to 1 as $|x_i|^2 = 1$ for all PSK modulations (e.g., for BPSK and quadrature phase-shift keying (QPSK)).

Observing that the terms C_0, C_1 and C_2 do not depend on $\mathbf{x}$, we now have all elements needed to proceed with the maximization:

$$\begin{aligned}\hat{\mathbf{x}} &= \arg\max_{\mathbf{x}\in\mathcal{C}} p(\mathbf{y}|\mathbf{x}) \\ &= \arg\max_{\mathbf{x}\in\mathcal{C}} \int_{-\pi}^{\pi} e^{\sum_i x_i \operatorname{Re}\{y_i e^{-j\phi}\}/\sigma^2} \cdot \frac{1}{2\pi} \mathrm{d}\phi \\ &= \arg\max_{\mathbf{x}\in\mathcal{C}} I_0\left(\frac{1}{\sigma^4}\left|\sum_{i=1}^{N} x_i y_i\right|^2\right)\end{aligned}$$

where I_0 is the modified Bessel function of the first kind of order 0. Since I_0 has the property of being both even and, if we constrain it in the domain $[0, \infty)$, monotonically increasing, maximizing it equals maximizing its argument, such that we end up with:

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}\in\mathcal{C}} \left|\sum_{i=1}^{N} x_i y_i\right|. \tag{2.6}$$

This metric is regarded as ***noncoherent correlation***, as it is made of the product between transmitted and received signal. Whilst its computation does not require any kind of phase estimation (thus noncoherent), it must be observed that its maximization cannot be performed by means of the traditional VA. This consideration follows from the fact that the computation of the noncoherent correlation cannot be performed recursively, i.e. adding a branch transition metric that depends only on $(x_t,\, y_t)$ to a cumulative metric associated with the partial noncoherent correlation $\left|\sum_{i=1}^{t-1} x_i y_i\right|$.
This is in contrast to the coherent scenario, where, when selecting the survivor path in the Viterbi algorithm, we were sure that, whenever a branch was discarded, we were getting rid of a suboptimal path thanks to the additive properties of the squared Euclidean distance.
In the following sections we will see how this fact impacts the overall error probability of the decoder and how it is possible to optimize its performance to still provide a tangible gain with respect to the classical pilot-aided approach.

2.5. Blind Viterbi decoding

We consider next a modified version of the Viterbi algorithm, tailored to the noncoherent correlation metric [10]. In contrast to the coherent case, it shall be noted that the algorithm is not ML, i.e., it is not optimum in the sense of minimizing the block error probability. The new rules for selecting the best path are outlined next:

1. First compute the *branch metric* as the inner product of transmitted and received signal at time t

$$\lambda_t^{i,j} = \langle \mathbf{x}_t^{i,j}, \mathbf{y}_t \rangle \tag{2.7}$$

with $\mathbf{x}_t^{i,j}$ being the BPSK output of the transition from state i to state j at time t.

2. Then *select the best path* by looking at the maximum squared absolute value of the cumulative metric up to time $t-1$ plus the branch metric at time t

$$\hat{i} = \arg\max_i (|\Lambda_{t-1}^i + \lambda_t^{i,j}|^2). \tag{2.8}$$

3. Update the *cumulative metric* by summing the cumulative metric at time $t-1$ and the branch metric at time t of the best path selected in step 2

$$\Lambda_t^j = \Lambda_{t-1}^{\hat{i}} + \lambda_t^{\hat{i},j}. \tag{2.9}$$

2.6. Performance analysis

We performed Monte Carlo simulations to assess the error probability of ideal channel state information (CSI), pilot-aided and noncoherent decoding algorithms.
We fixed $\frac{E_b}{N_0}$ ranging from 0 dB up to values that ensured us to reach error rates in the order of 10^{-4}, moving by steps of 0.5 dB. We fixed a minimum number of simulations, i.e. 10000, and a minimum number of block errors, i.e. 100, to be found at each $\frac{E_b}{N_0}$ step, in order for our analysis to be sufficiently reliable.

At first we had a look at results for ZTCCs encoding 64 information bits, with memory values ranging from 4 to 8 (Table 1.1); for simplicity's sake, we will display in this section only the outcomes for the [133, 171] ZTCC, which is the optimum in terms of minimum distance ($d_{\min} = 10$) amongst all possible ones with memory $\nu = 6$. Further results with different memory values and further analysis will be presented in later sections.

As a reference, we considered a system that is supposed to have channel state information knowledge without having to perform channel estimation, thus without the need of any pilot field. This is referred as *perfect CSI* in the following plots and it is a lower bound for the error correction performance of the different solutions implemented.

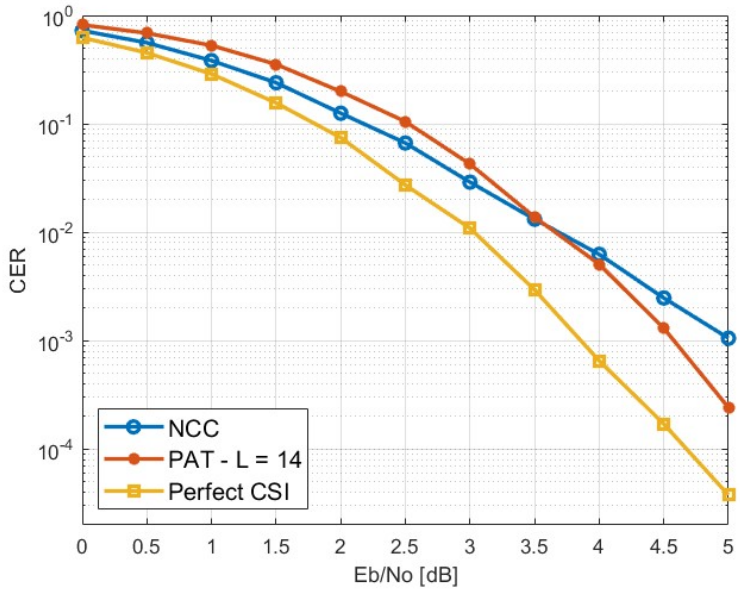

Figure 2.1: [133, 171] ZTCC, $K = 64$ information bits, noncoherent correlation (NCC) vs pilot-assisted transmission (PAT).

The performance is evaluated through the codeword error rate (CER), which corresponds to the ratio between wrong words and total transmitted words.
The chosen pilot length $L = 14$ was selected as the best by running simulations for several values of L.
As we can see, the NCC algorithm behaves better than the PAT one at low values of $\frac{E_b}{N_0}$, where it benefits from the missing pilot overhead, but then seems to reach a floor which is due to the suboptimality of the metric update rule of Section 2.5.
We observed that the floor is dependent on the memory of the code, but it is still present also at relatively large memory values.
In further sections we will have a look at different approaches to reduce the error floor and further optimize the performance of NCC decoding with respect to the PAT approach.

3 | Viterbi tracking

In this chapter we will have a look at different approaches to improve the performance of the NCC decoder with respect to the PAT one. In particular, we will see how it is possible to exploit the properties of the decoder, and especially of how and in what way errors happen, to set up a Viterbi tracking algorithm that significantly reduces the block error rate.
With this aim, we will propose two possible ways of identifying a decoding error, with their respective upsides and downsides. Results will be presented for both ZTCCs and TBCCs, as well as for a QPSK modulation.

3.1. Phase estimation error

To get rid of the floor that is visible in Figure 2.1, we tried to understand in the first place why and how errors happen, in order to realize if there exists a viable way to face this issue.

We collected then words that were decoded wrongly and looked at the number of flipped bits in each wrong word. Furthermore, we computed the ML channel estimate $\hat{h}$, assuming the decoded word $\hat{\mathbf{x}}$ to be correct, as

$$\hat{h} = \frac{\langle \mathbf{y}, \hat{\mathbf{x}} \rangle}{||\mathbf{x}||^2} = \frac{\langle \mathbf{y}, \hat{\mathbf{x}} \rangle}{N}. \tag{3.1}$$

Moreover, we denote the phase estimation error as

$$\Delta\phi = \angle\hat{h} - \angle h.$$

The notation $\angle h$ indicates the phase of the channel coefficient h. Once we had the phase estimate given by the wrongly decoded word, we then computed its difference with respect to the actual phase of the channel and then plotted the distribution of all phase errors $\Delta\phi$.

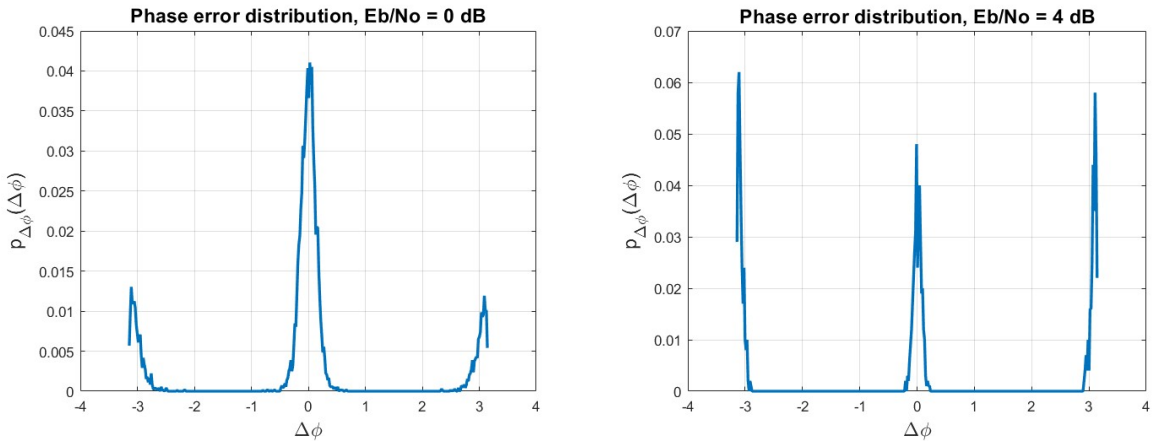

Figure 3.1: Phase estimate error distribution, $\frac{E_b}{N_0} = 0$ dB and $\frac{E_b}{N_0} = 4$ dB, ZTCC [133, 171].

It is easy to see that, whenever an error is made, the phase estimated through the codeword is either very close to the actual one of the channel or approximately shifted by $\pm\pi$. This can be justified by observing that decoding errors lead to words that have Hamming distance close to either $d_{\min}$ or $d_{\max}$ with respect to the correct one. Note that $d_{\max}$ is defined as the maximum Hamming distance between dwo distinct codewords in the codebook.

It is easy to understand why that is the case for $d_{\min}$, as it is logical that the algorithm will most likely mistake the correct word for one similar to it. The errors in the neighbourhood of $d_{\max}$ instead are a direct consequence of the pi-symmetry of the noncoherent correlation update metric.

For example, let's consider the sequence to be transmitted as a single bit $u = 0$; its corresponding zero-tail encoded word is $\mathbf{x} = \{0, 0, 0, 0\}$, which becomes $\mathbf{x} = \{+1, +1, +1, +1\}$ when BPSK mapped, while we define as $\mathbf{x}_s = \{-1, -1, -1, -1\}$ its symmetric version. Let's suppose the received signal $\mathbf{y}$ to be a random vector of four complex numbers, i.e. $\mathbf{y} = \{y_1, y_2, y_3, y_4\}$. When performing the path cancellation with the NCC algorithm, i.e. $\hat{i} = \arg\max_i(|\Lambda_{t-1}^i + \lambda_t^{i,j}|^2)$, the two branch metrics $\lambda = \langle \mathbf{x}, \mathbf{y} \rangle$ and $\lambda_s = \langle \mathbf{x}_s, \mathbf{y} \rangle$ will have the same absolute value and opposite sign, such that the two words will not be distinguishable when carrying out the maximization.

In this case, two words at Hamming distance close to $d_{\max}$ will lead to similar values of the NCC metric, thus justifying the outcomes of Figure 3.1 and Figure 3.2.

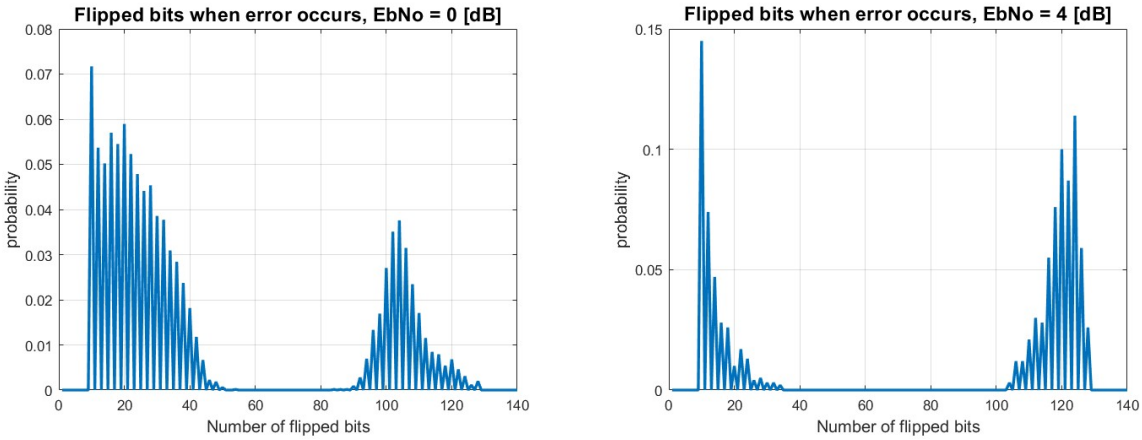

Figure 3.2: Empirical flipped bits distribution at $\frac{E_b}{N_0} = 0$ dB and $\frac{E_b}{N_0} = 4$ dB, ZTCC [133, 171].

At high values of $\frac{E_b}{N_0}$, the probability of committing errors at distances farther from $d_{\min}$ or $d_{\max}$ decreases, such that the probability density function of the phase estimate error $\Delta\phi$ gets sharper around 0 and $\pm\pi$.

For reference, Figure 3.3 shows the spectrum of the [133, 171] ZTCC, that, by displaying the Hamming weight of all words in the codebook, and therefore also its $d_{\min}$ and $d_{\max}$, further helps us understand the meaning of the results depicted in Figure 3.1 and Figure 3.2.

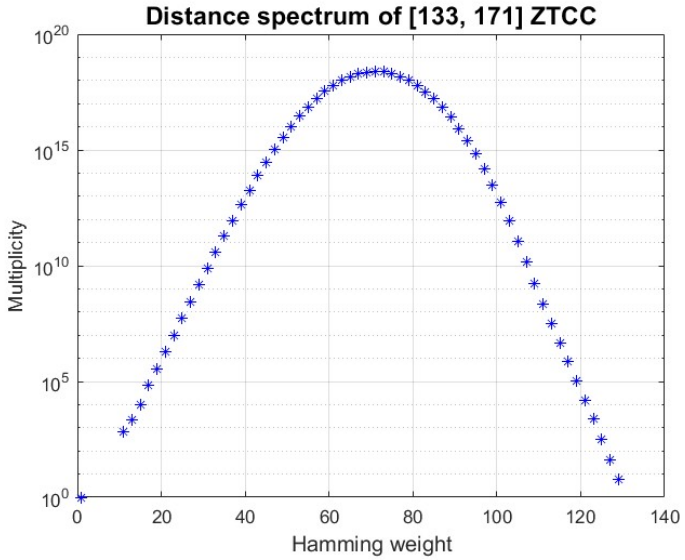

Figure 3.3: Distance spectrum of [133, 171] ZTCC ($K = 64$ information bits).

3.2. Viterbi tracking algorithm and extra loop

As a consequence of the phase estimation error $\Delta\phi$ being distributed around 0 and $\pm\pi$, we decided to implement a **Viterbi tracking algorithm (VTA)**: whenever an error is identified, it runs two different instances of the VA, once using the channel estimate $\hat{h}$ and once its rotated version $-\hat{h}$.

The first instance covers cases in which the decoded word is in the neighbourhood of d_{min} and thus has only a few flipped bits with respect to the transmitted one. The second instance instead covers cases where the decoded word is at Hamming distance close to d_{max}, with most bits being flipped with respect to the correct one.
Clearly the channel estimate will not be exact, but since most bits are still corresponding, or in the latter case the phase estimate is then rotated by π, it proves more than sufficient to yield a solid performance of the decoder.

Once we ran the two instances of the VA and we obtained the two different words, we select the one that presents the highest likelihood, thus the best NCC metric $\Lambda = \langle \mathbf{y}, \hat{\mathbf{x}}_i \rangle$. If we suppose to have perfect knowledge of the decoding errors, this procedure leads to the outcome shown in Figure 3.4:

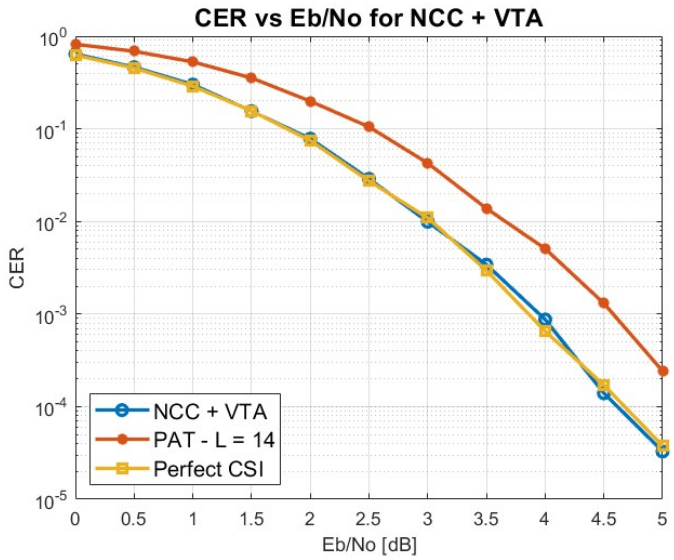

Figure 3.4: Extra loop with VTA, [133, 171] ZTCC.

As depicted above, with this solution we are able to make the NCC block error rate approach the performance of the decoder with perfect CSI. This coding gain is paid in terms of complexity, as we need two more instances of the VA when an error is committed. However, especially at higher values of SNR, where the error probability is low, this is less impacting as these further iterations are needed only a few times with respect to the total amount of received words.

Throughout this first analysis, we supposed to be able to know exactly when a decoded word is wrong, but this is obviously impossible in real systems. In regard, we considered two different practical implementations to identify decoding errors:

- Use **cyclic redundancy check (CRC) codes** [7].

- Set a **threshold** on the cumulative metric Λ below which the decoded word is supposed to be wrong.

Both approaches though presents some upsides and downsides.
Adding a CRC code allows us to detect errors with great accuracy, provided that one with a sufficiently high degree is used, but is paid through a loss in terms of rate. Once the CRC length is optimized, the resulting loss should in any case be smaller than the one introduced in PATs

Regarding the latter instead, it is not always easy to pick a good threshold value. Ideally, we would expect to enter the Viterbi tracking loop when an error is identified and avoid doing so when the received word was decoded correctly. However, given the presence of a substantial AWGN component, the distributions of the NCC metric for correct and wrong words are for some parts superposed. This means that, in order to reiterate over most wrong words, it would be necessary to enter the loop also in some cases where the decoded word was already correct, such that there exists a trade-off between accuracy and complexity of the decoder.

3.3. CRC codes

CRC codes can be used effectively to understand if the decoded word is wrong or right. In short, a very simple algorithm is used to detect possible errors and it works by adding parity bits at the end of the transmitted word. CRC codes are described by their polynomial generator $g(x)$, such that:

$$g(x) = g_0 + g_1 \cdot x + g_2 \cdot x^2 + ... + g_m \cdot x^m = 1 + \sum_{i=1}^{m-1} g_i \cdot x^i + x^m. \tag{3.2}$$

Where both the terms g_0 and g_m are constrained to be equal to 1. Such CRC code is said to be of degree m, which is in fact the order of the polynomial generator $g(x)$.
Once the polynomial generator is defined, the parity bits to be added at the end of the encoded sequence are given as the remainder of the polynomial division $\frac{u(x)\cdot x^m}{g(x)}$.
For example, let's consider the sequence $\mathbf{u} = [1\ 1\ 1\ 1]$, which corresponds to the polynomial $u(x) = x^3 + x^2 + x + 1$, and the CRC code $g = [1\ 0\ 1]$, which corresponds to the polynomial $g(x) = x^2 + 1$, with degree $m = 2$. The parity bits will be computed as the remainder of the polynomial division $\frac{u(x)\cdot x^2}{g(x)}$, and this results in the pair $[0\ 0]$, which will be appended at the end of the sequence to be encoded. Only after the parity bits are computed, the sequence is then encoded by the convolutional encoder.
The polynomial division over a binary field can be implemented efficiently through a circuit; examples of that can be found in [7].

At the decoder side, we need to perform the same operation on the received word, such that there should be no detectable errors if the remainder turns out to be equal to a vector of all zeros. If instead this is not the case, this means that the decoded word is supposedly wrong and we have to enter the Viterbi tracking loop, as we said using this time the VA with the estimated channel coefficient.

In general, the combination of block codes allows to construct more powerful channel codes. The serial concatenation of a convolutional code with an outer CRC code can be optimized by selecting the CRC polynomial that maximizes the minimum distance of the concatenated codes. This optimization has been proposed in [6, 13], and it has shown remarkable gains when the CRC code is constrained to have a small degree in the short block-length regime. In our case overall, CRC polynomials of degree three or four proved to be the most effective ones, as they represent a good trade-off between error detection capability and rate loss.

We remind that the convolutional codes used are the ones listed in Table 1.1. Here the polynomial is given in octal form, such that, for example, $g(x) = x^3 + x^2 + x + 1$ is expressed as $g = [17] = [1\ 1\ 1\ 1]$.

Optimum CRC codes for ZTCCs

	m = 2	**m = 3**	**m = 4**	**m = 5**
[27 31]	[7]	[17]	[25]	[63]
[53 75]	[7]	[11]	[21]	[51]
[133 171]	[5]	[17]	[33]	[61]
[247 371]	[5]	[11]	[31]	[41]
[561 753]	[5]	[17]	[21]	[63]

Table 3.1: Distance spectrum optimum degree-specific CRC polynomials for the ZTCCs specified in Table 1.1.

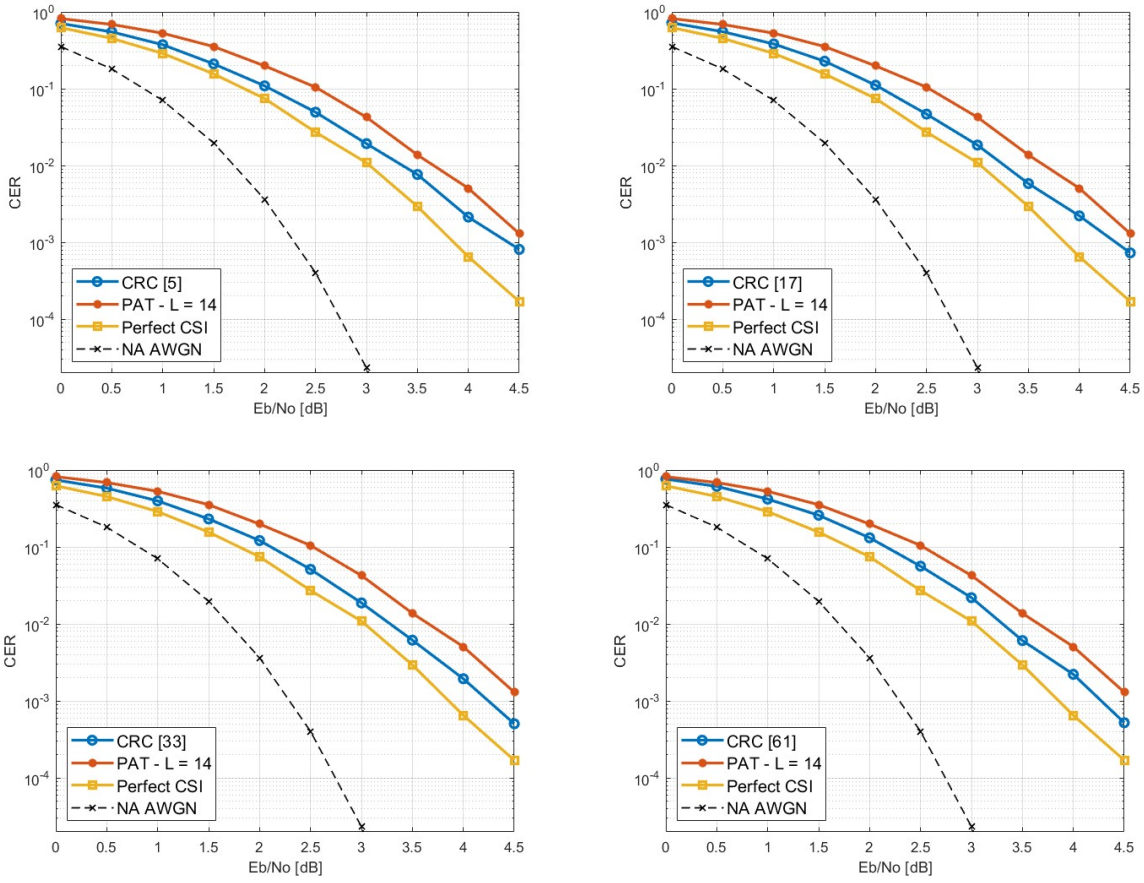

Figure 3.5: CER vs $\frac{E_b}{N_0}$ for CRC codes of degree 2 to 5, ZTCC [133, 171].

The performance of the implemented solution was evaluated through the obtained CER (Figure 3.5) as well as the complexity, which is intended in terms of average number of iterations of the VA per received word (Figure 3.6). At first we used again a ZTCC with memory $\nu = 6$, as it was done in previous sections. As a reference for our performance, we considered again the system with perfect CSI, the PAT approach with channel estimation through pilot bits as well as the normal approximation (NA) bound for the AWGN channel [8]. This is an approximation of the minimum achievable CER and it can be obtained under ML decoding of an ensemble of random codes at fixed rate and block-length.

Although the parity bits of the CRC introduce a rate loss, the CER of the proposed decoder is consistently lower than the PAT one, with a margin of approximately 0.3 dB up to error rates below 10^{-3}. Eventually, all CRC code orders show a valuable performance: lower order codes are closer to the perfect CSI at low $\frac{E_b}{N_0}$, as they suffer less from the cost of the added overhead. At high $\frac{E_b}{N_0}$ instead, higher order codes seem to be performing a little better, but overall the difference is mostly negligible up to the error rates considered.

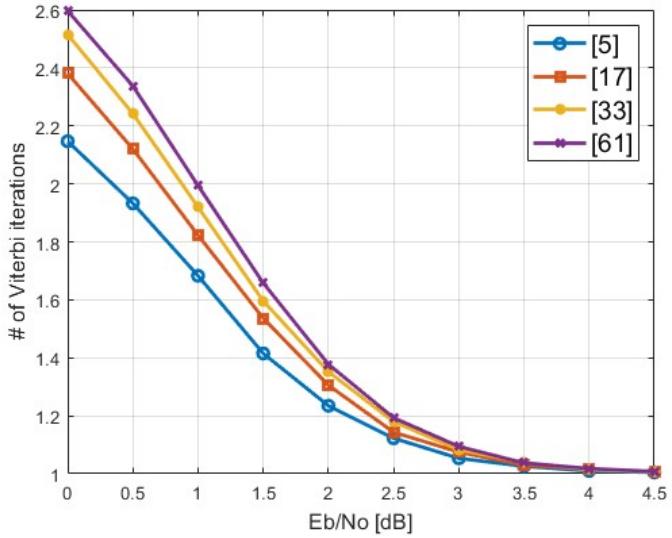

Figure 3.6: Average number of VA iterations vs $\frac{E_b}{N_0}$ for CRC codes of degree 2 to 5, ZTCC [133, 171].

Clearly, at low values of $\frac{E_b}{N_0}$ the number of iterations is larger since many received words will be decoded wrongly, which means entering the loop a larger number of times. Moving towards higher $\frac{E_b}{N_0}$ values, the number of iterations required quickly approaches one, meaning that the complexity of the system is not far from the PAT one. If we compare instead CRC codes with different order, we see that the larger order ones are able to detect errors with greater accuracy and thus enter the tracking loop more often.

3.4. Threshold on cumulative metric

The second option is to set a threshold on the cumulative metric for the NCC algorithm, $\Lambda = \langle \mathbf{y}, \hat{\mathbf{x}} \rangle$, in order to perform the extra iteration step only in the cases where the value of Λ is lower than the predefined threshold.
To determine a value Λ_{thr} of the threshold that allows for efficient performance and good error correction capabilities, it is paramount to understand how the variable Λ is distributed for both correct words and wrong ones, in order to reiterate on a sufficient number of incorrect words and avoid doing so when the decoded word is already correct. We decided to collect, at each $\frac{E_b}{N_0}$ step, 1000 wrong words and their cumulative metric Λ. We then set three different values of percentage of errors on which the system performs the Viterbi tracking loop, namely 75%, 90% and 99%. We fixed the threshold by converting this parameters into the corresponding Λ value through the distributions computed empirically, an example of which is shown in Figure 3.7.

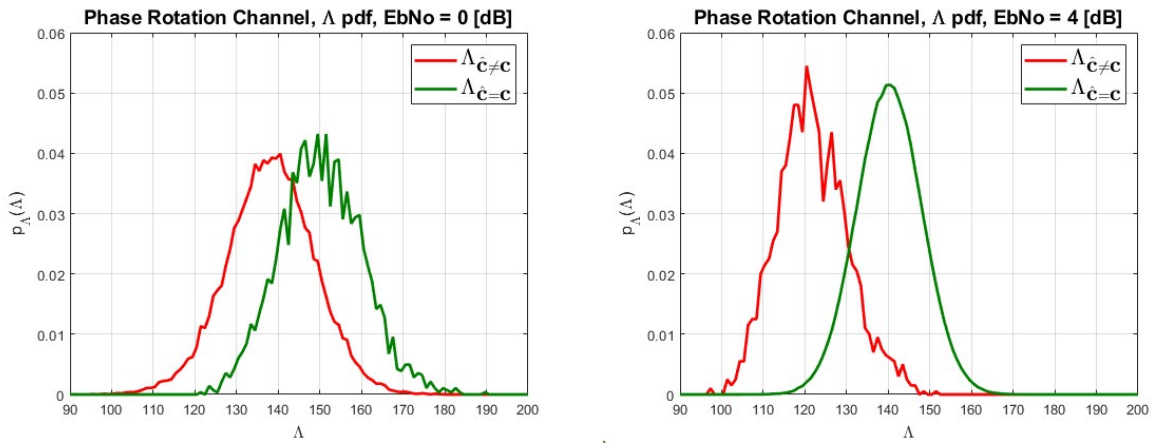

Figure 3.7: Distribution of $\Lambda = \langle \mathbf{y}, \hat{\mathbf{x}} \rangle$, [133, 171] ZTCC.

At higher SNR values and larger code memory lengths the distributions tend to get respectively sharper and more separated one from the other, allowing for a more efficient performance of this approach. This is still a consequence of how errors happen, as was shown in Figure 3.2.

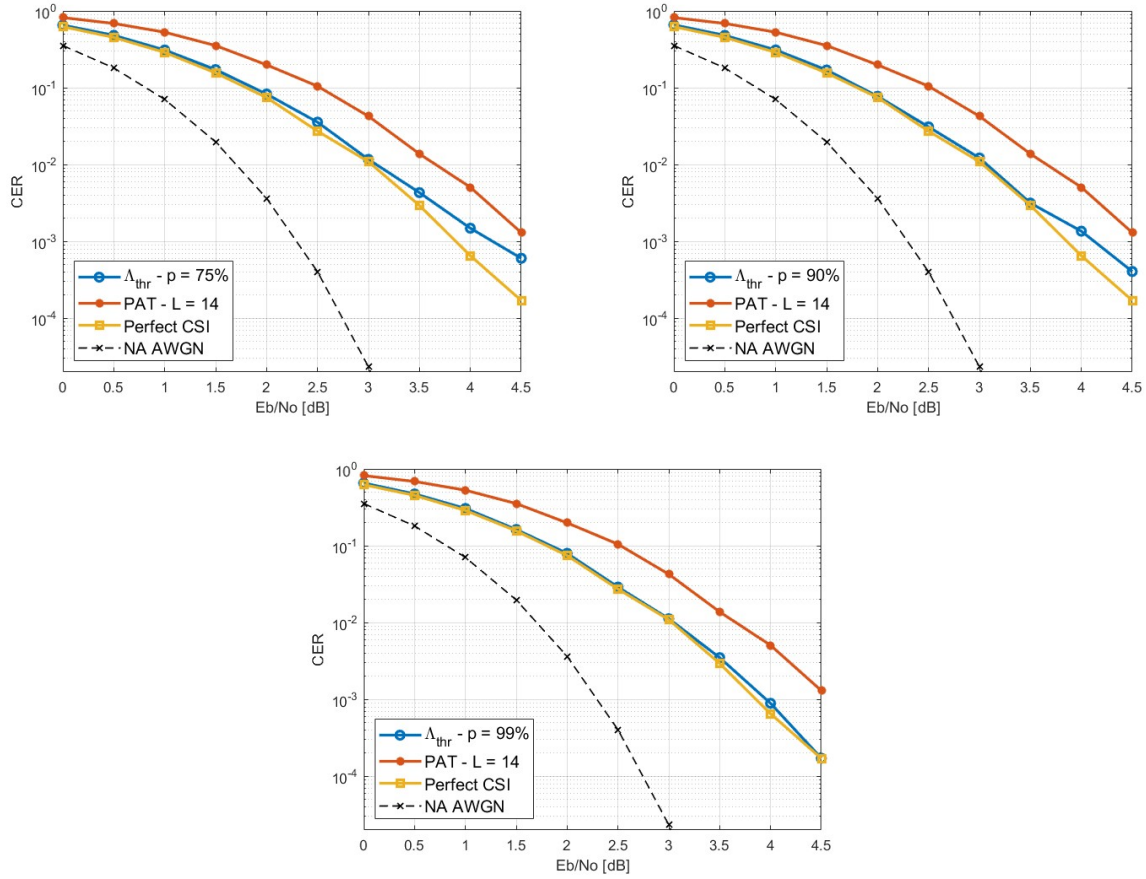

Figure 3.8: CER vs $\frac{E_b}{N_0}$ for different Λ_{thr} setups, ZTCC [133, 171].

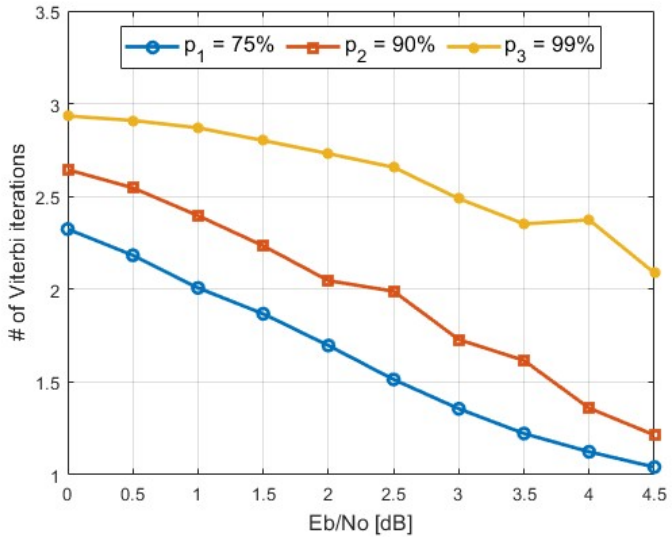

Figure 3.9: Average number of VA iterations vs $\frac{E_b}{N_0}$ for different Λ_{thr} ZTCC [133, 171].

As depicted in Figure 3.8, the algorithm shows an optimal behaviour up to error rates of 10^{-3}, for all threshold values selected, such that the gap to the perfect CSI curve is almost non-existent. At higher SNR values instead, the curves corresponding to lower probabilities of entering the Viterbi tracking loop tend to get further from the desired one, whilst the one at $p = 99\%$ is still overlaying with the optimum one. This means that, if we identify a sufficient number of decoding errors, we are able to successfully correct them and approach the perfect CSI curve.
This is paid in terms of complexity (Figure 3.9): the system has to perform several iterations of the VA even when it is not needed. This is due to the fact that the distributions of the cumulative metric of wrong and correct words are for a good part superposed.

3.5. Tailbiting terminated codes

As mentioned in Section 1.1.4, ZTCCs require a termination that translates into bits which do not carry any information and are thus an overhead on the rate. To further limit rate losses, we tried tailbiting convolutional codes instead, which do not need additional bits of termination. In fact, the only requirement is that the initial and ending state coincide, but that is not anymore constrained to be the all zero state.
However, this creates some problems with the implementation of the proposed blind metric due to symmetry reasons, as some of the optimal CCs in terms of minimum distance and multiplicity (e.g. the [133, 171] memory 6 code and the [561, 753] memory 8 code) present in their codebook the all ones codeword. This introduces a pi-symmetry, meaning that any codeword and its flipped version belong to the codebook.

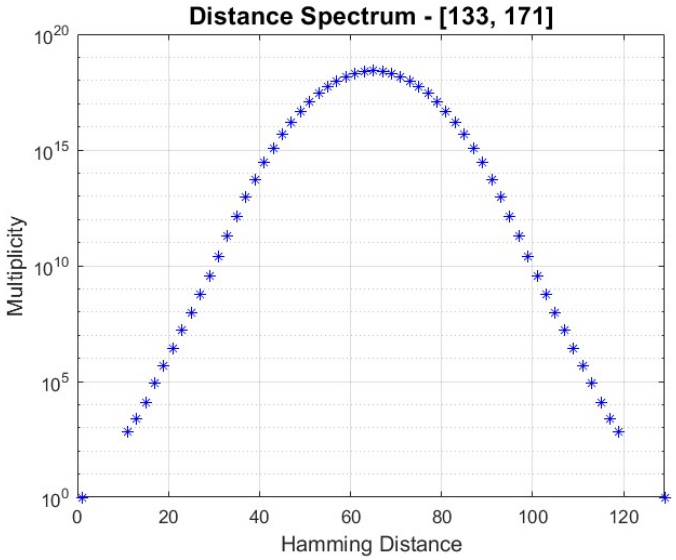

Figure 3.10: Distance spectrum of [133, 171] TBCC.

This fact prevents the use of the ML decoder straight-forward, being the noncoherent correlation metric the same for two distinct codewords of the codebook.
To get rid of this issue, two different solutions are proposed:

- Find the best CC which has not the all-one codeword in its codebook, with "best" being defined in the following paragraph.
- Add an extra bit as a check to the encoded message to eliminate the symmetry.

3.5.1. Suboptimal TBCCs

For the first approach we developed new metrics to identify the best suboptimal code: as highlighted in Section 3.1, due to the symmetry problems discussed, we need to take into account not only the minimum distance of the code, but also its maximum distance. The best polynomial is now characterized by the following parameters:

$$d_{best} = \min(d_{\min},\ N - d_{\max})$$

$$A_{best} = A(d_{best}) + A(N - d_{best}) \tag{3.3}$$

With A_{best} being the multiplicity of the lowest Hamming distance words of the best code according to this metric.
In Table 3.2 we highlighted the best TBCCs for memory $\nu = 4, ..., 8$ alongside their values of d_{best} and A_{best}.

	$\nu = 4$	$\nu = 5$	$\nu = 6$	$\nu = 7$	$\nu = 8$
G(D)	[27, 31]	[53, 75]	[105, 167]	[247, 371]	[663, 711]
$\mathbf{d_{best}}$	7	8	9	10	11
$\mathbf{A_{best}}$	128	64	64	64	128

Table 3.2: Optimum TBCCs according to metric in eq. (3.3).

To evaluate the performance of this solution, we repeated the simulations performed in Section 3.4, thus collecting the distributions of Λ and setting a value Λ_{thr} according to a percentage of errors we want the decoder to reiterate on.

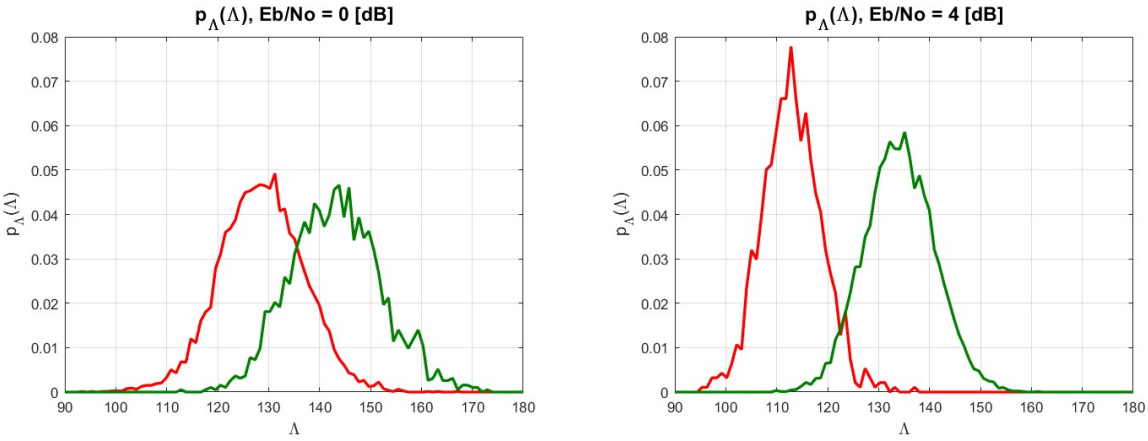

Figure 3.11: Distribution of $\Lambda = \langle \mathbf{y}, \hat{\mathbf{x}} \rangle$, [105, 167] TBCC.

The shape and behaviour of the Λ distribution is practically unchanged with respect to the zero-tail case, apart from a shift to the left due to the different word length.

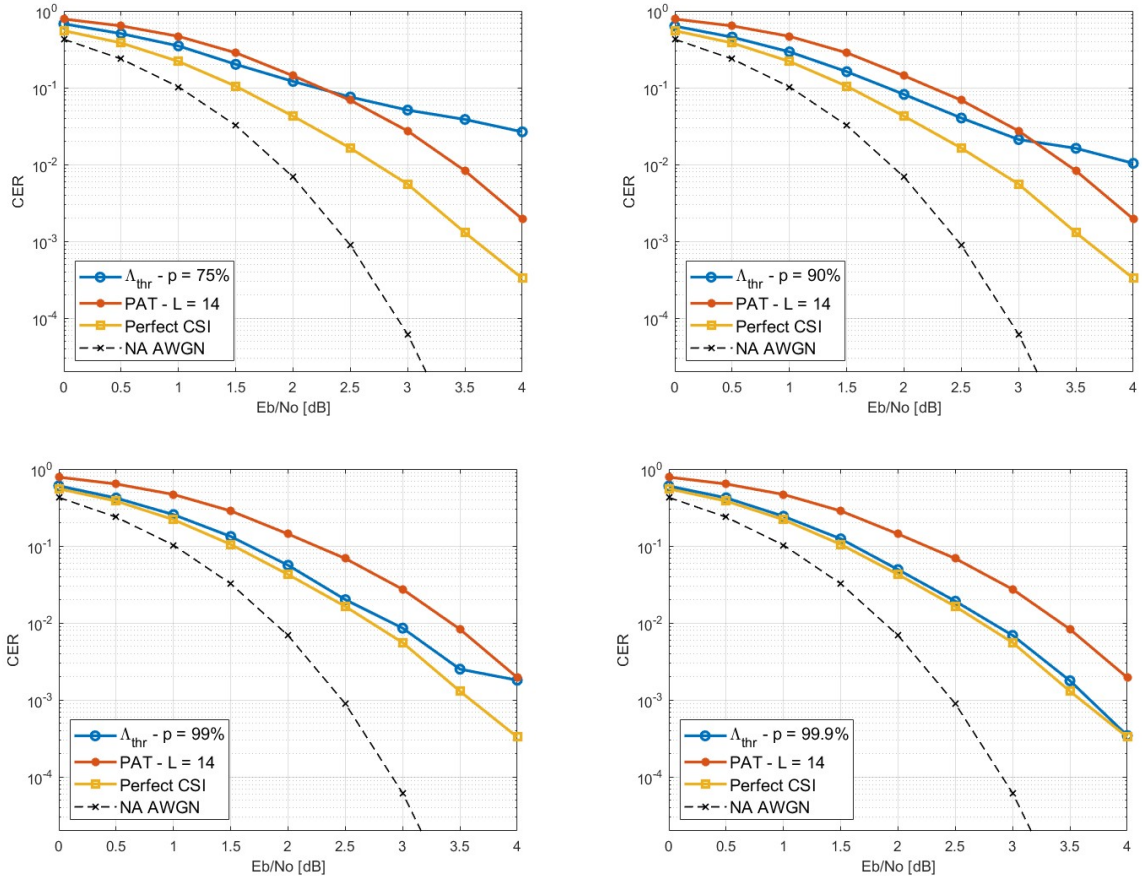

Figure 3.12: CER vs $\frac{E_b}{N_0}$ for different Λ_{thr} setups, TBCC [105, 167].

Regarding the CER performance, we can see that this time the algorithm tends to perform worse when it does not enter the loop; this means it is necessary to reiterate over a higher percentage of incorrect words in order to achieve a good error correction capability. The curve at $p = 99.9\%$ approaches almost perfect the perfect CSI error rate.

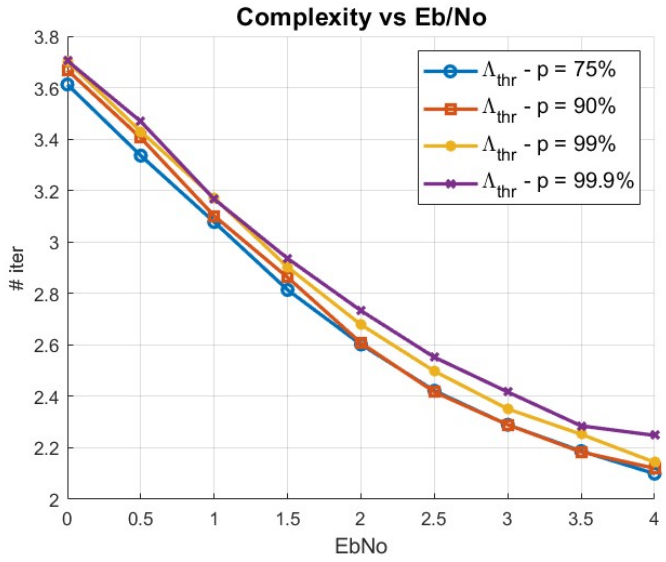

Figure 3.13: Average number of VA iterations vs $\frac{E_b}{N_0}$ for different Λ_{thr} setups, ZTCC [133, 171].

However, this does not affect significantly the complexity of the system, as the average number of runs of the VA is just slightly shifted at different looping probabilities. Overall, the usage of the WAVA algorithm already requires more iterations on average with respect to the ZTCC case. Nevertheless, compared to Figure 3.9, the difference between the two is reasonable and seems to close at higher $\frac{E_b}{N_0}$ values.
The second solution considered was to adopt CRC codes optimized together with the suboptimal TBCCs, in the same way it was done in Section 3.3.

	$m = 2$	$m = 3$	$m = 4$	$m = 5$
[27, 31]	[7]	[11]	[21]	[63]
[53, 75]	[7]	[11]	[21]	[77]
[105, 167]	[5]	[17]	[33]	[57]
[247, 371]	[5]	[17]	[21]	[63]
[663, 711]	[5]	[11]	[21]	[63]

Table 3.3: Optimum CRC codes for TBCCs for different memory values

This time however, since the NCC algorithm performs less efficiently at the first stage, CRC codes with higher order would be needed to identify a sufficient number of errors and guarantee an acceptable block error rate.
Yet, this is in contrast with the original principle of reducing the rate loss in order to improve the performance of the decoder. In this sense, looking for a way to improve the behavior of the NCC metric at the first stage, for example exploiting list decoding techniques, could be an interesting topic for future studies.

3.5.2. Additional bit check

The second approach instead consists in adding a check bit, in our case a 0, at the end of the message to be encoded. If the decoded word presents a 1 as a last bit, we can assume it has been rotated by π and flip all estimated bits. By doing this, we also allow for a lower complexity, as we do not need to run two times the VA with two different phase values when we do commit a decoding error in the blind decoding phase, but just one run of the VA is sufficient.

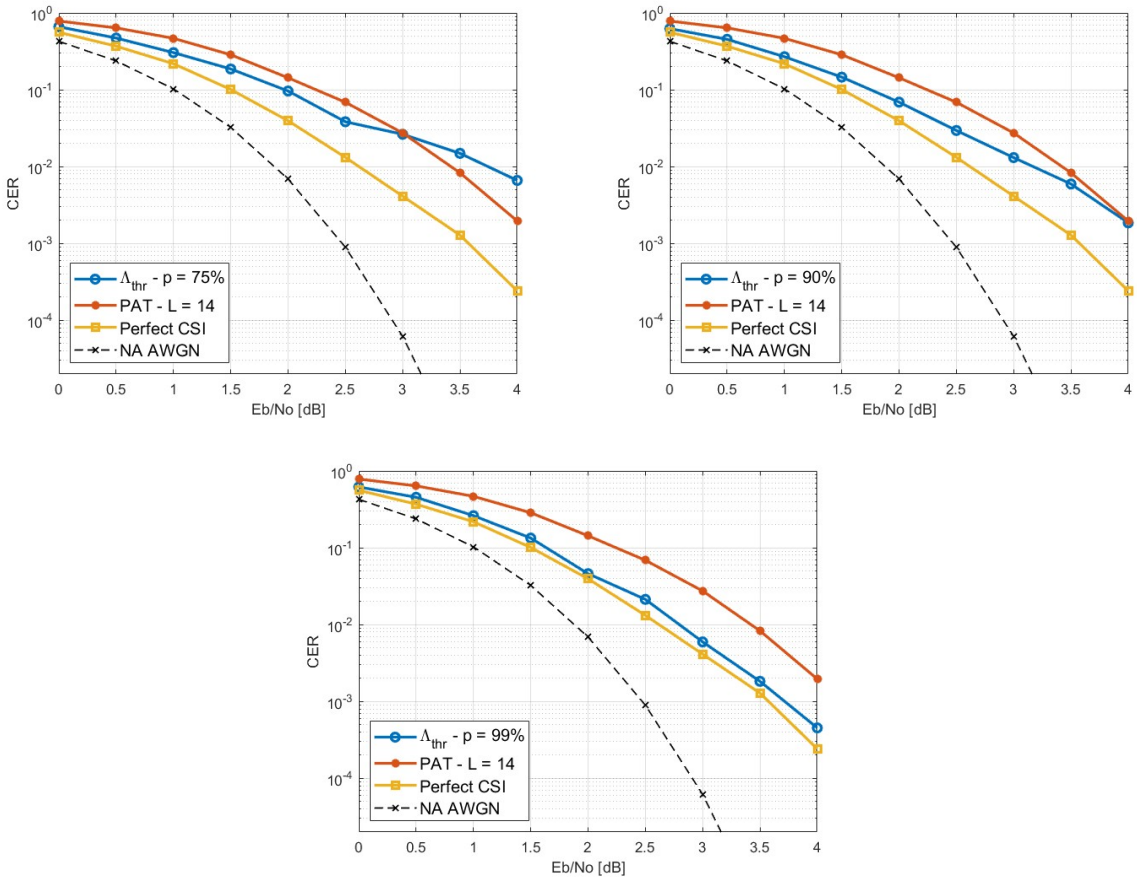

Figure 3.14: CER vs $\frac{E_b}{N_0}$ for different Λ_{thr} setups, TBCC [133, 171] with extra bit check.

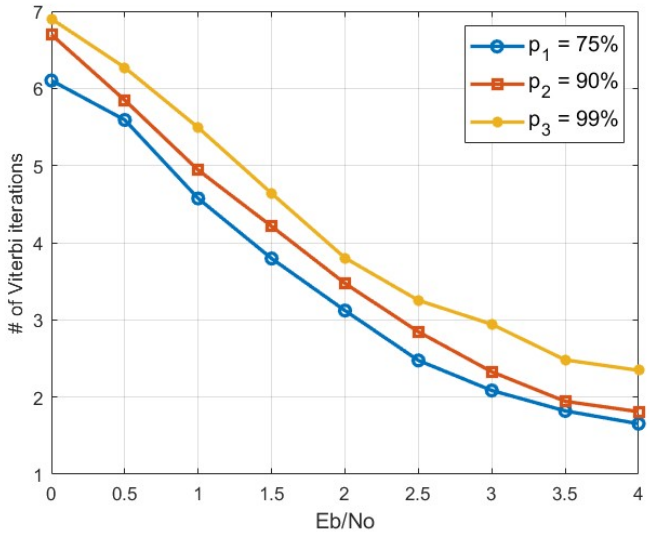

Figure 3.15: Average number of VA iterations vs $\frac{E_b}{N_0}$ for different Λ_{thr} setups, TBCC [133, 171] with additional bit check.

3.6. QPSK modulation

For completeness, we repeated the analysis with a QPSK modulation, which is possibly the most used one for this kind of application. In this configuration, two bits are mapped onto a single symbol, as the information is conveyed through both the real and imaginary components.

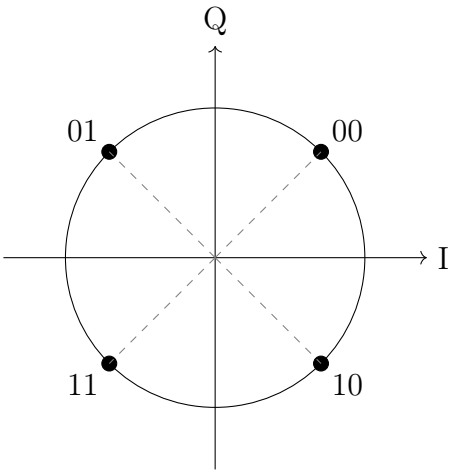

Figure 3.16: QPSK constellation.

As a consequence of this, wrong words show a phase error centered not only around 0 and $\pm\pi$ but also around $\pm\frac{\pi}{2}$, due to the constellation shape.

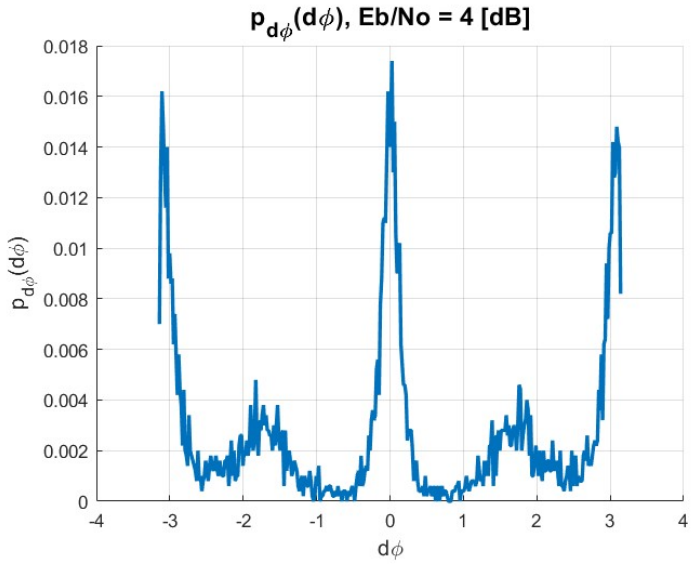

Figure 3.17: Phase error distribution, $\frac{E_b}{N_0} = 4$ dB, ZTCC [133, 171], QPSK modulation.

This means that, when performing the loop after the NCC decoding stage, we need to run four instances of the VA, thus increasing complexity with respect to the BPSK case.

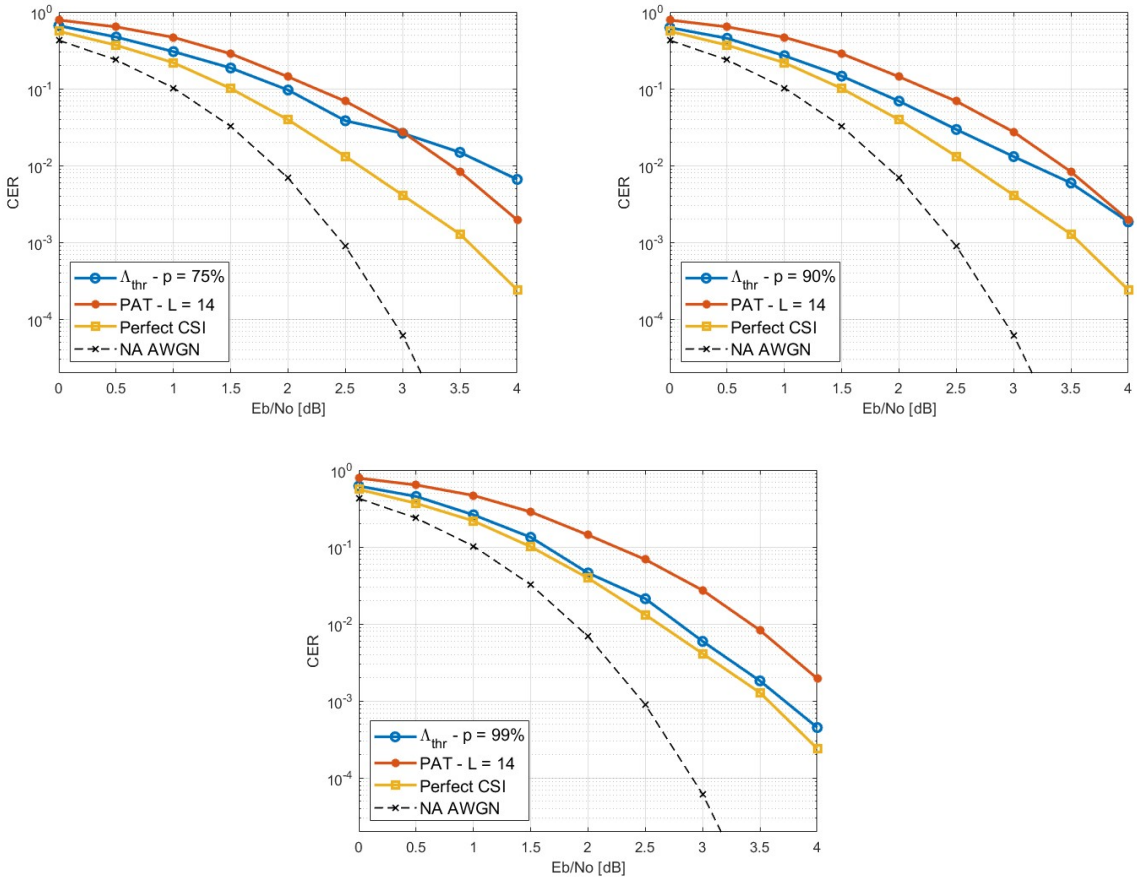

Figure 3.18: CER vs $\frac{E_b}{N_0}$ for different Λ_{thr} setups, ZTCC [133, 171], QPSK modulation.

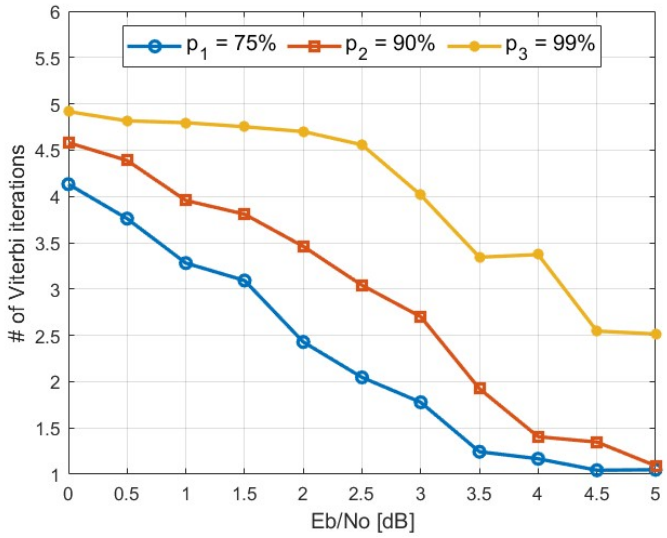

Figure 3.19: Average number of VA iterations vs $\frac{E_b}{N_0}$ for different Λ_{thr} setups, ZTCC [133, 171], QPSK modulation.

Compared to the BPSK modulation case, the NCC algorithm is just slightly more off with respect to the perfect CSI curve and that is due to the shape of the QPSK constellation (Figure 3.19). Nonetheless, when iterating over a sufficient number of wrong words, the algorithm still shows an excellent behavior, displaying a consistent gain with respect to the PAT decoder.

Even if the Viterbi tracking loop translates into double the runs of the VA of the BPSK case, the overall complexity shown in Figure 3.19 is very similar to the one of Figure 3.9, especially at high $\frac{E_b}{N_0}$ values.

3.7. Memory 8 codes

As a last analysis, we tried implementing the solutions proposed previously in this chapter for memory 8 codes as well, looking for further performance improvements.

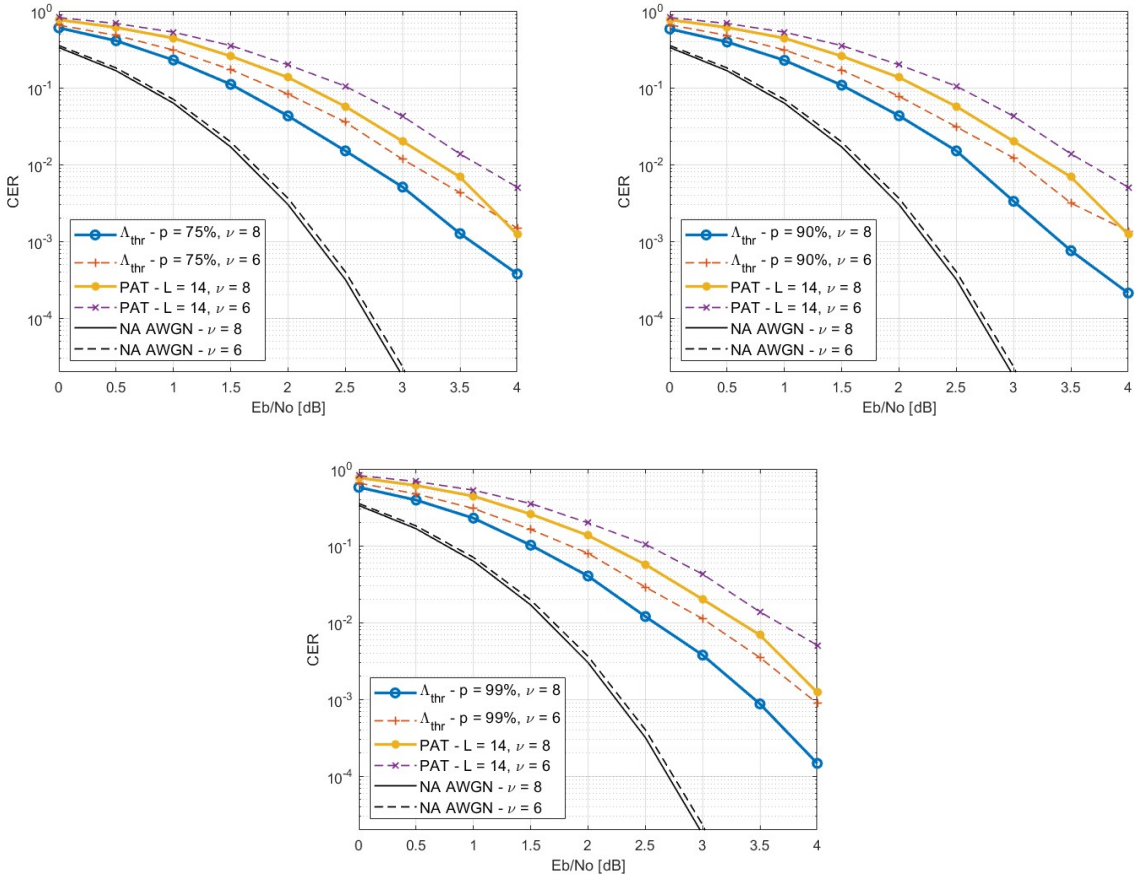

Figure 3.20: CER vs $\frac{E_b}{N_0}$ for different Λ_{thr} setups, ZTCC [561, 753].

The decoder shows an excellent behavior up to error rates of 10^{-4}, with a consistent gain around 0.5 dB with respect to the PAT curve (Figure 3.20). The CER is also closer to the normal approximation bound compared to the memory 6 code. We also note that the NCC memory 6 decoder has a better error rate than the memory 8 PAT one: this means we can basically have a gain in memory, as we can achieve the performance of a memory 8 decoder while using a memory 6 one.

The average number of Viterbi iterations, as shown in Figure 3.21, is even lower than the one of the [133, 171] code seen in Figure 3.9. The distance between the distributions of Λ for wrong and correct words is in fact dependent on the code memory, such that the larger ν, the farther the distributions. This allows to enter the Viterbi tracking loop in less cases where the decoded word is already correct after the blind decoding stage.

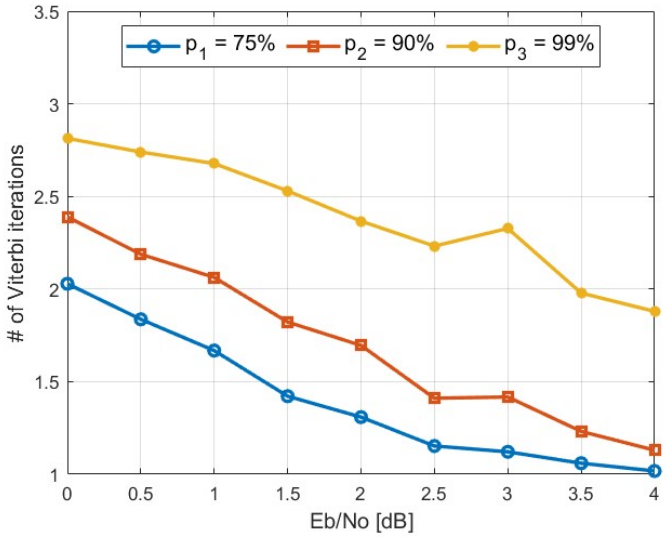

Figure 3.21: Average number of VA runs for different Λ_{thr} setups, ZTCC [561, 753].

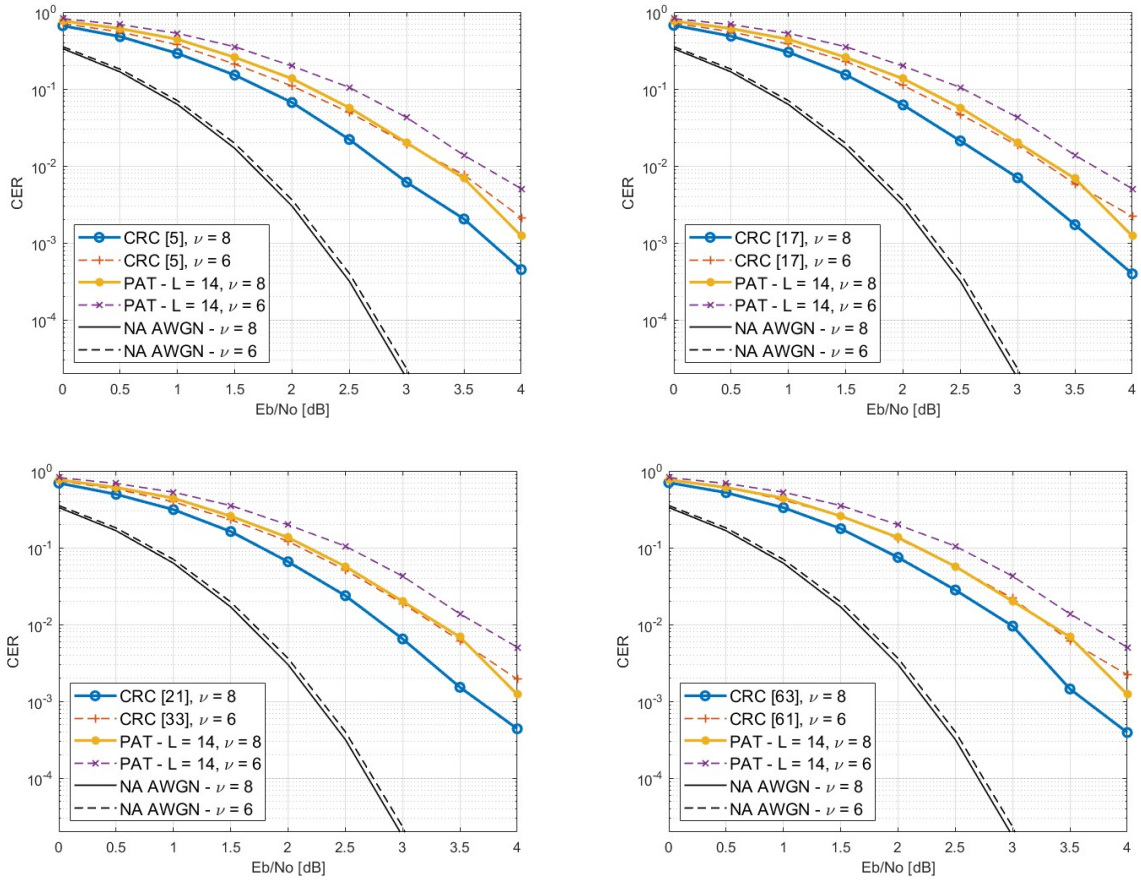

Figure 3.22: CER vs $\frac{E_b}{N_0}$ for CRC codes of order 2 to 5, ZTCC [561, 753].

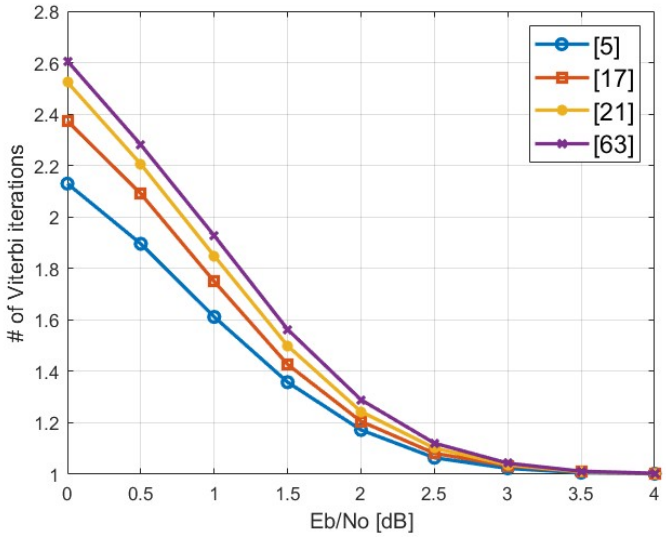

Figure 3.23: Average number of VA runs for CRC codes of order 2 to 5, ZTCC [561, 753].

In Figure 3.22 we can see that using CRC codes with degree 2 to 5 we can consistently achieve a gain of approximately 0.3 to 0.4 dB. The complexity (Figure 3.23) of the decoder tends to approach one at increasing $\frac{E_b}{N_0}$.

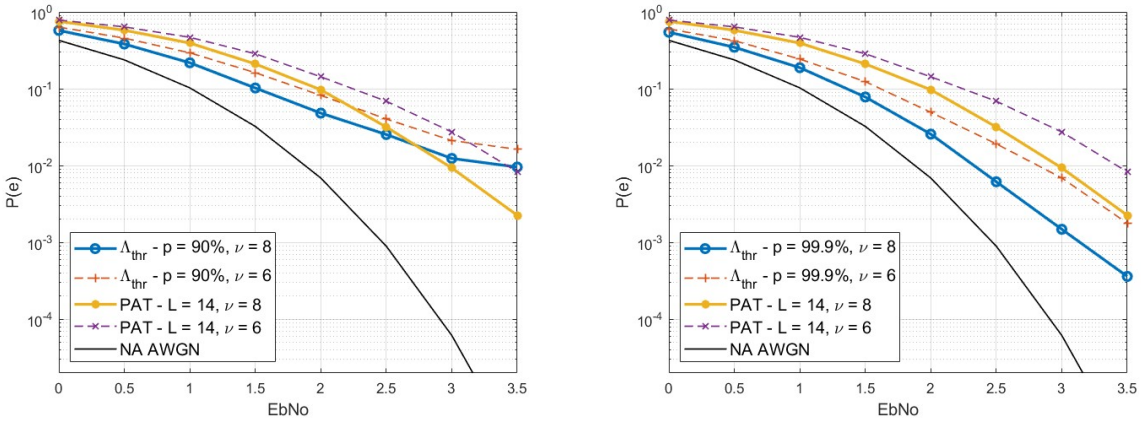

Figure 3.24: CER vs $\frac{E_b}{N_0}$ for different Λ_{thr} setups, TBCC [663, 711].

As seen in Figure 3.24, when using tailbiting codes we are able to further close the gap to the normal approximation bound down to values lower than 0.5 dB up to error rates of 10^{-3}. This is paid through an increase in complexity (Figure 3.25) with respect to ZTCCs, as was already pointed out when discussing the performance of memory 6 codes.

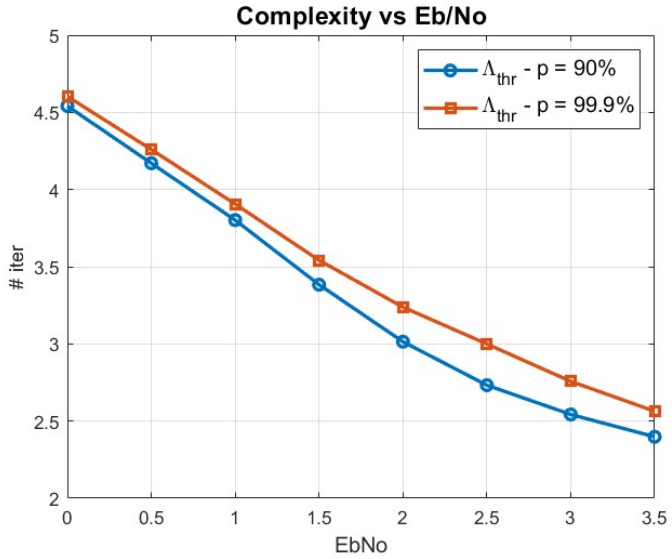

Figure 3.25: Average number of VA iterations for different Λ_{thr} setups, TBCC [663, 711].

4 | Conclusions

Throughout the discussion, we were able to demonstrate the effectiveness of convolutional codes in the framework of short-packet transmission.

After developing the noncoherent metric, it was possible to successfully exploit some properties of the decoder, namely of the nature of decoding errors, to come up with a few options to improve the performance with respect to the PAT system.

In a few alternative settings, this approach outperformed the classical pilot-aided decoder, consistently showing gains of 0.3 to 0.5 dB up to error rates of 10^{-4} and in some cases approaching the perfect CSI curve. Furthermore, tailbiting codes with memory 8 even showed error rates within 0.5 dB of the normal approximation AWGN bound.

These improvements were observed for both BPSK and QPSK modulation schemes.

A central theme throughout the discussion was the trade-off between error correction capabilities and decoding complexity. While a higher complexity ensures the best error rate performance, it was shown that well-balanced compromises can still deliver highly satisfactory results.

Bibliography

[1] M. C. Coşkun, G. Durisi, T. Jerkovits, G. Liva, W. Ryan, B. Stein, and F. Steiner. Efficient error-correcting codes in the short blocklength regime. *Physical Communication*, 34:66–79, 2019.

[2] P. Elias. Coding for noisy channels. In *IRE WESCON Convention Record, 1955*, volume 2, pages 94–104, 1955.

[3] G. Forney Jr. Review of random tree codes. *NASA Ames Research Center, Moffett Field, CA, USA, Tech. Rep. NASA CR73176*, 1967.

[4] L. Gaudio, T. Ninacs, T. Jerkovits, and G. Liva. On the performance of short tail-biting convolutional codes for ultra-reliable communications. In *SCC 2017; 11th International ITG Conference on Systems, Communications and Coding*, pages 1–6. VDE, 2017.

[5] L. Gaudio, B. Matuz, T. Ninacs, G. Colavolpe, and A. Vannucci. Approximate ml decoding of short convolutional codes over phase noise channels. *IEEE Communications Letters*, 24(2):325–329, 2019.

[6] C.-Y. Lou, B. Daneshrad, and R. D. Wesel. Convolutional-code-specific crc code design. *IEEE Transactions on Communications*, 63(10):3459–3470, 2015.

[7] W. W. Peterson and D. T. Brown. Cyclic codes for error detection. *Proceedings of the IRE*, 49(1):228–235, 1961.

[8] Y. Polyanskiy, H. V. Poor, and S. Verdú. Channel coding rate in the finite blocklength regime. *IEEE Transactions on Information Theory*, 56(5):2307–2359, 2010.

[9] W. Sui, B. Towell, A. Asmani, H. Yang, H. Grissett, and R. D. Wesel. Crc-aided high-rate convolutional codes with short blocklengths for list decoding. *IEEE Transactions on Communications*, 2023.

[10] G. Taricco and E. Biglieri. Space-time decoding with imperfect channel estimation. *IEEE Transactions on Wireless Communications*, 4(4):1874–1888, 2005.

[11] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269, 1967.

[12] L. Wang, D. Song, F. Areces, T. Wiegart, and R. D. Wesel. Probabilistic shaping for trellis-coded modulation with crc-aided list decoding. *IEEE Transactions on Communications*, 71(3):1271–1283, 2023.

[13] H. Yang, L. Wang, V. Lau, and R. D. Wesel. An efficient algorithm for designing optimal crcs for tail-biting convolutional codes. In *2020 IEEE International Symposium on Information Theory (ISIT)*, pages 292–297. IEEE, 2020.

[14] H. Yang, E. Liang, M. Pan, and R. D. Wesel. Crc-aided list decoding of convolutional codes in the short blocklength regime. *IEEE Transactions on Information Theory*, 68(6):3744–3766, 2022.

List of Figures

1.1 Encoder of [7, 5] CC. 5
1.2 FSM diagram representation of [7, 5] code. 7
1.3 Trellis diagram of the [7, 5] ZTCC, $K = 2$ information bits. 8
1.4 Viterbi update rule over the code trellis. 11

2.1 [133, 171] ZTCC, $K = 64$ information bits, NCC vs PAT. 19

3.1 Phase estimate error distribution, $\frac{E_b}{N_0} = 0$ dB and $\frac{E_b}{N_0} = 4$ dB, ZTCC [133, 171]. 22
3.2 Empirical flipped bits distribution at $\frac{E_b}{N_0} = 0$ dB and $\frac{E_b}{N_0} = 4$ dB, ZTCC [133, 171]. 23
3.3 Distance spectrum of [133, 171] ZTCC ($K = 64$ information bits). 23
3.4 Extra loop with VTA, [133, 171] ZTCC. 24
3.5 CER vs $\frac{E_b}{N_0}$ for CRC codes of degree 2 to 5, ZTCC [133, 171]. 27
3.6 Average number of VA iterations vs $\frac{E_b}{N_0}$ for CRC codes of degree 2 to 5, ZTCC [133, 171]. 28
3.7 Distribution of $\Lambda = \langle \mathbf{y}, \hat{\mathbf{x}} \rangle$, [133, 171] ZTCC. 29
3.8 CER vs $\frac{E_b}{N_0}$ for different Λ_{thr} setups, ZTCC [133, 171]. 30
3.9 Average number of VA iterations vs $\frac{E_b}{N_0}$ for different Λ_{thr} ZTCC [133, 171]. 30
3.10 Distance spectrum of [133, 171] TBCC. 31
3.11 Distribution of $\Lambda = \langle \mathbf{y}, \hat{\mathbf{x}} \rangle$, [105, 167] TBCC. 33
3.12 CER vs $\frac{E_b}{N_0}$ for different Λ_{thr} setups, TBCC [105, 167]. 33
3.13 Average number of VA iterations vs $\frac{E_b}{N_0}$ for different Λ_{thr} setups, ZTCC [133, 171]. 34
3.14 CER vs $\frac{E_b}{N_0}$ for different Λ_{thr} setups, TBCC [133, 171] with extra bit check. 35
3.15 Average number of VA iterations vs $\frac{E_b}{N_0}$ for different Λ_{thr} setups, TBCC [133, 171] with additional bit check. 36
3.16 QPSK constellation. 36
3.17 Phase error distribution, $\frac{E_b}{N_0} = 4$ dB, ZTCC [133, 171], QPSK modulation. 37
3.18 CER vs $\frac{E_b}{N_0}$ for different Λ_{thr} setups, ZTCC [133, 171], QPSK modulation. . 37

3.19 Average number of VA iterations vs $\frac{E_b}{N_0}$ for different Λ_{thr} setups, ZTCC [133, 171], QPSK modulation. 38
3.20 CER vs $\frac{E_b}{N_0}$ for different Λ_{thr} setups, ZTCC [561, 753]. 39
3.21 Average number of VA runs for different Λ_{thr} setups, ZTCC [561, 753]. . . 40
3.22 CER vs $\frac{E_b}{N_0}$ for CRC codes of order 2 to 5, ZTCC [561, 753]. 40
3.23 Average number of VA runs for CRC codes of order 2 to 5, ZTCC [561, 753]. 41
3.24 CER vs $\frac{E_b}{N_0}$ for different Λ_{thr} setups, TBCC [663, 711]. 41
3.25 Average number of VA iterations for different Λ_{thr} setups, TBCC [663, 711]. 42

List of Tables

1.1 Optimum CCs for different memory values ν. 6
1.2 State transition table of [7, 5] code. 6

3.1 Distance spectrum optimum degree-specific CRC polynomials for the ZTCCs specified in Table 1.1. 27
3.2 Optimum TBCCs according to metric in eq. (3.3). 32
3.3 Optimum CRC codes for TBCCs for different memory values 34

Acronyms

AWGN additive white Gaussian noise

BPSK binary phase-shift keying

CC convolutional code

CER codeword error rate

CRC cyclic redundancy check

CSI channel state information

FSM finite state machine

ML maximum likelihood

NA normal approximation

NCC noncoherent correlation

PAT pilot-assisted transmission

QPSK quadrature phase-shift keying

SNR signal-to-noise ratio

TBCC tailbiting terminated convolutional code

VA Viterbi algorithm

VTA Viterbi tracking algorithm

WAVA wrap-around Viterbi algorithm

ZTCC zero-tail terminated convolutional code