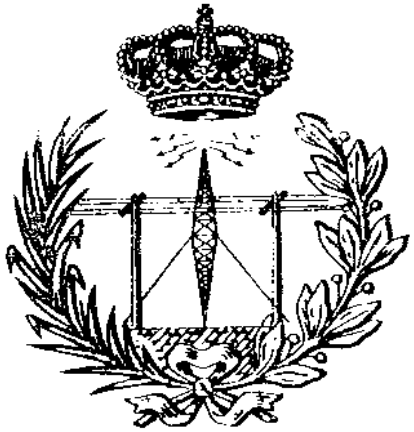
**UNIVERSIDAD DE SEVILLA**

**ESCUELA SUPERIOR DE INGENIEROS**

**INGENIERO DE TELECOMUNICACIÓN**



**PROYECTO FIN DE CARRERA**

# ESTUDIO DE LAS PRESTACIONES DE UN DECODIFICADOR VITERBI PARA DVB-T

**AUTORA: ELISA GARZÓN MAESTRE**

**TUTOR: JORGE CHÁVEZ ORZÁEZ**

**JUNIO 2004**

## Tabla de Contenidos

**1 ACRÓNIMOS.....3**

**2 BREVE DESCRIPCIÓN. .....5**

**3 MEMORIA DESCRIPTIVA: DESCRIPCIÓN DE UN SISTEMA DE TELECOMUNICACIONES. .....6**

3.1 CODIFICADOR DE FUENTE. .....8
*3.1.1 MODELADO DE FUENTES DE INFORMACIÓN. .....8*
3.1.1.1 Características estadísticas. .....8
3.1.1.2 Modelado matemático de la fuente. .....9
3.1.1.3 Tipos de modelos de fuentes. .....10
*3.1.1.3.1 Fuente discreta sin memoria. (DMS Discrete Memoryless Source) .....10*
*3.1.1.3.2 Fuente con memoria. .....10*
*3.1.1.3.3 Fuente analógica. .....10*
*3.1.2 MEDIDA DE LA INFORMACIÓN. .....11*
3.1.2.1 Entropía, indeterminación o incertidumbre de la fuente. .....11
3.1.2.2 Entropía conjunta y condicional. .....12
*3.1.3 TEOREMA DE LA CODIFICACIÓN DE FUENTE. .....13*
3.1.3.1 Fuente discreta con memoria (DMS). .....13
3.1.3.2 Fuentes con memoria. .....13
3.2 CODIFICACIÓN DE CANAL. .....14
*3.2.1 ARQ. CÓDIGOS DE PETICIÓN DE RETRANSMISIÓN AUTOMÁTICA. .....15*
*3.2.2 FECC. CÓDIGOS CORRECTORES EN RECEPCIÓN. .....15*
3.2.2.1 Códigos de bloque. .....16
*3.2.2.1.1 Códigos de bloque lineales. .....16*
3.2.2.1.1.1 Códigos cíclicos. .....16
3.2.2.2 Códigos convolucionales. .....16
3.3 CODIFICADOR DE LINEA. .....18
*3.3.1 TIPOS DE CÓDIGOS DE LÍNEA. .....19*
3.3.1.1 Unipolar sin retorno a cero (NRZ). .....19
3.3.1.2 Unipolar con retorno a cero (RZ). .....19
3.3.1.3 Polar NRZ. .....20
3.3.1.4 Polar RZ. .....20
3.3.1.5 Bipolar NRZ. .....21
3.3.1.6 Bipolar RZ. .....21
3.3.1.7 Manchester o señalización bifásica. .....22
3.3.1.8 Señalización M-aria multinivel. .....22
*3.3.2 ESQUEMA COMPARATIVO DE LOS DIFERENTES CÓDIGOS DE LÍNEA. .....24*
3.4 MODULADOR DIGITAL. .....25
*3.4.1 MODULACIÓN DIGITAL EN BANDA BASE (BB). .....25*
3.4.1.1 Modulación por Amplitud de Pulsos (PAM). .....26
3.4.1.2 Modulación por Posición de Pulsos (PPM). .....26
3.4.1.3 Señales Biortogonales. .....27
3.4.1.4 Señales Simplex. .....27
*3.4.2 MODULACIÓN DIGITAL EN SISTEMAS PASO DE BANDA (BP). .....27*
3.4.2.1 Modulación ASK. .....28
3.4.2.2 Modulación PSK. .....28
3.4.2.3 Modulación QAM. .....29
3.4.2.4 Modulación FSK. .....30
3.5 CANAL. .....31
*3.5.1 CLASIFICACIÓN DE CANALES SEGÚN EL TIPO DE MEDIO DE TRANSMISIÓN UTILIZADO. .....31*
3.5.1.1 Canales de líneas de cable. .....31
3.5.1.2 Canales de fibra óptica. .....31
3.5.1.3 Canales vía radio (Canales electromagnéticos). .....32
3.5.1.4 Canales submarinos. .....32
*3.5.2 MODELOS MATEMÁTICOS PARA CANALES DE COMUNICACIONES. .....32*
3.5.2.1 Canal de ruido aditivo. .....32
3.5.2.2 Canal de filtro lineal. .....33

3.5.2.3 Canal de filtro lineal variante con el tiempo. ... 33
*3.5.3 CANALES MÁS UTILIZADOS EN LA SIMULACIÓN DE SISTEMAS.* ... *34*
3.5.3.1 Canal AWGN (Additive White Gaussian Noise). ... 34
3.5.3.2 Canal "Rayleigh fading". ... 35
3.5.3.3 Canal "Rician fading". ... 36
3.6 DEMODULADOR. ... 37
*3.6.1 DEMODULACIÓN.* ... *37*
3.6.1.1 Demodulador de correlación. ... 37
3.6.1.2 Demodulador de filtros adaptados. ... 38
*3.6.2 DETECCIÓN.* ... *39*
3.7 DECODIFICADOR DE CANAL. ... 40
*3.7.1 DECODIFICACIÓN DE MÍNIMA DISTANCIA.* ... *40*
*3.7.2 DECODIFICACIÓN DEL SÍNDROME.* ... *41*
*3.7.3 DECODIFICACIÓN DE VITERBI: DECODIFICACIÓN DE CÓDIGOS CONVOLUCIONALES.* ... *41*

**4 ESTÁNDAR DVBT. ... 42**

4.1 INTRODUCCIÓN. ... 42
4.2 OFDM: PRINCIPIOS DE FUNCIONAMIENTO. ... 43
4.3 COFDM. ... 44
4.4 RESUMEN. ... 45

**5 PRUEBAS DE MODULACIÓN DIGITAL. ... 46**

5.1 PRUEBAS Y SIMULACIONES. ... 46
*5.1.1 PRUEBA TEST_0.* ... *46*

**6 CODIFICADOR CONVOLUCIONAL Y DECODIFICADOR DE VITERBI. ... 49**

6.1 INTRODUCCIÓN TEÓRICA. ... 49
*6.1.1 CODIFICADOR CONVOLUCIONAL.* ... *49*
*6.1.2 EL ALGORITMO DE VITERBI.* ... *51*
6.1.2.1 Introducción a los códigos FEC. ... 51
6.1.2.2 Algoritmo de Viterbi. ... 53
*6.1.3 CASO PRÁCTICO.* ... *55*
6.2 PRUEBAS Y SIMULACIONES. ... 59
*6.2.1 PRUEBAS DE VERIFICACIÓN DE CÓDIGO.* ... *59*
6.2.1.1 Prueba TEST_1. ... 59
6.2.1.2 Prueba TEST_2. ... 61
*6.2.2 PRUEBAS DE SIMULACIÓN DE SISTEMA.* ... *63*
6.2.2.1 Prueba TEST_3. ... 64
6.2.2.2 Prueba TEST_4. ... 72

**7 CONCLUSIONES / RESULTADOS. ... 80**

**8 MATLAB. ... 81**

8.1 INTRODUCCIÓN. ... 81
8.2 SISTEMA DE COMUNICACIONES SIMULADO. ... 81
*8.2.1 FUNCIONES.* ... *82*
8.2.1.1 Codificador convolucional: conv_module. ... 82
8.2.1.2 Punctureado de los datos: punc_dvbt. ... 84
8.2.1.3 Mapeo de datos: mapper_dvbt. ... 86
8.2.1.4 Canal: channel. ... 89
8.2.1.5 Saturador: Saturador. ... 90
8.2.1.6 Demapeo de datos: demapper_dvbt. ... 91
8.2.1.7 Depunctureado: depunc_dvbt. ... 96
8.2.1.8 Decodificador: vit_module. ... 99

**9 ANEXOS. ... 101**

9.1 TIPOS DE MÉTRICAS. ... 101

**10 BIBLIOGRAFÍA. ... 105**

# 1 ACRÓNIMOS.

- APK: QAM.
- ARQ : *Automatic Repeat Request* (Códigos de petición de retransmisión automática).
- ASK: *Amplitude Shift Keying* (Modulación por Conmutación de Amplitud).
- AWGN: *Additive White Gaussian Noise* (Ruido aditivo Blanco Gaussiano).
- BB: *Base Band* (Banda Base).
- BER: *Bit error rate* (Tasa de error de bit).
- BP: *Band Pass* (Paso de banda).
- BM: *Branch-metric.*
- BW: *Bandwidth* (Ancho de Banda).
- COFDM: *Coded Orthogoanl Frecuency Division Multiplexing* (Multiplexión por división en frecuencia ortogonalmente, con codificación).
- CSI: *Channel State Information* ( Información de Estado del Canal).
- DMS.: *Discrete Memoryless Source* (Fuente discreta sin memoria).
- DVB-T: *Digital Video Broadcasting- Television*.
- FECC: *Forward Error Correction Coding* (Códigos de corrección de errores en recepción).
- F.F.T.: *Fast Fourier Transform* (Transformada Rápida de Fourier).
- FSK: *Frecuency Shift Keying* (Modulación por Conmutación en Frecuencia).
- HDTV: *High Definition TV* (TV de alta definición).
- IEEE: *Institute of Electrical and Electronics Engineers.*
- IG: Intervalo de Guarda.
- i.i.d.: variables aleatorias idénticamente distribuidas.
- LOS: Line of Sigth (Camino dominante).
- QAM: Quadrature-Amplitude Modulation (Modulación en fase y cuadratura).
- MLSE: *Maximun Likelihood Sequence Estimator* (Estimador de Secuencia de Máxima Verosimilitud).
- NB_SD: Número de bits de entrada en “soft-decision”
- N.R.Z.: No retorno a cero.
- OFDM: *Orthogoanl Frecuency Division Multiplexing* (Multiplexión por división en frecuencia ortogonalmente).
- PAM : *Pulse Amplitude Modulation* (Modulación por amplitud de pulsos).
- PCM.: *Pulse Codification Modulation* (Modulación por codificación de pulsos).
- PDF: Probability Density Function (Función Densidad de Probabilidad).
- PM: *Path-metric.*
- PPM: *Pulse Position Modulation* (Modulación por posición de pulsos).

- PSD.: *Power Spectrum Density* (Espectro de densidad de potencia).
- PSK: *Phase Shift Keying* (Modulación por Conmutación de Fase).
- R.Z.: Con retorno a cero.
- SNR: *Signal-Noise rate* (Relación señal a ruido).
- SP: *Survivor Path*
- TV: Televisión.
- TB*: Trace-back.*
- TBL: *Trace-back-length.*

# 2 BREVE DESCRIPCIÓN.

Hoy en día, las telecomunicaciones forman parte esencial en la vida diaria, por tanto su investigación y desarrollo colaboran en proporcionar calidad de vida y confort para todos.

Pero, las telecomunicaciones, no son sólo las nuevas tecnologías, como telefonía móvil o Internet; comprenden también otros servicios mucho más arraigados en la vida diaria, como pueden ser la radio, el fax o la televisión, por citar alguno. Y, en todos estos servicios de telecomunicación el avance también es primordial.

En el caso de este estudio, el proporcionar una mejor calidad en TV minimizando costes, es principal en la sociedad actual. Porque lo importante de un cambio en la transmisión de TV, de analógica a digital, no sería importante si no supusiera mejoras a todos los niveles, tanto para el usuario, como para las operadoras que ofrecen el servicio. Este avance supone además servicios que hasta ahora no eran viables, como se ven en un capítulo dedicado a ello.

Para afrontar el estudio de un sistema de telecomunicaciones para TV digital, es importante saber cómo es un sistema de telecomunicaciones general, sus bloques funcionales, y las tareas principales de cada uno de ellos. Y, a partir de él poder extrapolar al sistema en estudio. A este fin se dedica el Capítulo 3.

Una vez hecha la descripción de un sistema de telecomunicaciones general, en el Capítulo 4 nos centramos en el sistema objeto de posterior estudio, un estándar para DVB-T (TV Digital), basado en sistemas COFDM (Coded Orthogonal Frecuency Division Multiplexing). Se introducen los conceptos más relevantes de este tipo de sistemas así como sus prestaciones y el por qué utilizarlo en el desarrollo del estándar DVB-T. Se desarrolla la importancia dentro de este sistema de la codificación convolucional, y la decodificación de Viterbi cómo método para la codificación de canal.

En el Capítulo 5 se hace una comprobación de código, dessde el punto de vista de modulación y canal., puesto que existen resultados teóricos que se deben verificar, antes de proseguir el estudio. Uno de los resultados de estas simulaciones está orientado a la comprensión del papel de las distintas modulaciones en la BER.

El Capítulo 6 es el principal en este estudio. Primero se explica de forma teórica y exhaustiva lo que es un sistema basado en codificación convolucional, y decodificación de Viterbi, codificación de canal de tipo FEC. Además se desarrolla un ejemplo sencillo que pone de manifiesto la capacidad correctora del 'Algoritmo de Viterbi' a la vez que enseña de forma práctica su realización. Una vez terminada la parte teórica, se pasa a las simulaciones. Las primeras de verificación de código nuevamente, y un segundo tipo, de sistema, que producen los resultados a discutir para poder optimizar el funcionamiento del sistema con un decodificador de Viterbi.

Una vez terminadas las simulaciones, se pasa a la puesta en común de los resultados en el Capítulo 7, para poder sacar conclusiones sobre los parámetros que necesita el Viterbi para un funcionamiento óptimo, según la modulación utilizada y la tasa de "puncturing", teniendo en cuenta su posterior implementación hardware, que es el motivo de estas simulaciones.

Para finalizar, en el Capítulo 8, se estudia el lenguaje de programación utilizado. Además se adjunta el código, puesto que existen líneas de investigación abiertas y puede resultar útil en trabajos posteriores como línea de consulta.

# 3 MEMORIA DESCRIPTIVA: DESCRIPCIÓN DE UN SISTEMA DE TELECOMUNICACIONES.

Un sistema de comunicaciones se puede ver de modo muy simplificado de la siguiente forma:

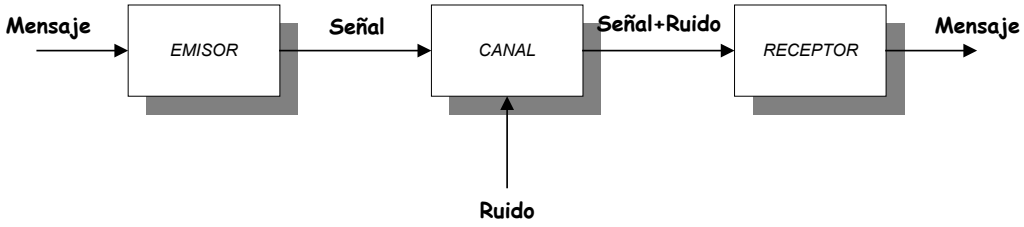



Figura 1: Sistema de comunicaciones simplificado.

Cada uno de estos bloques se subdivide en otros, que se encargan de transformar el mensaje original en información que pueda ser enviada por el canal, simular el canal de comunicaciones y transformar la señal que nos llega a través de este canal en el mensaje original. Recuperando así la información que se envió.

El bloque emisor se subdivide en:

- Codificador de fuente: Elimina las redundancias de la fuente emisora.
- Codificador de canal: Se introduce redundancia para el control de errores.
- Codificador de línea: Asigna la forma de onda en banda base.
- Modulador: Nos lleva la señal a la banda de frecuencia elegida.

En este apartado vamos a hacer una breve introducción de la funcionalidad de los distintos bloques, para pasar en apartados posteriores a describirlos de forma mas completa.

La información de la fuente puede venir de forma analógica o digital, si es digital, directamente pasaría una secuencia de bits al codificador de fuente. Si, por el contrario, es analógica, primero tenemos que digitalizar esta información, usando un convertidor analógico/digital, que se encarga de la cuantización y codificación de la señal en una secuencia de bits.

El siguiente paso es la codificación de fuente, donde la salida será un conjunto de símbolos de un alfabeto binario y finito. Este bloque selecciona este conjunto de valores finitos según los valores de entrada. Su principal objetivo es la transmisión de información sin redundancia, suprimiendo los símbolos que no transmiten información, compactando de datos. La salida, una secuencia de k dígitos binarios representa por tanto toda la información de la fuente. El concepto de "información" manejado por el codificador de fuente está íntimamente relacionado con la probabilidad de ocurrencia de los símbolos. Lo estudiaremos más adelante, cuando veamos este bloque de modo independiente.

El codificador de canal se encarga de añadir información o datos redundantes que nos van a permitir el control de errores en el canal y disminuir la probabilidad de error de un símbolo. Su principal desventaja es su costo, que nos viene impuesto por un aumento del ancho de banda necesario en transmisión.

El codificador de línea transforma la señal de salida del codificador de canal en una forma de onda susceptible de ser enviada por el canal en banda base. Es decir, aplica una determinada tensión según el bit a transmitir. Sólo es utilizado en sistemas en banda base.

El bloque modulador digital se encarga de trasladar la señal modulada a la banda de paso del canal, para esto utiliza otra señal, a la que va a llamar moduladora. Esta señal va a llevar inmersa en sí la señal que ha salido del codificador de línea. Una señal no siempre se modula del mismo modo, pero esto se verá más adelante cuando tratemos con detenimiento este bloque.

El siguiente bloque funcional es el canal, uno de los más difíciles de modelar porque un canal no siempre se comporta de la misma forma ni le afectan los mismos factores, en él influyen muchos parámetros externos y aleatorios. Estos factores aleatorios son los más difíciles de controlar, porque afectan de modo diferente en cada instante. En un apartado posterior veremos una serie de modelos de canales, para la simulación matemática, así como los distintos medios de transmisión, como por ejemplo fibra óptica, cable de pares, transmisión vía radio,... Cada uno de ellos se comporta de forma diferente ante las distintas interferencias que influyen en la transmisión en el canal, y además, estas interferencias, no afectan de la misma manera a cada uno de ellos. De cada uno de ellos obtendremos una serie de conclusiones.

El bloque receptor tiene un comportamiento inverso al emisor, debe recuperar la información después de que esta haya pasado por el canal y haya sido afectada por distintos factores aleatorios, y entregarla al destino sin errores. Cada uno de los bloques que nos encontramos, realiza la función inversa al que se encuentra en la misma posición del transmisor.

Vemos un esquema del sistema completo:

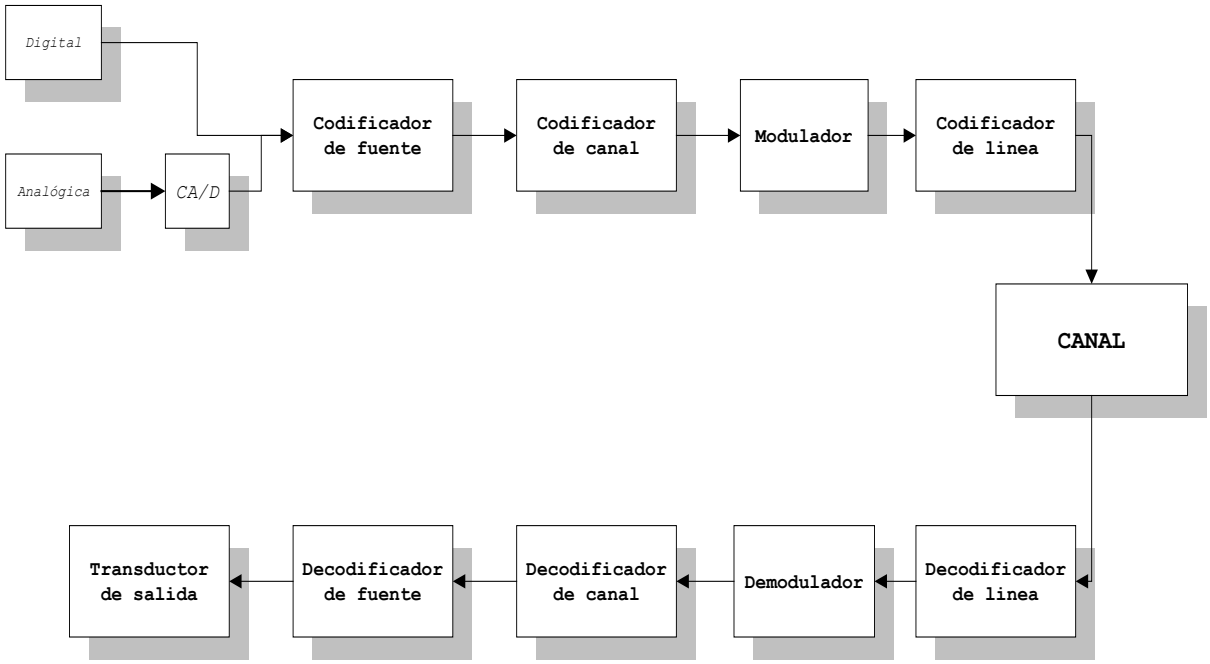



Figura 2: Esquema completo de un sistema de telecomunicaciones.

## *3.1 CODIFICADOR DE FUENTE.*

En este bloque se aplica la "Teoría de la Información", debida a Shannon, en 1948. Es a partir de ese momento cuando empiezan a cambiar los conceptos de comunicación digital.

El campo de aplicación de la Teoría de la Información es la representación de la información en símbolos, dando resultados sobre la realización teórica en sistemas de comunicaciones, y permitiendo valorar resultados sobre la eficiencia de un sistema real.

Una vez que tenemos la información de la fuente, queremos comprimirla, darle un determinado formato, etc. De esto es de lo que se encarga nuestro "Codificador de Fuente". Pero, en primer lugar, necesitaremos un modelo matemático o medida cuantitativa para ver qué es la información. De esto se encargaron Hartley, Nyquist y Shannon, mediante la mencionada Teoría de la Información.

### 3.1.1 MODELADO DE FUENTES DE INFORMACIÓN.

Lo primero va a ser definir el concepto de información tal y como va a ser utilizado. Por tanto, la Información es todo conocimiento nuevo sobre algo. Aplicado a un sistema de comunicaciones, será aquello que es necesario transmitir al receptor, aquello que aporta conocimiento. No todo lo que sale de la fuente de mensajes es susceptible de ser enviado, contiene muchos datos redundantes que no son necesarios en el receptor, se pueden omitir sin que se altere el mensaje, y con esto se gana eficiencia. El diseñador lo que intenta es que el mensaje llegue completo y con las mínimas perturbaciones al receptor.

Las fuentes de información se van a modelar como procesos aleatorios, porque a priori no se conoce el contenido de los datos de una fuente, si estos datos fueran conocidos, no aportarían información. Todo aquello que no se conoce a priori, se modela como proceso aleatorio.

#### 3.1.1.1 Características estadísticas.

Van a depender de las características de la fuente, cada fuente tiene características diferentes, no es lo mismo modelar imagen, voz o datos. Además, según el estándar que se emplee para cada una de ellas, también cambia la fuente.

Para imagen, la señal de vídeo, es una sucesión de imágenes fija y su BW (ancho de banda) va a depender de la resolución. Esta secuencia, es una secuencia con mucha redundancia, una imagen apenas cambia respecto a la anterior o a la siguiente, y debe ser eliminada para no desaprovechar el ancho de banda del canal. Además también tenemos que tener en cuenta, que según el estándar de TV utilizado, la frecuencia de muestreo, y como consecuencia el ancho de banda también cambian.

Lo mismo ocurre cuando tenemos fuentes de voz o datos. De este modo ponemos de manifiesto que según el tipo de fuente, la codificación debe hacerse de modo diferente, esto es, no tenemos un mismo codificador de fuente que sirva en todos los casos.

Lo único que tienen en común todas las señales es que son procesos aleatorios limitados en banda, por eso se van a poder muestrear a la tasa de Nyquist, o mayor. Al muestrear, un proceso continuo obtenemos señales discretas, por tanto nosotros vamos a considerar nuestra fuente de información como una fuente discreta.

### 3.1.1.2 Modelado matemático de la fuente.

Se representa según el siguiente esquema:

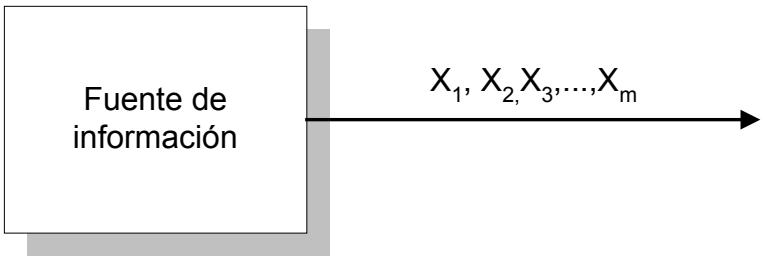



Figura 3: Modelado de la fuente de información.

El conjunto $\{Xi\}$ que viene de la fuente puede tomar valores de un alfabeto tanto binario A= {$a_1$, $a_2$}, como M-ario A= {$a_1$, $a_2$, $a_3$,... , $a_m$}. Cada uno de los símbolos de los alfabetos, está compuesto a su vez por símbolos binarios.

Ilustramos esto con varios ejemplos:

1. Tenemos un alfabeto binario, de símbolos binarios, esto es: A= { 0, 1 }.
2. Tenemos un alfabeto 4-ario (cuaternario), de símbolos binarios, esto es:
   A= { 00, 01, 10, 11 }.
3. Tenemos un alfabeto 8-ario, de símbolos binarios:
   A= { 000, 001, 010, 011, 100, 101,110,111 }
4. Tenemos un alfabeto 9-ario, con símbolos 3-arios:
   A= { 00, 01, 02, 10, 11, 12, 20, 21, 22 }

Los más usuales, son los alfabetos M-arios de símbolos binarios, en este tipo de alfabetos se cumple la relación:

$$\text{nº de bits/símbolo} = \log_2 M$$

Esta es una relación muy importante, porque nos da la información sobre el número de posibles valores de los símbolos de entrada al codificador de fuente.

### 3.1.1.3 Tipos de modelos de fuentes.

Presentamos aquí varios tipos, son los más usuales y nos van a servir para ilustrar este apartado a modo de ejemplo.

#### *3.1.1.3.1 Fuente discreta sin memoria. (DMS Discrete Memoryless Source)*

Genera un proceso aleatorio X(t), cuyos valores $X_i$ son variables aleatorias estadísticamente independientes con igual distribución de probabilidad. Además este proceso aleatorio X(t) es discreto en tiempo y amplitud.

Este conjunto toma sus valores del alfabeto M-ario A= $\{ a_1, a_2, a_3, ... , a_m \}$

Donde la asociación de probabilidad viene dada por:

$$p_i = p\ (X = a_i)\ ;\ \text{donde } i = 1, 2, ..., m.$$

Con A y $p_i$, la fuente queda completamente identificada.

#### *3.1.1.3.2 Fuente con memoria.*

En este caso las variables estadísticas $X_i$, que forman el proceso aleatorio X(t) son estadísticamente dependientes. Esto es, cada una de ellas tiene dependencia con la anterior.

Se suponen fuentes estacionarias y discretas. Una fuente estacionaria es aquella en la cual la probabilidad conjunta de dos secuencias de longitud n no depende del origen de tiempo que cojamos.

Al igual que en el caso anterior, esta fuente queda perfectamente determinada por el alfabeto utilizado, y por la probabilidad de ocurrencia de cada uno de sus símbolos, $p_i$, aunque, para poder determinar esta probabilidad hay que acudir a la función de densidad de probabilidad conjunta.

#### *3.1.1.3.3 Fuente analógica.*

Suponemos que tenemos una fuente que me da una salida continua en el tiempo X(t), compuesta por m funciones muestra continuas.

Esto es: $X(t) = \{x_1(t), x_2(t), x_3(t), ..., x_m(t)\}$,. Esta salida se muestreará cuando tengamos X(t) limitada en banda, obtendremos un conjunto de valores $\{X(nT_s)\}$ que denominaremos "Fuente Equivalente Discreta".

Las amplitudes de $X_i(nT_s)$ van a ser amplitudes continuas, esto es no están definidas dentro de un conjunto finito de valores, no van a estar dentro de un alfabeto. Hay que realizar una cuantización de los valores.

Este tipo de fuente necesita un CA/D antes de entrar en el codificador de fuente. El CA/D introduce un error que después no vamos a ser capaces de recuperar, es el llamado error de cuantización.

Vemos esto con un esquema:

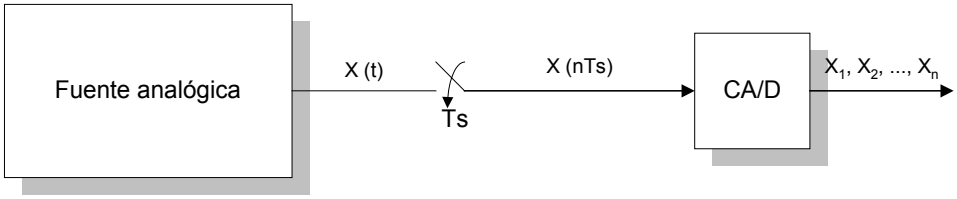



Figura 4: Fuente analógica y entrada en el codificador de fuente.

### 3.1.2 MEDIDA DE LA INFORMACIÓN.

Vamos a dar una medida cuantitativa de la información, esto es, un modelo matemático.

Partimos de un alfabeto: A= $\{a_1, a_2, a_3, ... , a_N\}$ donde cada símbolo lleva asociada una probabilidad $p_i$ y donde se cumple que $a_1$ es el símbolo más probable, y $a_N$ el menos. Por tanto, observamos que un alfabeto se ordena según la probabilidad en orden decreciente.

De aquí, es fácil deducir que la información de salida de una fuente tendrá que ser una función decreciente con la probabilidad asociada a esa salida concreta. Y, por otro lado, si la probabilidad variara un poco, queremos que esto no nos provoque un cambio grande en el contenido de la información, por tanto, necesitamos que sea una función continua, sin cambios bruscos, que nos puedan provocar saltos. Mientras menos probable es un hecho más información nos aporta

#### 3.1.2.1 Entropía, indeterminación o incertidumbre de la fuente.

Necesitamos hacer una medida de la cantidad de información que tiene un estado de la función de información cuando se ha eliminado la incertidumbre. Esto es, hacer una medida de la incertidumbre o indeterminación de la elección del estado de una función de información.

Vamos a suponer la salida de una fuente discreta, para dar generalidad, puesto que, aunque la fuente sea analógica, hemos visto la forma de digitalizar esos valores y obtener una cadena de datos digital.

Por otro lado vamos a sentar las condiciones que debe cumplir el contenido de información de la salida de una fuente, aunque algunas ya se han visto en apartados anteriores. La solución que buscamos es una función con probabilidad $p_i$ . Vemos estas condiciones:

- La información va a depender de la probabilidad, y no del símbolo de salida $a_i$.
- La información va a ser una función decreciente con la probabilidad.
- La información debe ser una función continua con la probabilidad.
- Debe cumplirse: $p_i = p^{i1} \cdot p^{i2} \Rightarrow I_i = I^{i1} + I^{i2}$ donde $I_i$ es la información asociada a la probabilidad $p_i$.

La única función que cumple estas condiciones es la función logaritmo, a la que denominamos “Función de Información Propia”. Su formulación matemática viene dada por:

$$I(p_i) = - \log (p_i)$$

Una vez definida la función de información, pasamos a ver qué es la entropía, y que aporta a este estudio.

La entropía es una medida de incertidumbre, nos da la tasa mínima de la fuente en número de bits por muestra. A partir de un alfabeto dado, y de la probabilidad de los símbolos del alfabeto. Su formulación matemática viene dada por:

$$H(X) = -\sum_{i=1}^{N} pi.\log(pi) = \sum pi.\log\left(\frac{1}{pi}\right)$$

Para el caso particular de símbolos equiprobables, esta fórmula se reduce a :

$$H(X) = \log N$$

Esta es una cota superior de la entropía, esto es, para símbolos equiprobables, la entropía de una fuente es máxima. Necesita un número máximo de bits por cada símbolo.

### 3.1.2.2 Entropía conjunta y condicional.

La entropía conjunta es una medida de la incertidumbre cuando existen relaciones entre dos o más fuentes de un mismo sistema, o entre los estados de una misma fuente. Nos da la influencia mutua de las fuentes, o de sus estados. Viene definida por:

$$H(X_1, X_2, ..., X_n) = -\sum_{i1=1}^{N1}\sum_{i2=1}^{N2}\cdots\sum_{in=1}^{Nn} p(x_1, x_2, ..., x_n) \cdot \log\left(p(x_1, x_2, ..., x_n)\right)$$

En general, se tienen fuentes estadísticamente independientes con lo cual, el cálculo de los estadísticos que aparecen en la fórmula se simplifica notablemente.

Por otro lado, la entropía condicional de una fuente respecto a otra, mide la incertidumbre de una fuente teniendo la certeza del resultado de otra. Se calcula a partir de la incertidumbre de una fuente respecto a un estado concreto de otra. Vemos su formulación matemática.

Entropía particular de X con el estado $y_j$ de Y:

$$H(X/Y = y_j) = -\sum_{i=1}^{N} p(x_i/y_j) \cdot \log\left(p(x_i/y_j)\right)$$

Entropía condicional de X con Y:

$$H(X/Y) = -\sum_{j=1}^{M} p(y_j) \cdot \log\left(H(X/Y = y_j)\right) = -\sum_{i=1}^{N}\sum_{j=1}^{M} p(x_i, y_j) \cdot \log\left(p(x_i/y_j)\right)$$

### 3.1.3 TEOREMA DE LA CODIFICACIÓN DE FUENTE.

Este teorema intenta dar un límite en la tasa a la que se pueden comprimir las salidas de una fuente, manteniendo la distorsión mínima.

Se enuncia de la siguiente forma, "*Una fuente con una entropía H(X) puede ser codificada con una tasa de bits R (bits/salida de fuente) con una probabilidad de error arbitrariamente pequeña, siempre que cumplamos que R>H. Si por el contrario R<H, entonces la probabilidad de error está acotada lejos del cero, independientemente del codificador*".

El método se divide en dos, según tengamos fuentes con o sin memoria. Vemos cada uno de los casos por separado.

#### 3.1.3.1 Fuente discreta con memoria (DMS).

La fuente emite secuencias de n símbolos de un alfabeto de N letras, por tanto, el número total de secuencias posibles viene dado por $N^n$.

Este teorema, divide este número de secuencias según la probabilidad de ocurrencia, a las que tienen mayor ocurrencia se las denomina secuencias típicas, y vienen dadas por $2^{n\,H(X)}$.

Por tanto, aunque exista un número total de secuencias igual a $N^n$, el número efectivo viene dado por $2^{n\,H(X)}$. Con $2^{n\,H(X)} < N^n$. Aquí se pone de manifiesto la existencia de una tasa de compresión de datos, o sea la eliminación de redundancias.

Por último, comentar que si la fuente tiene una función de densidad de probabilidad uniforme entonces la compresión es nula. El número de secuencias típicas coincide con el total.

#### 3.1.3.2 Fuentes con memoria.

Para fuentes con memoria, el cálculo de secuencias típicas se basa en la Tasa de entropía.

Tasa de entropía:

$$H = \lim_{n->\infty} H\left( X_n / X_1, X_2, ..., X_{n-1}\right)$$

El número de secuencias típicas viene dado por $2^{n\,H}$.

## 3.2 CODIFICACIÓN DE CANAL.

En un sistema de comunicaciones, los recursos a minimizar son la potencia emitida, y el ancho de banda (BW) de la señal en el canal. Por tanto, el objetivo principal va a ser ofrecer una tasa máxima de transmisión de datos, en un BW mínimo, manteniendo una calidad aceptable.

La codificación de canal se utiliza para detectar, y en algunos casos corregir los datos recibidos con error, se encarga por tanto de hacer la transmisión más fiable. Para ello introduce una serie de bits de redundancia en la secuencia a transmitir.

Existen códigos detectores de errores, y otros que son detectores y correctores, se aplican uno u otro según el canal específico con el que nos encontramos, esto se ve más adelante.

La detección del error puede ser utilizada como un primer paso en la técnica de corrección.

Existen dos técnicas principales de corrección, estas son ARQ (Automatic Repeat Request) y FECC (Forward Error Correction Coding). En la primera, ARQ, se pide la retransmisión de los datos una vez se ha detectado el error. En FECC, tenemos códigos capaces de detectar y corregir el error por sí solos. En el siguiente esquema de la vemos una clasificación de los codificadores de canal.

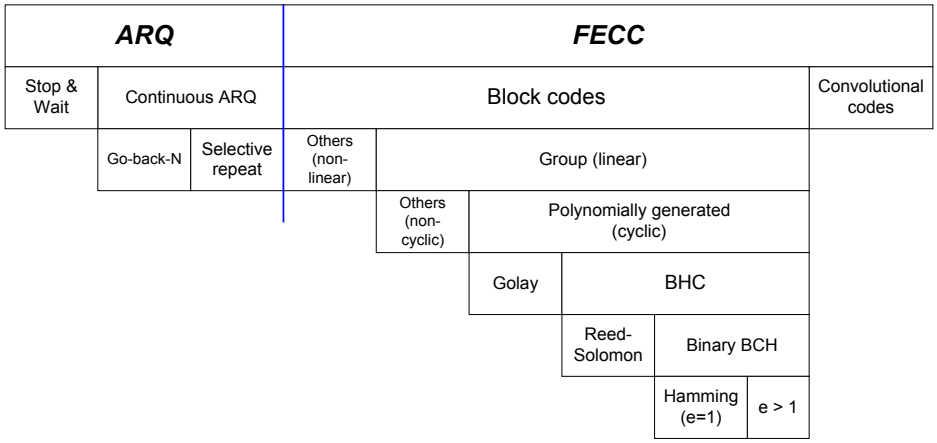



Figura 5: Clasificación de códigos de canal.

En general, el utilizar ARQ o FECC, depende de distintos factores, bien sea el canal, los aspectos software o hardware del sistema, o ambos. Cada uno tiene sus ventajas e inconvenientes, los ARQ tienen algoritmos de codificación y decodificación más simples, necesitan menos bits de redundancia y además son adaptativos, esto es, retransmiten sólo cuando existe error. En contrapartida, necesitan un canal duplex. Por otro lado, los FECC no necesitan canal de retorno. Si la probabilidad de error es alta no compensa la retransmisión, sino directamente la corrección.

Hecha una introducción general, vemos con más detenimiento cada uno de ellos.

### 3.2.1 ARQ. CÓDIGOS DE PETICIÓN DE RETRANSMISIÓN AUTOMÁTICA.

Como ya se ha dicho, son códigos que necesitan un canal duplex para transmisión, ya que necesitan el canal de retorno para el asentimiento de datos.

Utiliza un código detector de errores, y si existe un error, directamente se pide la retransmisión. En función del modo de retransmisión, tenemos las distintas técnicas de ARQ.

El primer caso, lo denominamos *"parada y espera"*, en ella el transmisor espera un asentimiento positivo que le da permiso para transmitir el siguiente bloque de datos. En el caso de asentimiento negativo, se produce la retransmisión, tantas veces como sea necesario.

En un segundo caso, denominado *"ARQ continuo"*, los paquetes de datos son transmitidos de forma continua, ordenados en paquetes cada uno con un número de secuencia, sin necesitar un asentimiento previo. Como se puede observar en la Figura 5, esta técnica se subdivide en otras dos variantes, en función del modo en que se retransmite en caso de error. La primera, denominada *"Go-back-N"*, cuando llega un asentimiento positivo, asiente todos las paquetes hasta su número de secuencia. Sin embargo, si el asentimiento es negativo, retransmite todas las tramas desde la última con asentimiento positivo, hasta el número de secuencia que traía el negativo. En la segunda variante, llamada "Repetición selectiva", sólo se retransmiten aquellas tramas que llegan con asentimiento negativo.

### 3.2.2 FECC. CÓDIGOS CORRECTORES EN RECEPCIÓN.

Estos códigos, en contrapartida a la detección y corrección de errores, al introducir más bits de redundancia que los anteriores, bajan el rendimiento, a la vez que suben el ancho de banda necesario en transmisión. Suponen también un aumento de la complejidad del hardware e introducen un cierto retraso.

Vemos un esquema:

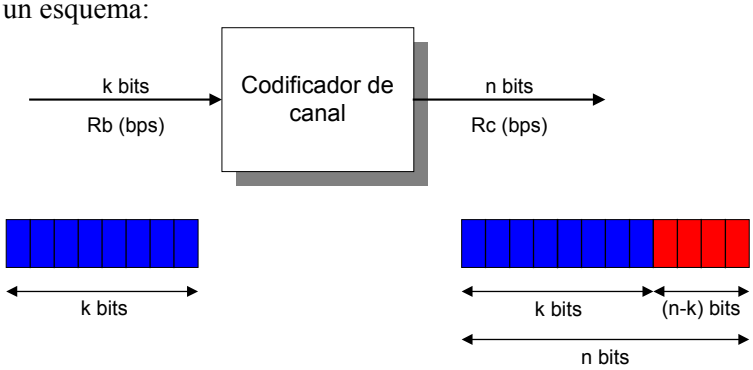



Figura 6: Codificador de canal FECC.

Pero, estas desventajas se ven compensadas por los beneficios que proporciona el control de errores sin necesidad de retransmisión.

Se van a dividir en dos grandes grupos, *"códigos de bloque"*, y *"códigos convolucionales"*. Los primeros se tratan en este apartado de forma más o menos extensa, pero los segundos, al ser una parte importante de estudio de este proyecto ocupan un apartado posterior por sí solos.

#### 3.2.2.1 Códigos de bloque.

Se toman palabras o bloques de información de k bits a los que se le añaden (n-k) bits de redundancia, formando una palabra de código de n bits. Se van a denotar como *"códigos (n,k)"*.

Existe otra división totalmente independiente, y es que pueden ser sistemáticos, en los que los bits de información se incluyen explícitamente en la palabra de código, y los no sistemáticos, en caso contrario. Aquí vamos a tratar los sistemáticos por ser los más utilizados.

##### *3.2.2.1.1 Códigos de bloque lineales.*

Transforman un bloque de k bits en otro de n bits realizando operaciones matemáticas. Por tanto, vamos a tener una matriz llamada *"generadora de código"* de dimensión (k x n).

Estos códigos tienen la propiedad de que una palabra de código se puede obtener como la suma de otras dos que pertenezcan al código, o dicho de otra manera, la suma de dos palabras de código cualesquiera, siempre da lugar a otra palabra de código.

Dentro de estos códigos lineales de bloque existen otros dos subconjuntos, los códigos generados mediante polinomios, o códigos cíclicos y los que no. Estos últimos no revisten mayor interés por lo que no se van a detallar.

###### *3.2.2.1.1.1 Códigos cíclicos.*

Tienen una estructura algebraica muy potente, lo que nos lleva a una serie de propiedades y ventajas que otros códigos no poseen. Estas características se detallan a continuación:

- Se pueden generar a partir de registros de desplazamiento.
- El vector del síndrome se calcula a partir de estos registros.
- Poseen una estructura algebraica que permite una decodificación potente.

Son códigos largos con una implementación sencilla. Para comprobar si un código es cíclico hay que ver si cumple la siguiente propiedad: "*Cualquier desplazamiento cíclico de una palabra de código, es otra palabra de código*".

Dentro de los códigos cíclicos, siguen apareciendo subgrupos, que no se van a tratar por ser particularizaciones de lo ya expuesto, cosa que se escapa del objeto de estudio.

#### 3.2.2.2 Códigos convolucionales.

Los códigos convolucionales, como su nombre indica, se generan a partir de un codificador convolucional. En este tipo de generador de código la secuencia de información va pasando a través de un registro de estados. Podemos definirlo como 'k' registros de estados, tantos como entradas tengamos, y 'n' generadores de funciones, tantos como salidas. Por tanto la tasa de transmisión será de 'k/n'. Sólo queda por comentar que cada uno de los registros, tendrá L estados. A esta variable L se le va a llamar *"constraint-length"*. En la Figura 7 de la siguiente página, vemos de forma gráfica esta definición.

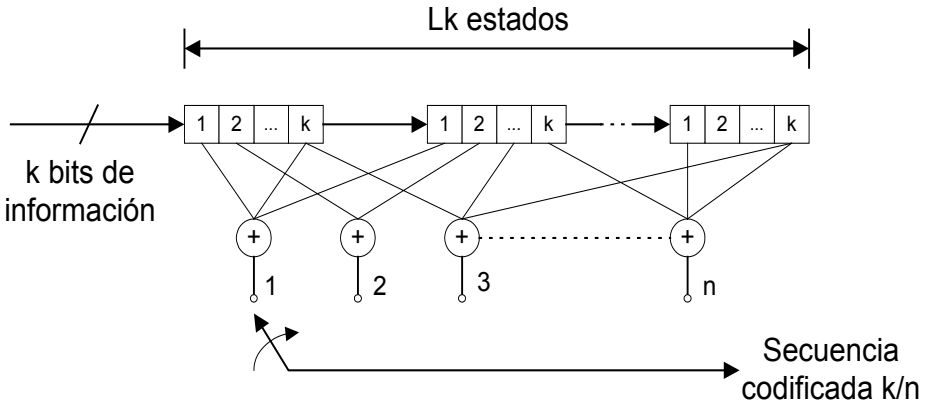



Figura 7: Codificador convolucional.

Vemos una explicación mas exhaustiva en el Capítulo 6, dedicado especialmente a codificación convolucional y su método MLSE de decodificación.

## *3.3 CODIFICADOR DE LINEA.*

Tras el codificador de canal, se ha obtenido la palabra de código, ahora necesitamos transformar esta palabra de código en forma de onda, para poder ser transmitida por el canal. Hacer notar que el codificador de linea sólo se usa en sistemas en banda base.

Y, el bloque encargado de dar a la palabra de código una forma susceptible de ser enviada por el canal es el 'Codificador de línea'. La forma de onda adoptada es la información que se va a pasar al siguiente bloque, el modulador. Esta señal va a ser una señal analógica, esto es una señal que transcurre en el tiempo susceptible de ser enviada por el canal.

La señal de salida de este bloque va a ser una secuencia de pulsos definida de acuerdo a una regla bien estipulada. A este proceso se le llama "*señalización digita*l".

Para la codificación, el codificador de línea supone:

- Ancho de banda del canal infinito.
- Pulsos rectangulares perfectamente definidos.
- La codificación se hace en banda base.

Necesitamos un código robusto, esto influye en el codificador de línea, para que esto sea posible debe cumplir ciertas propiedades:

1. Autosincronización. Debemos ser capaces de extraer el reloj a partir de la señal que llega al receptor, sin necesitar un canal auxiliar de señalización para el reloj. Esto es, en la forma de onda enviada, debe ir incluido el reloj de transmisión.

2. Robustez frente a errores. Necesitamos un código que a la hora de decodificar en el receptor no nos introduzca errores. Esto es fácil de decodificar sin errores, teniendo en cuenta el ruido que introduce el canal de comunicaciones.

3. Transparencia. Puede ser decodificado por sí mismo, no necesita información adicional, no depende de la secuencia que se está transmitiendo. Independientemente de los valores transmitidos no cambia. Mas adelante, cuando se citen los diferentes códigos de línea, se verá esta característica de forma más clara.

4. Necesitamos un espectro de densidad de potencia (PSD) que se adapte lo mejor posible al canal. Esto es, nosotros tenemos pulsos rectangulares en el dominio del tiempo, y necesitamos que al hacer la FFT, y trasladar la señal al dominio de la frecuencia, esta señal tenga un espectro adecuado al canal de comunicación que se está utilizando. Por otro lado, también es conveniente que el ancho de banda utilizado sea mínimo. Cada una de las señales no debe ocupar demasiado ancho de banda para poder multiplexar la mayor cantidad posible de transmisores sobre un mismo canal.

5. Alta capacidad de detección de errores. Esto es, la diferencia entre un símbolo recibido y otro debe ser tal que, aunque la señal llegue distorsionada al receptor, este sea capaz de diferenciar los distintos símbolos de forma correcta.

### 3.3.1 TIPOS DE CÓDIGOS DE LÍNEA.

#### 3.3.1.1 Unipolar sin retorno a cero (NRZ).

Tiene la propiedad de transparencia, no necesita de símbolos anteriores, o de los siguientes para ser decodificado.

Espectro de densidad de potencia:

$$S_X(\omega) = \frac{A^2\pi}{2}\delta(\omega) + \frac{A^2 T_b}{4} Sa^2\left(\frac{\omega T_b}{2}\right)$$

Esta señal se obtiene a partir de:

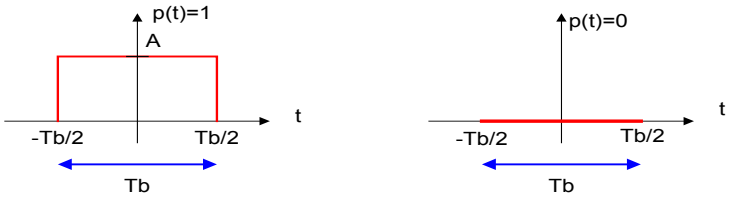



Figura 8: Código de línea unipolar NRZ.

#### 3.3.1.2 Unipolar con retorno a cero (RZ).

A mitad del $T_b$ (tiempo de bit) el pulso se va a cero. Se produce un descenso de la energía, potencia necesaria en la transmisión. Por otro lado, y como contrapartida, aumenta el ancho de banda de la señal transmitida, se convierte en el doble del que tenia.

Sigue siendo un código transparente.

Su espectro de densidad de potencia:

$$S_X(\omega) = \frac{A^2\pi}{8}\delta(\omega) + \frac{A^2 T_b}{16} Sa^2\left(\frac{\omega T_b}{4}\right) + \frac{A^2\pi}{8}\sum_{\substack{k=-\infty \\ kimpar}}^{\infty} Sa^2\left(\frac{k\pi}{2}\right)\delta(\omega - k\omega_b)$$

Figuras a partir de las que se ha obtenido:

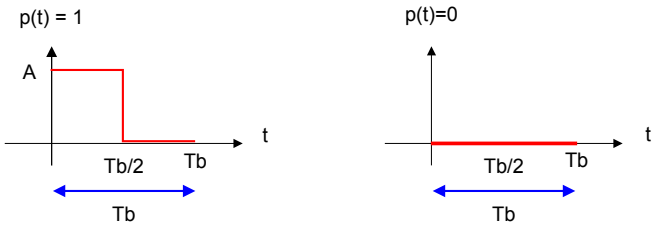



Figura 9: Código de línea unipolar RZ.

### 3.3.1.3 Polar NRZ.

Su ventaja principal está bajar la potencia media transmitida, pretende eliminar la componente de continua de la señal.

Es un código transparente

Su espectro de densidad de potencia:

$$S_X(\omega) = A^2 T_b Sa^2\left(\frac{\omega T_b}{2}\right)$$

Figuras a partir de las que se ha obtenido:

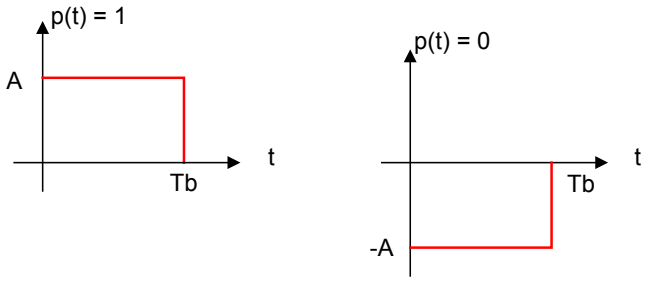



Figura 10: Código de línea polar NRZ.

### 3.3.1.4 Polar RZ.

Baja aún más la potencia media transmitida, a consta de doblar el ancho de banda necesario para la transmisión.

Es un código transparente.

Su espectro de densidad de potencia:

$$S_X(\omega) = \frac{A^2 T_b}{4} Sa^2\left(\frac{\omega T_b}{4}\right)$$

Figuras a partir de las que se ha obtenido:

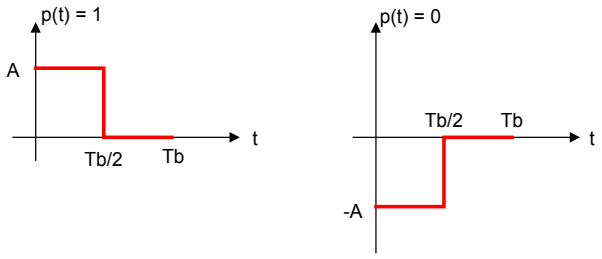



Figura 11: Código de línea polar RZ.

### 3.3.1.5 Bipolar NRZ.

Los bits '0' los codifica a cero voltios, y los bits a '1' los va alternando con $\{\pm A\}$. De este modo se comprueba que no estamos ante un código transparente, necesitamos conocer el bit anterior para poder decodificar de forma correcta. Tenemos que tener un convenio para el primer bit, y poner de acuerdo transmisor y receptor.

Su espectro de densidad de potencia:

$$S_X(\omega) = \frac{A^2 T_b}{4} \text{sen}^2\left(\frac{\omega T_b}{2}\right) Sa^2\left(\frac{\omega T_b}{2}\right)$$

Figuras a partir de las que se ha obtenido:

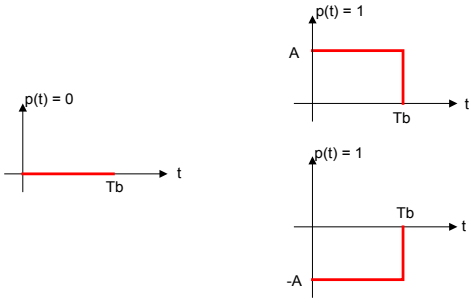



Figura 12: Código de línea bipolar NRZ.

Este es el estándar utilizado para la transmisión con PCM binario en canal telefónico.

### 3.3.1.6 Bipolar RZ.

Los bits "0" los codifica a cero voltios, y los bits "1" los va alternando a $\{\pm A\}$, además, a $T_b/2$ vuelve a cero.

No es un código transparente, por la misma razón que el caso anterior. Además tiene un ancho de banda doble del anterior.

Su espectro de densidad de potencia:

$$S_X(\omega) = \frac{A^2 T_b}{4} \text{sen}^2\left(\frac{\omega T_b}{2}\right) Sa^2\left(\frac{\omega T_b}{4}\right)$$

Esta densidad espectral de potencia se ha obtenido a partir de las señales:

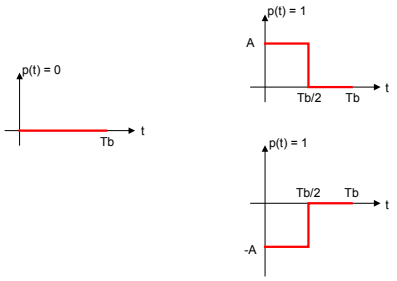



Figura 13: Código de línea bipolar RZ.

### 3.3.1.7 Manchester o señalización bifásica.

Para los bits '0', codificamos la primera mitad del pulso negativa, y la segunda mitad positiva. En el caso de los bits '1', va al contrario, se codifican la primera mitad del pulso positiva, y la segunda mitad negativa. Se ve en la Figura 14.

Ventajas de esta codificación:

- Tensión media, nivel de continua en el canal es nulo.
- Buena sincronización. Siempre se producen cambios, tanto si enviamos "0", o "1", seguidos o alternados. No hay problema al enviar un número grande de bits iguales seguidos, no se produce el desenganche por sincronización.

Su espectro de densidad de potencia:

$$S_X(\omega) = A^2 T_b \,\mathrm{sen}^2\left(\frac{\omega T_b}{4}\right) Sa^2\left(\frac{\omega T_b}{4}\right)$$

Esta señal densidad de potencia se ha obtenido a partir de:

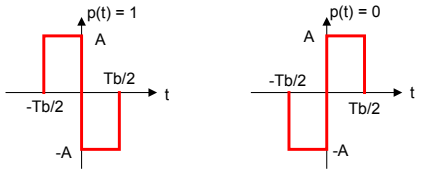



Figura 14: Código de línea Manchester.

### 3.3.1.8 Señalización M-aria multinivel.

Se utiliza cuando agrupamos varios bits en un mismo símbolo.

Si tenemos M bits, entonces necesitaremos:

$$\lceil \log_2 M \rceil \text{ bits / símbolo}$$

Pueden ser de dos tipos unipolares y bipolares. Esto se ilustra en la Figura 15.

Su espectro de densidad de potencia:

$$S_X(\omega) = 10 A^2 T_b Sa^2(\omega T_b)$$

Aquí, tenemos que el tiempo característico de símbolo depende del número de bits en cada símbolo. De este modo, el tiempo de símbolo nos viene dado por:

$$T_{simb} = T_b \lceil \log_2 M \rceil$$

**Ejemplo:**

Suponemos una señalización M-aria con M=4. Queremos saber como quedaría la forma de onda asociada a la transmisión de la secuencia: $\{1,1,0,0,0,1\}$, tanto con un código unipolar, como bipolar.

1. Vemos el número de bits/símbolo necesarios: $\lceil \log_2 M \rceil$. En nuestro caso la expresión queda como $\lceil \log_2 4 \rceil = 2$. Necesitamos 2 bits / símbolo.
2. Dividimos los bits a transmitir de dos en dos: $\{11,00,01\}$.
3. Vemos cuanto ocupara cada símbolo en el canal: $T_{simb} = T_b \lceil \log_2 M \rceil = 2.T_b$
4. Sólo nos queda por ver las formas de onda asociadas a cada una de las soluciones, según estemos en el caso unipolar, o bipolar:

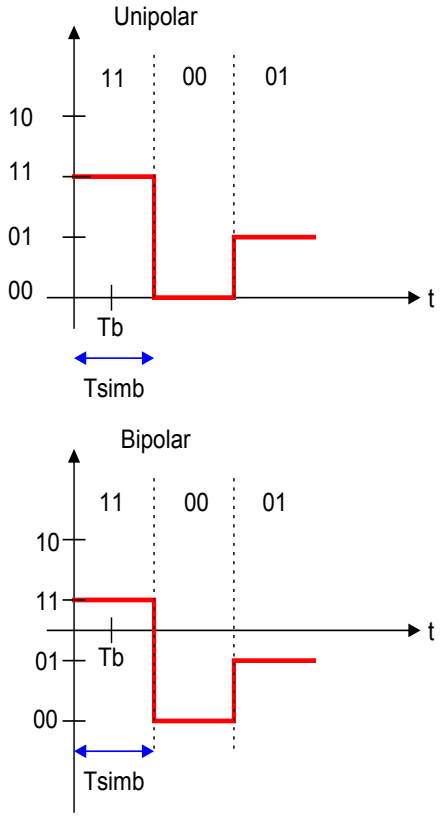



Figura 15: Señalización M-aria.

### 3.3.2 ESQUEMA COMPARATIVO DE LOS DIFERENTES CÓDIGOS DE LÍNEA.

La mejor forma para comparar los distintos códigos es verlos todos juntos. Esto se ilustra en la siguiente figura, en la que aparecen todos los códigos de línea anteriores, excepto el M-ario o multinivel, ya que este no es fijo, depende de varios parámetros.

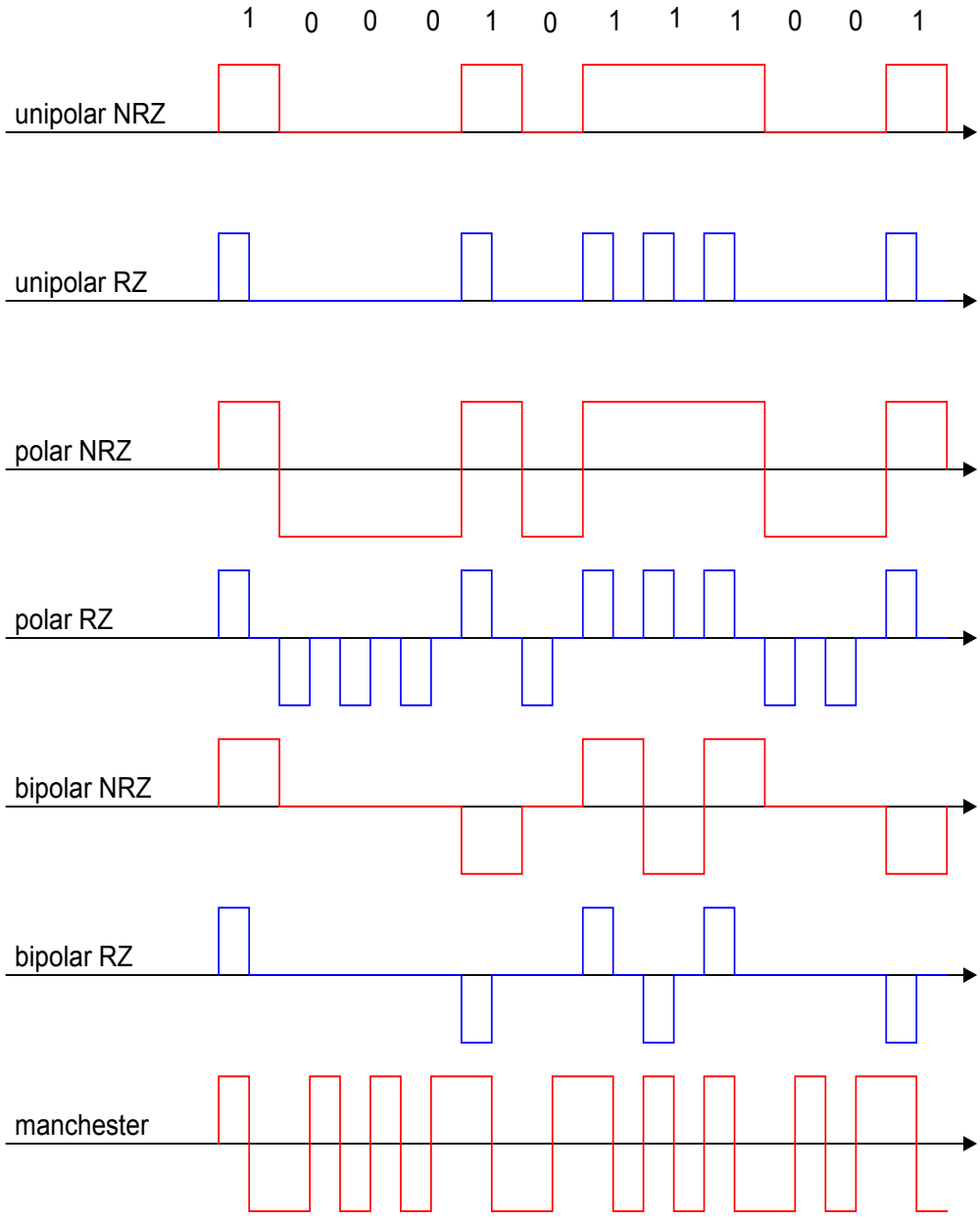



Figura 16: Comparativa de los distintos códigos de línea.

## *3.4 MODULADOR DIGITAL.*

Sirve como interfaz del canal. Ante la capacidad del canal para transmitir señales eléctricas en forma de onda, el principal propósito del modulador es mapear la secuencia de información binaria en secuencias no binarias que puedan ser enviadas por el canal. Para conseguir este propósito tenemos varias alternativas según el tipo de modulación que se vaya a utilizar.

Por un lado, tenemos los sistemas en banda base, en los que no necesitamos una portadora para trasladar la señal por el canal, sino que el canal tiene como frecuencia central fc=0 Hz.

Por otro lado, se encuentran los sistemas paso de banda, en los que sí se necesita una portadora para trasladar la señal a transmitir hasta la frecuencia deseada.

A continuación se ven con más detalle los tipos de modulación digital tanto en banda base como paso de banda.

### 3.4.1 MODULACIÓN DIGITAL EN BANDA BASE (BB).

Asocia una determinada forma de onda a un vector de entrada.

Figura 17: Esquema de bloques para la modulación en banda base.

Dividimos la cadena de datos de entrada en vectores de N datos, por tanto, se van a necesitar $M= 2^N$ símbolos de salida, suponiendo los datos de entrada binarios.

La forma de asociar estas $2^N$ señales a los vectores de salida es lo que da lugar a las distintas modulaciones. En banda base, las señales moduladas se denominan señales moduladas por pulsos.

### 3.4.1.1 Modulación por Amplitud de Pulsos (PAM).

En este caso, la información viaja en la amplitud de la señal:

$$S_i(t) = A_i \cdot g_T(t) \qquad \text{donde} \quad 0 \leq i \leq 2^N$$

El pulso $g_T(t)$ es fijo. En la Figura 18 aparece su representación gráfica.

La constelación para esta señal es unidimensional, con una distancia igual entre símbolos adyacentes, pero con una energía que va aumentando a medida que aumenta la amplitud de la señal.

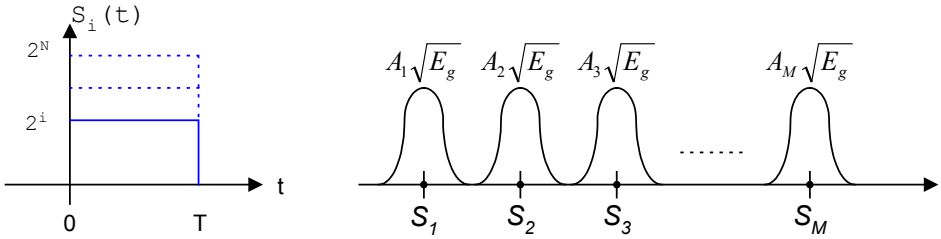



Figura 18: Señal PAM. Representación gráfica y constelación.

### 3.4.1.2 Modulación por Posición de Pulsos (PPM).

En este caso, la información va en el intervalo de tiempo en que se transmite el pulso, tenemos señales ortogonales con amplitudes constantes y por tanto energías constantes.

Ahora la señal a transmitir será:

$$S_i(t) = A \cdot g_i(t) \qquad \text{donde} \quad 0 \leq i \leq 2^M$$

La representación de $S_i(t)$ quedaría:

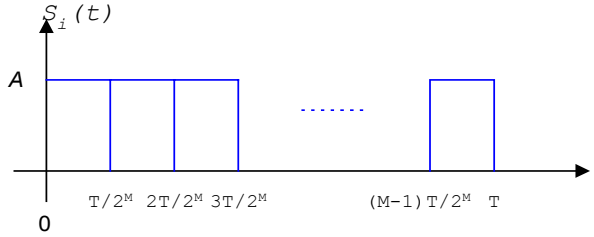



Figura 19: Señal PPM.

Una representación de la constelación es difícil, puesto que tenemos una constelación M-dimensional en la que todos lo símbolos se encuentran a la misma distancia entre ellos, ya sean adyacentes o no.

### 3.4.1.3 Señales Biortogonales.

Las señales biortogonales son un intermedio entre los dos casos anteriores, modulamos tanto en amplitud como por posición de pulsos. Para ello, se toman M/2 señales ortogonales, como en PPM, y otras M/2 señales iguales a las anteriores excepto en el signo, modulación PAM. Por tanto, conseguimos señales biortogonales.

Su expresión matemática viene dada por:

$$S_K(t) \equiv \begin{cases} S_i(t) & 1 \le i \le M/2; 1 \le k \le N/2 \\ -S_i(t) & 1 \le i \le M/2; (N/2)+1 \le k \le N \end{cases}$$

Una posible representación gráfica resulta difícil por el mismo motivo que en PPM, con la salvedad de que aquí el espacio de representación de las señales sería N/2 dimensional.

### 3.4.1.4 Señales Simplex.

Se toma un conjunto de M señales ortogonales y se genera otro conjunto de M señales restando la media del conjunto inicial. Esto es:

$$S'_i(t) = S_i(t) - \frac{1}{M}\sum_{k=1}^{M} S_k(t) \qquad \text{donde} \quad 0 \le t \le T,\ 0 \le i \le M$$

Conseguimos un conjunto de M señales ortogonales, con energía óptima.

La representación gráfica tiene el mismo problema que en señales PPM.

## 3.4.2 MODULACIÓN DIGITAL EN SISTEMAS PASO DE BANDA (BP).

Este apartado de modulación es el más interesante desde nuestro punto de vista, puesto que nos introduce a las modulaciones que van a ser objeto de estudio.

Una señal susceptible de ser enviada por un canal BP viene dada por la expresión:

$$S_i(t) = A_i(t).\cos(f_i(t) + \phi_i(t))$$

en esta expresión podemos observar que la señal de información puede viajar en la amplitud $A_i(t)$, en la frecuencia $f_i(t)$, o en la fase $\phi_i(t)$. Además, esta información se puede dividir en datos binarios o M-arios.

Si se unen estos dos conceptos, obtenemos los tres tipos básicos de modulación BP, Modulación por Conmutación de Amplitud ASK (Amplitude Shift Keying), Modulación por Conmutación de Frecuencia FSK (Frecuency Shift Keying) y por último Modulación por Conmutación de Fase PSK (Phase Shift Keying).

Pero, además de estos tres tipos básicos, existen modulaciones híbridas, entre las que destacamos la Modulación en Amplitud y Cuadratura más conocida como QAM, y también denominada APK (Amplitude-Phase-Keying).

Para recepción, tendremos señales coherentes y no coherentes. Esto es, las primeras suponen sincronización de referencia de fase de portadora, y las segundas no. Nosotros nos vamos a centrar en las coherentes por ser más generales.

### 3.4.2.1 Modulación ASK.

Vienen de las señales PAM de banda base, su expresión matemática está dada por:

$$S_i(t) = A_i.g_T(t).\cos(\omega_c.t) \qquad \text{donde} \quad 0 \le t \le T,\ 0 \le i \le M$$

Su constelación es igual a la de una señal PAM y viene dada por:

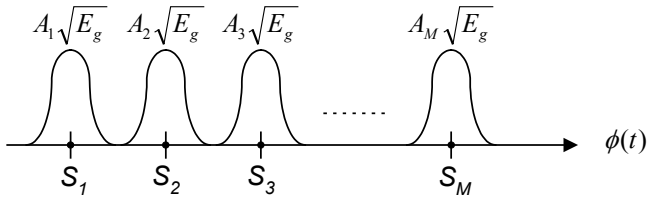



Figura 20: Constelación M-ASK.

### 3.4.2.2 Modulación PSK.

La señal de información viene dada por:

$$S_i(t) = A.g_T(t).\cos(\omega_c.t + \theta_i) \qquad \text{donde} \quad 0 \le t \le T,\ 0 \le i \le M$$

$$\text{con} \quad \theta_i = {}^{2\pi}\!/_{M}\,(i-1)$$

Su constelación tiene forma de circunferencia con símbolos adyacentes equiespaciados. La distancia intersímbolo depende del número de ellos. Vemos a continuación algún ejemplo que ponga esto de manifiesto.

**Ejemplos:**

1. 2-PSK, mas conocida como BPSK.

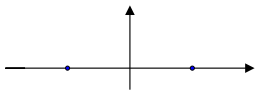


Figura 21

2. 4-PSK, mas conocida como QPSK.

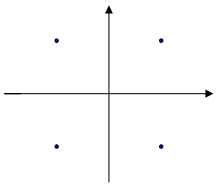


Figura 22

3. 8-PSK.

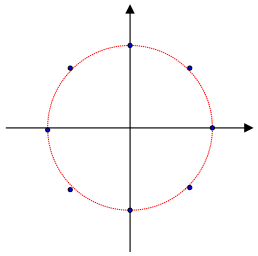


Figura 23

Por último decir que estas modulaciones pueden aparecer rotadas un ángulo cualquiera sin que esto afecte a la transmisión.

### 3.4.2.3 Modulación QAM.

Es un híbrido entre ASK, y PSK, por tanto va a tener características comunes con ambas. La señal de información va a venir dada por:

$$S_i(t) = r_i . g_T(t) . \cos(\omega_c . t + \theta_i) \qquad \text{donde} \quad 0 \le t \le T,\ 0 \le i \le M$$

$$\text{con} \quad r_i = \sqrt{A_{i1}^2 + A_{i2}^2}$$

$$\theta_i = \text{arctg}(A_{i2} / A_{i1})$$

Existen muchas posibles constelaciones, puesto que se varia tanto amplitud como fase, vemos varios ejemplos de una constelación que se denomina de igual modo en todos los casos.

**Ejemplos:** Modulación 8-QAM.

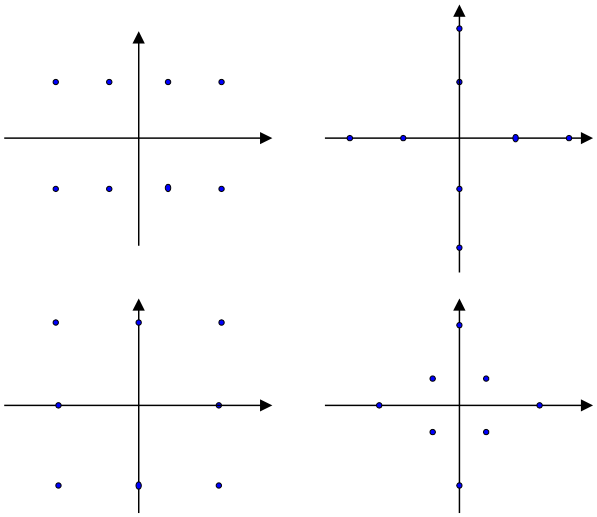


Figura 24: Distintas constelaciones posibles para una 8-QAM.

### 3.4.2.4 Modulación FSK.

En este caso la información va codificada en la frecuencia de la señal portadora, la expresión de la señal de información queda como:

$$S_i(t) = A.g_T(t).\cos(\omega_i.t + \theta) \qquad \text{donde} \quad 0 \leq t \leq T,\ 0 \leq i \leq M$$

con $\omega_i = \omega_c + (i-1).\Delta\omega$, llamada frecuencia instantánea.

La constelación para esta modulación es difícil de describir, puesto que al igual que ocurría con PPM, los símbolos son ortogonales y están equiespaciados entre sí. Volvemos a tener por tanto un espacio N-dimensional que depende del número de símbolos posibles.

## *3.5 CANAL.*

El canal de comunicaciones es la interfaz que tiene que atravesar la señal entre el transmisor y el receptor, tal y como se ha visto en la Figura 2. Estos canales son los que introducen ruido, desvanecimientos, interferencias..., y otras distorsiones en las señales transmitidas a través de ellos.

Simular un sistema de comunicación implica el modelado del canal, basándonos en una descripción matemática de éste.

Los diferentes medios de transmisión tienen diferentes propiedades, y, por tanto, son modelados de forma diferente.

Para una simulación, en un esquema, como el que se ha visto en la Figura 2, el modelado del canal tiene que corresponderse con el bloque situado entre el transmisor y el receptor.

A continuación se detallan distintos tipos de canales, modelos matemáticos generales, y los canales típicos de simulación.

### 3.5.1 CLASIFICACIÓN DE CANALES SEGÚN EL TIPO DE MEDIO DE TRANSMISIÓN UTILIZADO.

#### 3.5.1.1 Canales de líneas de cable.

Generalmente se agrupan formando redes de cable de pares o coaxial, con un BW modesto.

El cable de pares se usa básicamente en redes telefónicas para conectar al usuario a una central, con un BW de cientos de KHz. Por otro lado, el cable coaxial nos proporciona un BW algo mayor, del orden de MHz.

En estos canales, el ruido afecta tanto a la amplitud como a la fase de la señal transmitida, se modela matemáticamente como ruido aditivo.

#### 3.5.1.2 Canales de fibra óptica.

Ofrecen al diseño de sistemas de telecomunicaciones un BW mucho más elevado que en el caso anterior, además de una muy baja atenuación.

Esto ha hecho que su uso se extienda de forma muy rápida en los últimos años, siendo este cada vez más común.

En este tipo de medio, la señal de información viaja en forma de luz y se amplifica periódicamente, con lo que se va compensando la atenuación que haya podido sufrir. Las pérdidas o fuentes de ruido se encuentran principalmente en los transmisores, posibles empalmes, amplificadores y receptores.

#### 3.5.1.3 Canales vía radio (Canales electromagnéticos).

En estos tipos de sistemas, la energía electromagnética se acopla al medio de transmisión a través de una antena.

El modo de propagación de ondas electromagnéticas se divide según el rango de frecuencias que se vaya a utilizar.

El ruido se va a deber principalmente a interferencias electromagnéticas de unas señales con otras, y desvanecimientos provocados principalmente por el clima, tormentas, etc.,...

#### 3.5.1.4 Canales submarinos.

Tenemos un canal multitrayecto, debido a las reflexiones desde la superficie y el fondo del mar.

Debido al movimiento de las olas, estas señales multitrayecto se retrasan unas respecto a otras, lo que provoca el desvanecimiento de la señal. Además, la atenuación de la señal depende de la frecuencia. Tenemos también otras fuentes de ruido, provocadas por el ambiente marino que por su propia naturaleza es ruidoso, y por el hombre cuando nos encontramos cerca de un puerto. Esta última fuente de ruido, se le suma a las anteriores.

Aún así, estos sistemas se están imponiendo para la transmisión de señales digitales en largas distancias, como son las intercontinentales.

### 3.5.2 MODELOS MATEMÁTICOS PARA CANALES DE COMUNICACIONES.

#### 3.5.2.1 Canal de ruido aditivo.

Es el modelo más simple para un canal de comunicaciones. En este modelo, la señal transmitida, denominada $s(t)$, es corrompida por una señal aleatoria, denominada $n(t)$, que se le suma. Además, la señal puede sufrir una cierta atenuación que vendrá dada por un factor $\alpha$. Matemáticamente, esto se modela como:

$$r(t) = \alpha.s(t) + n(t)$$

La señal $n(t)$ responde a una muestra de un proceso aleatorio, con una PDF determinada. Se aproxima por una u otra según el tipo de canal.

### 3.5.2.2 Canal de filtro lineal.

En algunos canales físicos, como las líneas telefónicas, los filtros son utilizados para asegurar que las especificaciones de BW son cumplidas por las señales transmitidas y no se interfieren unas con otras.

Estos canales son caracterizados matemáticamente de forma general como canales con filtros lineales y ruido aditivo.

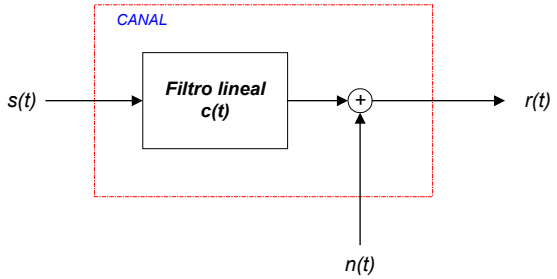



Figura 25: Canal de filtro lineal.

La señal de salida:

$$r(t) = s(t) * c(t) + n(t)$$

### 3.5.2.3 Canal de filtro lineal variante con el tiempo.

Con este modelo matemático simulamos canales como por ejemplo los submarinos, o los de propagación ionosférica (vía radio), en los que tenemos propagación multitrayecto y variación de la señal con el tiempo.

Para este tipo de canales, se necesita un filtro lineal, pero variable con el tiempo. Su respuesta viene dada por $c(\tau;t)$, donde la variable $\tau$ representa el retraso, el tiempo que ha transcurrido. Además como ya se explicó en los tipos de canales, también tienen ruido aditivo, que es común en todos.

Su modelo:

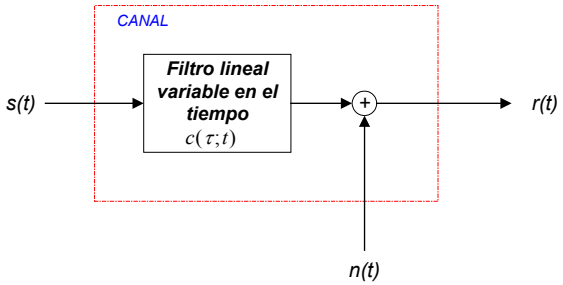



Figura 26: Esquema de un canal de filtro lineal variante en el tiempo.

La señal de salida:

$$r(t) = s(t) * c(\tau;t) + n(t)$$

### 3.5.3 CANALES MÁS UTILIZADOS EN LA SIMULACIÓN DE SISTEMAS.

Por último, en este apartado, veremos los tres tipos de canal, junto con su modelado matemático, más utilizados en las simulaciones de sistemas digitales.

#### 3.5.3.1 Canal AWGN (Additive White Gaussian Noise).

Es el ruido más simple modelado en la simulación de un canal. Básicamente se define como el ruido que se genera en el receptor cuando el camino de la transmisión es ideal. [5]

Se caracteriza principalmente por tener una PSD constante en todo el BW del canal, e igual a:

PSD (f)= No/2 [W/Hz]

La amplitud de la señal de ruido, vendrá dada por una PDF gaussiana de media cero y varianza dependiente de la amplitud de la señal de entrada al canal.

Su modelado matemático es muy sencillo, se consigue con una simple función suma. La función de salida del canal se representa por:

$$y(t) = x(t) + n(t)$$

donde $y(t)$ es la señal de salida, $x(t)$ la señal de entrada y $n(t)$ representa la señal de ruido blanco y gaussiano (WGN).

Este es el canal que se ha utilizado en la simulación de nuestro sistema, lo vemos de forma mas concreta que los demás, viendo las ecuaciones que se utilizan en el modelado.

En primer lugar, se necesita conocer la varianza de ruido, que, como se ha dicho antes se relaciona con la amplitud de la señal de entrada por medio de su potencia. Por tanto el primer paso será el cálculo de la potencia de la señal:

$$P = \frac{E}{T} = \frac{\sum |x(t)|^2}{T}$$

donde $E$ es la energía de la señal, y $T$ el periodo de tiempo en que transcurre.

Ahora si podemos calcular la varianza, pero teniendo en cuenta el ruido que se introduce en el canal. Hay canales gaussianos más ruidosos, y otros menos ruidosos, o incluso el mismo canal no siempre se ve afectado de la misma manera por el ruido, ya que un canal es algo dinámico, y depende de muchos factores externos. Por tanto, la varianza se v a relacionar directamente con la SNR de la siguiente forma:

$$\sigma = \sqrt{\frac{P \diagup SNR}{2}}$$

Una vez que tenemos la varianza, simulamos cada una de las muestras del proceso aleatorio $n(t)$ como una muestra de la variable aleatoria $n(t_n) = N(0, \sigma)$. Cada una de estas muestras es la que se le suma a la señal de entrada.

### 3.5.3.2 Canal "Rayleigh fading".

Si en el receptor, cada una de las componentes de la señal recibida es independiente, entonces la PDF de la envolvente es Rayleigh.

Un generador sencillo de modelar matemáticamente nos viene dado por la Figura 27.

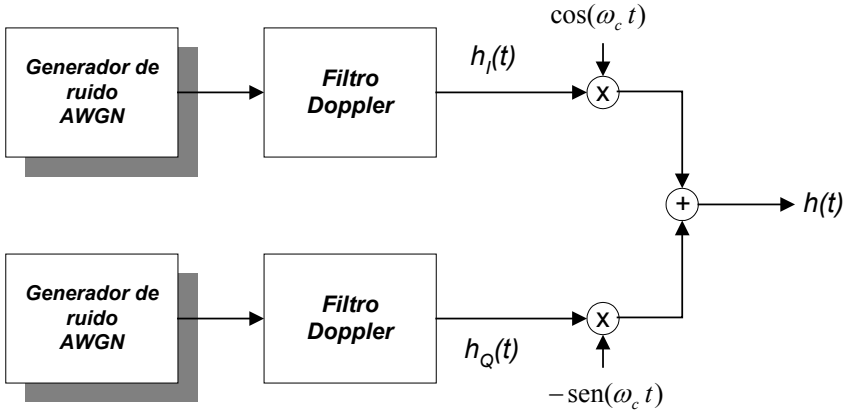



Figura 27: Esquema de generación de un canal Rayleigh.

La salida de este generador representa la respuesta impulsiva del canal, y viene dada en fase y cuadratura por:

$$h(t) = h_I(t).\cos(\omega_c.t) - h_Q(t).\text{sen}(\omega_c.t)$$

También se puede representar en módulo y fase como:

$$h(t) = R(t).\cos(\omega_c.t + \psi(t))$$

donde $R(t) = \sqrt{h_I^2(t) + h_Q^2(t)}$ es la envolvente , con distribución Rayleigh,

y $\psi(t) = \text{tg}^{-1}\left(h_Q / h_I\right)$ es la fase, de distribución uniforme.

Por tanto, una vez vista la respuesta impulsiva del canal, la señal recibida viene dada por:

$$r(t) = h(t) * x(t) + n(t)$$

donde $x(t)$ es la señal de entrada al canal, y $n(t)$ es un posible ruido aditivo que siempre puede aparecer.

### 3.5.3.3 Canal "Rician fading".

Este tipo de canal se simula en sistemas de comunicaciones móviles, en los que, en el canal existe un camino dominante, directo denominado "line-of-sigth" LOS, además de los secundarios. Si en el camino dominante la probabilidad de desvanecimiento es mucho menor que en los otros casos, entonces decimos que la envolvente recibida es una variable aleatoria tipo "Rician".

El parámetro para una PDF Rician viene dado por:

$$k = \frac{Potencia\ del\ camino\ pricipal}{Potencia\ de\ los\ caminos\ \sec undarios}$$

También se podría haber definido el cociente contrario. Nótese además que, si $k = 0$ el canal es Rayleigh, y, si $k = \infty$ el canal es gaussiano.

El diagrama de bloques para representar un canal Rician, es el dado por la siguiente figura:

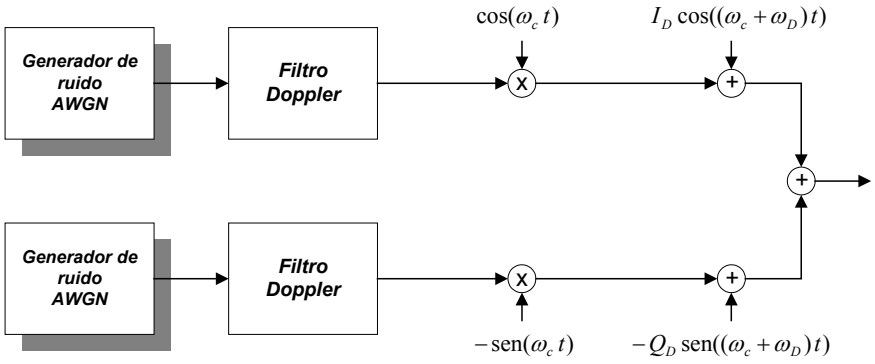



Figura 28: Esquema para generar un canal Rician.

En él se ve la importancia del camino principal o dominante. La señal de salida viene dada por:

$$r_D(t) = \left(I_D^2 + Q_D^2\right)^{1/2}.\cos\left(\left(\omega_c + \omega_D\right)t + \phi_D\right)$$

donde $I_D, Q_D$ son las amplitudes de la señal en cuadratura, $\left(\omega_C + \omega_D\right)$ es la frecuencia angular del camino dominante, y por último $\phi_D = \mathrm{tg}^{-1}\left(Q_D \big/ I_D\right)$ representa la fase.

A la vista de esta gráfica, y comparándola con la de un canal Rayleigh, otra forma de obtener el parámetro $k$ sería obtener el valor cuadrático medio de $r_D(t)$ y dividirlo por el obtenido de $h(t)$. De este modo, ponemos de manifiesto la relación real que existe entre ambas variables aleatorias.

Para terminar, decir que, en muchos casos, la aleatoriedad de un canal no se adapta totalmente a un canal Rayleigh, Gaussiano o Rician. En estos casos, tomamos un canal Rician para su simulación, por ser el caso más general, y vamos variando su $k$ para adaptarlo de la forma más adecuada posible.

## *3.6 DEMODULADOR.*

Es el proceso por el cual la señal recibida se transforma en una secuencia binaria de datos que coinciden con patrones ideales en transmisión. Se divide en dos pasos, la demodulación en sí, que es el proceso por el cual a partir de la señal de entrada $x(t)$, sacamos un vector de datos $\vec{x}$; y la detección, en la que a partir de los datos del vector $\vec{x}$, corrompidos por el ruido del canal, se obtiene el vector $\vec{m}$ que contiene datos que coinciden con los patrones ideales en el transmisor. El detector o receptor vectorial también es denominado "demapper" (término heredado del Inglés).

### 3.6.1 DEMODULACIÓN.

Tenemos dos tipos de demoduladores principalmente, y que, se van adaptando a las distintas modulaciones. Estos son los demoduladores por correlación, y por filtros adaptados.

#### 3.6.1.1 Demodulador de correlación.

Un demodulador de correlación corresponde al siguiente esquema:

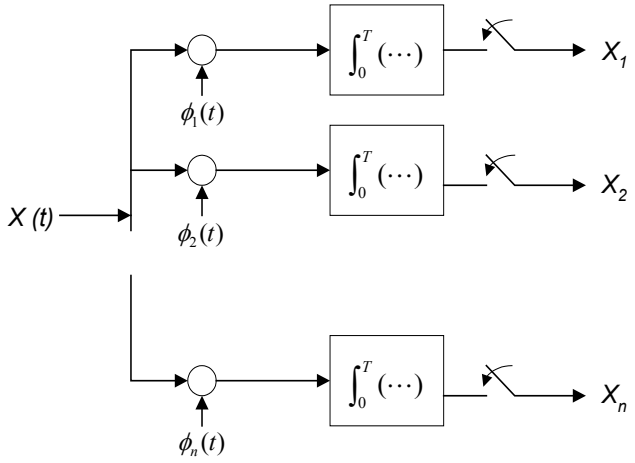



Figura 29: Demodulador de correlación.

donde $\phi_i(t)$ son las funciones base, que varían según la modulación.

El vector de entrada:

$$x(t) = s(t) + n(t)$$

donde $s(t)$ es la señal transmitida, y $n(t)$ es el ruido que introduce el canal.

Tras el integrador:

$$x_i(t) = \int_0^T [s(t) + n(t)]\phi_i(t).dt = \int_0^T s(t)\phi_i(t).dt + \int_0^T n(t)\phi_i(t).dt = s_i(t) + \omega_i(t)$$

Con este esquema, tras el muestreador, la señal: $x_i = s_i + \omega_i \qquad i = 1, \ldots, n$

Si unimos todas las salidas, obtenemos el vector $\vec{x} = (x_1, x_2, x_3, \ldots, x_n)$, que será la entrada del detector.

### 3.6.1.2 Demodulador de filtros adaptados.

Un demodulador de filtros adaptados corresponde a la siguiente gráfica:

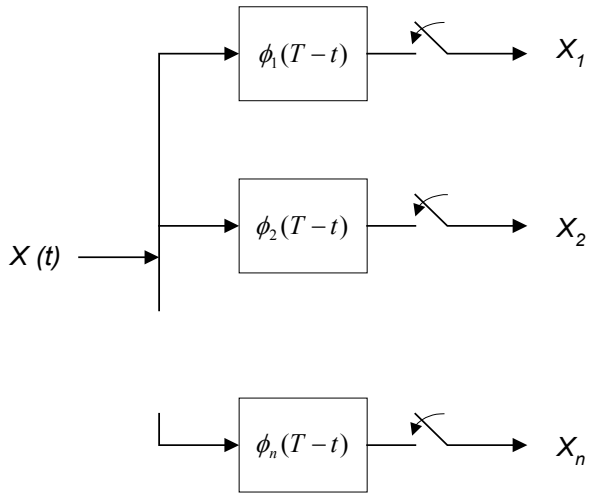



Figura 30: Demodulador de filtros adaptados.

Los filtros adaptados se definen como $h_i(t) = \phi_i(T-t)$, son filtros adaptados a cada una de las funciones base de la modulación correspondiente. La señal de entrada, al igual que en el caso anterior, viene dada por:

$$x(t) = s(t) + n(t)$$

Tras el filtro:

$$x_i(t) = x(t) * h_i(t) = \big(s(t) + n(t)\big) * \phi_i(T-t) = \big(s(t) * \phi_i(T-t)\big) + \big(n(t) * \phi_i(T-t)\big) = s_i(t) + \omega_i(t)$$

Después del muestreador: $x_i = s_i + \omega_i \qquad i = 1, \ldots, n$

El vector de salida, por tanto, queda: $\vec{x} = (x_1, x_2, x_3, \ldots, x_n)$

Los filtros adaptados tienen una propiedad fundamental, y esta nos dice que si a la entrada tenemos una señal corrompida por ruido AWGN, el filtro adaptado a $s(t)$, maximiza la $(SNR)_o$ esto es, a la salida del mismo, tras el muestreador.

### 3.6.2 DETECCIÓN.

Una vez que tenemos el vector de símbolos ruidosos hay que hacerlos corresponder con los símbolos de entrada que le corresponden. Este es el proceso de detección, en el cual también entra la cuantificación.

El esquema es sencillo, tal y como nos muestra la Figura 31:

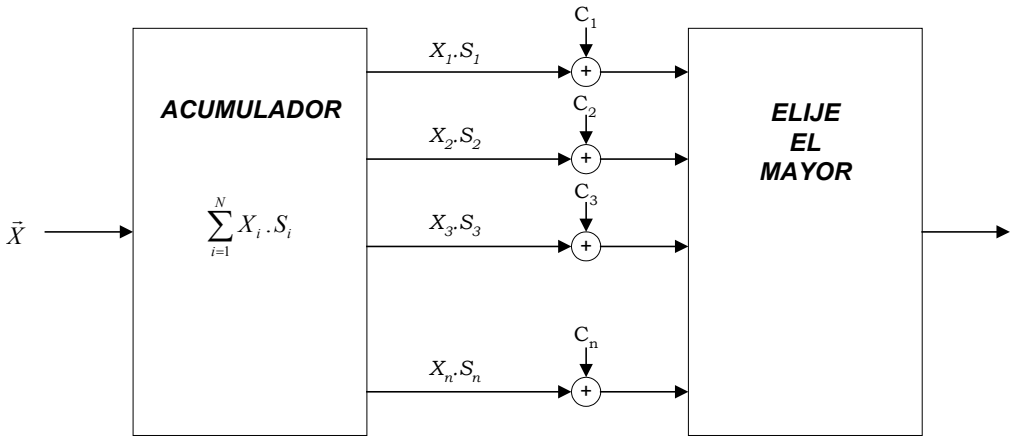



Figura 31: Detector.

La notación utilizada en la gráfica coincide con la utilizada en el apartado anterior de demodulación. El único parámetro nuevo es el $C_i$, al que llamamos métrica, puesto que es el valor que almacena la distancia óptima del valor ideal que se debe recibir. Su valor viene dado por:

$$C_i = -\frac{1}{2} E_i + \frac{N_o}{2} \ln(p(m_i)) \qquad 1 \le i \le n$$

Existen dos detectores, que se diferencian en el modo de calcular la métrica. Uno es el detector ML (máxima verosimilitud), en el cual los símbolos de entrada son equiprobables, y el detector MAP (máxima probabilidad a posteriori) para símbolos no equiprobables.

Este es un esquema general de detección, pero las métricas a utilizar pueden ir variando según la modulación utilizada, consiguiendo detectores óptimos.

## *3.7 DECODIFICADOR DE CANAL.*

Según el tipo de codificación utilizada, tenemos uno u otro método para la decodificación. Abordamos desde aquí los más significativos. Para códigos lineales de bloque, vemos "*Decodificación de mínima distancia*" y "*Decodificación del síndrome*". Para códigos convolucionales, introducimos el "*Decodificador de Viterbi*".

### 3.7.1 DECODIFICACIÓN DE MÍNIMA DISTANCIA.

Es el método más sencillo, pero a la vez el de mayor coste computacional.

Antes de ver el método una definición previa:

- *"Peso Hamming"*: es el número de 1´s en un determinado vector. Si llamamos "b" al vector, el peso Hamming se denota como $\omega_H(b)$.

Vemos el método por pasos:

- **Paso 1**: A partir del vector recibido a la entrada, al que denotamos por $\underline{Y}$ se calculan todos los vectores con error. Para ello se suman todas las posibles palabras de código, $\underline{C_i}$ :

$$\underline{Y} \oplus \underline{C_i} = \underline{e_i}$$

Estas sumas son las que hacen que aumente el coste computacional.

- **Paso 2**: Calculamos el peso Hamming de cada uno de los vectores $\underline{e_i}$ . Un uno en una determinada posición indica que la palabra de código difiere con el vector recibido en dicha posición. En este paso obtenemos $\omega_H(\underline{e_i})$ .
- **Paso 3**: Nos quedamos con el vector $\underline{e_i}$ que tenga el $\omega_H(\underline{e_i})$ menor, y calculamos $\underline{\hat{C}_i}$ , que es la palabra de código estimada, esto es, la que se supone que ha sido transmitida.

$$\underline{Y} \oplus \underline{e_i} = \underline{\hat{C}_i}$$

### 3.7.2 DECODIFICACIÓN DEL SÍNDROME.

Es un método más complicado que el anterior, pero a cambio con menor coste computacional.

Antes de exponerlo, unas definiciones previas:

- "*Vector de Síndrome del error*": se denota por $\underline{S}$, y se define como $\underline{S} = \underline{Y} \cdot \underline{H}^t$ donde $\underline{Y}$ es el vector recibido, y $\underline{H}^t$ es la matriz de chequeo de paridad traspuesta.

- "*Tabla Estándar*": tabla que contiene todos los patrones de error, es fija según los códigos. Esto es no hay una para cada código.

- "*Tabla de Síndromes*": tabla que asocia cada uno de los síndromes a un error determinado.

Como en el caso anterior, vemos el algoritmo por pasos:

- **Paso 1:** Se calcula el síndrome asociado al vector recibido $\underline{S} = \underline{Y} \cdot \underline{H}^t$, necesitamos la matriz $\underline{H}^t$ del transmisor.

- **Paso 2:** Se construye la tabla de síndromes para ese tipo de código. Una vez que tenemos la tabla, se mira en ella el patrón de error que corresponde.

- **Paso 3:** Una vez que tenemos el patrón de error, obtenemos la palabra de código estimada.

$$\underline{Y} \oplus \underline{e_i} = \underline{\hat{C}_i}$$

### 3.7.3 DECODIFICACIÓN DE VITERBI: DECODIFICACIÓN DE CÓDIGOS CONVOLUCIONALES.

Un codificador convolucional, es básicamente una máquina de estados, por lo tanto, un decodificador óptimo seria un MLSE para señales de codificación de línea con memoria (NRZI, CPM). [6]

Por lo tanto, para la decodificación óptima de un código convolucional supone la búsqueda a través del "Trellis" de la secuencia mas probable.

Aunque existen tres tipos principales de decodificadores para códigos convolucionales; los secuenciales, los basados en un determinado umbral y las técnicas de Viterbi, estos últimos son los más utilizados. Esto es porque si la distancia utilizada para computar el algoritmo de Viterbi es la euclídea tenemos que es MLSE si en el canal el ruido es AWG (Additive Gaussian Noise).

# 4 ESTÁNDAR DVBT.

## *4.1 INTRODUCCIÓN.*

Hoy en día, hablar de TV Digital, es hablar de TV Digital Terrestre. Ésta comprende el último paso en el trayecto de la emisión, esto es, desde la antena hasta el usuario. Hay que hacer notar que además de ser el último paso en la cadena de transmisión, es el último eslabón en la digitalización.

La TV Digital es un medio de comunicación que va a estar al servicio de todos los usuarios, puesto que poco a poco va a ir sustituyendo a la actual TV analógica, ganando en prestaciones y calidad de servicio.

El retraso sufrido en la digitalización de la TV responde principalmente a la incompatibilidad del nuevo sistema con el actual, siendo necesario un cambio de equipos, tanto para las operadoras, como para los usuarios, o, en este último caso, la incorporación de un dispositivo denominado "Set Top Box".

De esta manera, aunque el cambio va a producir una evolución en el servicio actual, su introducción no será fácil, ya que supone inversiones importantes. Será un proceso lento pero ventajoso.

Entre estas mejoras, podemos citar un aumento de la calidad de la señal, gracias a la multiregeneración, técnicas de corrección de errores, aumento del BW para el color,..., con todo esto, conseguimos la denominada HDTV (High Definition TV/ TV de alta definición).

Otras mejoras que se introducen son el sonido 3D, o sonido envolvente; ofertar nuevos servicios, como VoD (Video bajo demanda), EPG (Electronic Programming Guide), recepción en terminales móviles,... y mejora en la conectividad, esto es, lima diferencias entre los estándares de TV existentes.

Pero, si tuviésemos que destacar dos ventajas como las más significativas o importantes, estas serían sin duda la reducción de costes totales en transmisión, recepción y equipos, así como la reducción del BW para cada uno de los canales. En TV Digital, el BW de un canal corresponde a una tercera o una cuarta parte del BW de un canal analógico.

Para tener un marco de referencia en el que basar la transmisión, existen hasta cuatro normativas:

- DVB-T propuesta por Europa.
- ATSC propuesta por EEUU.
- ISDB propuesta por Japón.
- DMT propuesta por China.

De entre ellas, sólo para las dos primeras existen equipos de transmisión y recepción que se producen a escala industrial. Este estudio está basado en DVB-T, que es la normativa europea. A partir de aquí se ven sus características generales.

Si se acude a la norma publicada por la IEEE, para TV Digital [8], se puede encontrar el esquema de bloques funcionales básicos. Vemos que la transmisión del sistema se basa en OFDM (Othogonal Frecuency Division Multiplexing). OFDM es una técnica de modulación basada en FDM (Frecuency Division Multiplexing), técnica que permite a su vez la división de información en portadoras para ser enviadas por un canal. Por tanto, OFDM se presenta como alternativa para la transmisión de datos de banda ancha en canales dispersivos y selectivos en frecuencia, ya que permite la disminución de ecos y el empleo de técnicas de compensación de desvanecimientos. Da lugar a los denominados "*sistemas OFDM*".

En la actualidad, para la generación de esta señal, se utiliza la FFT (Fast Fourier Trasform), esto permite un procesamiento dual tiempo-frecuencia. Esta dualidad, es aprovechada desde el punto de vista de la detección y corrección de errores, dando lugar a los sistemas COFDM (Coded OFDM).

Por tanto, en realidad lo que se presenta en la normativa para DVB-T es un sistema COFDM.

## *4.2 OFDM: PRINCIPIOS DE FUNCIONAMIENTO.*

OFDM se basa en el reparto de los datos a transmitir en varias portadoras para ser enviados por el canal. De este modo la tasa de cada una de estas portadoras es menor que la tasa total de transmisión, ocupando cada una un BW determinado dentro del BW total del canal. Como consecuencia, la longitud del símbolo aumenta. Otra característica principal es que los símbolos deben estar adaptados para ocurrir simultáneamente y el sistema pueda funcionar correctamente.

Todas estas portadoras disponen del mismo espacio en frecuencia, división del total. Esta duración en frecuencia, es la inversa del espacio temporal ocupado por cada una, y se denomina "*periodo de símbolo activo*" (F). Este es el periodo en el que el receptor examina la señal recibida. Esta elección del espacio de portadora asegura la ortogonalidad entre ellas, de manera que, el demodulador para una de ellas no interfiere con las demás, con lo que no es probable un cruce entre portadoras ni solapamiento de datos.

Un avance posterior fue la inclusión del llamado "*intervalo de guarda*" (IG). Esto es, cada símbolo modulado es transmitido en un periodo total superior a F, esta diferencia entre tiempos es el denominado IG. Con esto, el receptor, no tendrá interferencias intersímbolo o inter-portadoras provocadas por posibles retrasos en la señal siempre que no se exceda este IG.

Una desventaja de este avance es la disminución de la capacidad de transmisión, que va a depender de su duración. En OFDM, es posible la protección contra ecos y retrasos largos sin tomar como IG una parte importante del F. Los IG que se utilizan en DVB-T, así como los retrasos permitidos, se recogen en la norma [8].

En los procesos de modulación y demodulación, lo que puede parecer muy complejo, miles de portadoras a transportar de modo simultáneo, se reduce de modo sencillo, puesto que estas operaciones son equivalentes a las que se realizan en el cálculo de la FFT de una señal. De esta manera, se pueden fabricar en serie circuitos integrados sencillos para demoduladores OFDM.

## *4.3 COFDM.*

En un sistema de telecomunicaciones, es preciso determinar los efectos introducidos por el canal. En este caso, OFDM está especialmente indicado para canales selectivos en frecuencia. Por tanto, esto va a dar lugar a que cada una de las portadoras tenga una SNR diferente, que da lugar a su vez a una BER diferente en cada caso. El cálculo de la BER total, se hace como una media estadística.

Pero, observando un canal selectivo, es fácil darse cuenta de que, para un número razonable de portadoras, el problema principal es el ruido a 0db. Esto lleva a la conclusión de que OFDM solamente no es suficiente, se necesita un codificador de canal. Esta incorporación, es lo que da lugar a los "*sistemas COFDM*" (Coded OFDM).

Para sistemas COFDM, en recepción, una decodificación con "hard-decison" no es suficiente, se necesita "soft-decision". Se utilizan codificadores convolucionales en transmisión, y en decodificación el Algoritmo de Viterbi. Además de técnicas de codificación de bloque, en este caso Reed-Solomon. Estas dos técnicas concatenadas dan lugar a un código robusto ante cualquier tipo de canal sin memoria.

Como en el sistema objeto de estudio uno de los parámetros principales de simulación es precisamente esto, el número de bits para "soft-decision", es necesario la introducción de estos conceptos. Se va a exponer mediante un ejemplo sencillo.

Se considera una señal con dos niveles $\pm$ 1V, y un solo bit transmitido por símbolo, de este modo, el '0' corresponde a -1V, y el '1' al nivel de tensión +1V. Estos niveles son enviados por el canal, que introduce ruido en la señal, y en recepción, es donde existen varios modos de operar para recuperar el bit transmitido de forma correcta.

El modo más sencillo, consiste en considerar los valores recibidos positivos como '1', y los negativos como '0', a este modo de operación se le denomina "hard-decision", y sólo es necesario un bit, el de signo, para almacenar el valor recibido.

Pero, hay una pega, y es que puede ser que el ruido introducido por el canal sea muy alto, y el valor -1V se haya convertido en 0.1V (hay que tener en cuenta que el ruido que introduce el canal es aleatorio), con lo cual se produce un error al demodular. Por tanto, se necesita un parámetro que sea capaz de medir el nivel de confianza que se tiene en que el valor recibido sea uno u otro. Por ello, se introduce el segundo modo de operación, denominado "soft-decision". Consiste en dividir la región de decisión en varias regiones, según el número de bits asignados en cada caso, mientras más bits se tengan, más confianza se tiene en uno u otro. Como se ha podido observar, "hard-decision" es un caso particular de "soft-decision" con el número de bits asignados igual a uno. Este concepto se explota en el decodificador de Viterbi, que realiza la decodificación del bit más probable basándose en "soft-decision" y en la cadena de datos recibida.

Esto indica que "soft-decision" es un factor muy importante a tener en cuenta, y se toman los datos afectados por SNR mayores como más fiables que aquellos que han sido afectados por SNR bajas. Esta información que se aporta a priori, es denominada "*Channel-State Information*" (CSI), y es clave en este tipo de sistemas.

## *4.4 RESUMEN.*

Para dar por finalizado este capítulo, se va a hacer un breve resumen en el que se exponen las ideas principales del sistema presentado en la norma para DVB-T, basado en sistemas COFDM.

COFDM es un sistema que se adapta especialmente bien para trabajar con canales selectivos en frecuencia. La clave de estos sistemas radica en la codificación utilizada sobre el sistema subyacente OFDM, además del uso de CSI, que permite un método distinto de regeneración adecuado a cada una de las portadoras, complementado con el modo de operación "soft-decision" utilizado por el decodificador de Viterbi.

El énfasis puesto en este capitulo radica en que este proyecto es un estudio sobre un sistema COFDM, en el que se miran las prestaciones de un decodificador de Viterbi variando sus parámetros propios de funcionamiento así como la codificación de datos y el mapeo de los mismos.

En las simulaciones que se van a realizar no se tendrán en cuenta canales selectivos ni multitrayecto puesto que no va a aparecer la codificación Reed-Solomon.

# 5 PRUEBAS DE MODULACIÓN DIGITAL.

## *5.1 PRUEBAS Y SIMULACIONES.*

### 5.1.1 PRUEBA TEST_0.

Esta es una prueba de validación de código, tomando los datos entre mapeador y demapeador, teniendo en cuenta el canal de ruido gaussiano (AWGN). Vemos su forma esquemática en la Figura 32.

Figura 32: test_0: Esquema de la prueba

En la realización de esta prueba se tienen en cuenta las constelaciones: 'QPSK', '16-QAM' y '64-QAM', así como un amplio rango de SNR, desde 0db a 30db, que nos permite observar la BER en un amplio rango de señal. En la Figura 33 observamos los resultados para cada una de las constelaciones por separado.

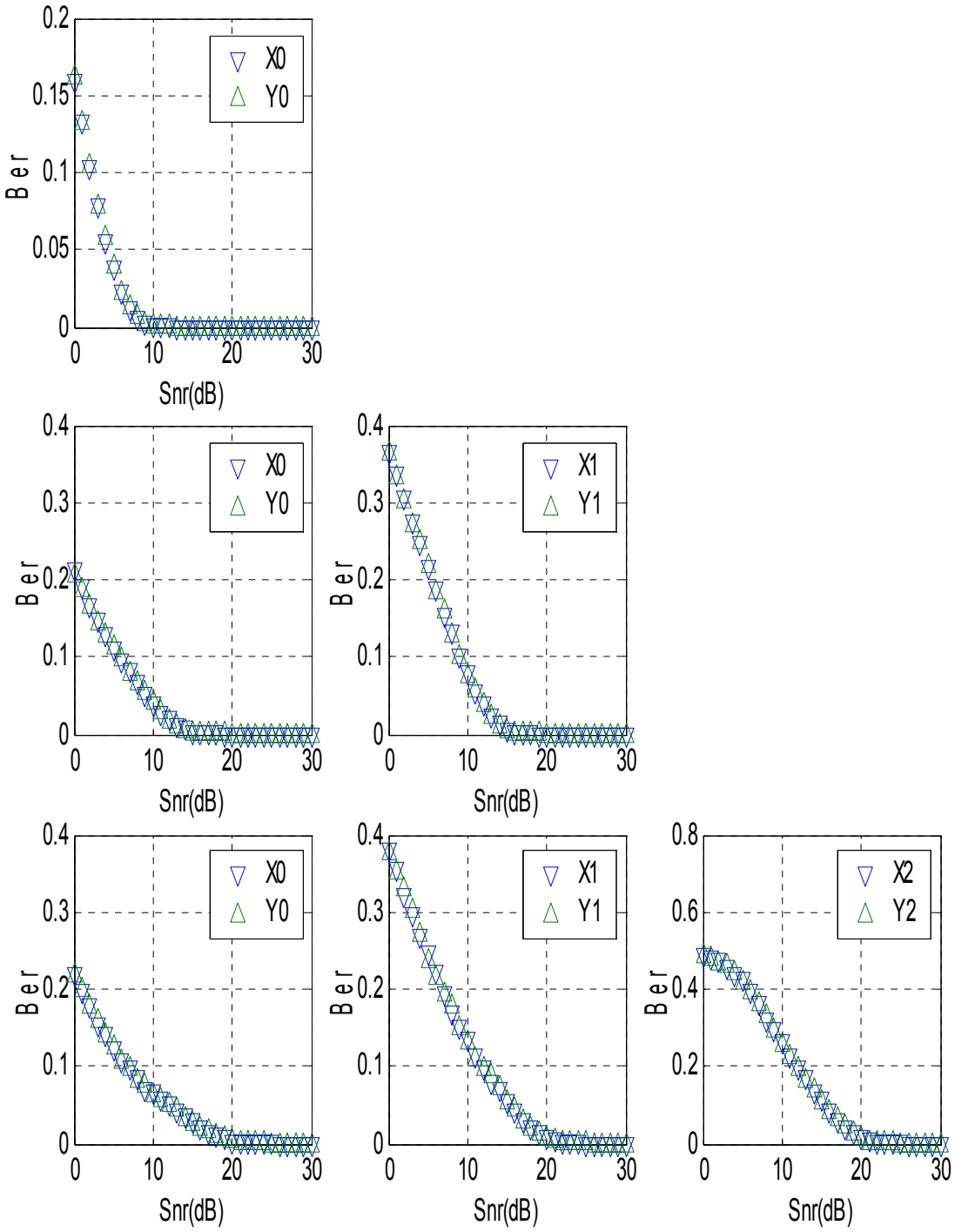



Figura 33: test_0: BER para las tres modulaciones en estudio según parejas de bits.

Como podemos observar de las gráficas, la probabilidad de error en los dos primeros bits de 'QPSK', es mucho menor que en los casos de '16-QAM' Y '64-QAM'. Esto se debe a que la distancia desde los puntos a la región de decisión, en media, es mayor en 'QPSK' que en las otras modulaciones. El mismo efecto se produce entre '16-QAM' y '64-QAM', aunque en menor medida, ya que estas distancias, en media, son más parecidas. Esto se pone de manifiesto en la siguiente gráfica, Figura 34, en la que las distancias medias aparecen representadas por una flecha de color verde y ordenadas de modo descendente según la modulación correspondiente.

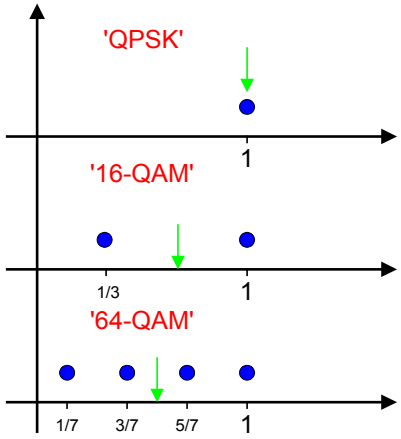



Figura 34: Comparativa de distancias para las distintas modulaciones.

Si miramos la probabilidad conjunta de error, esto es, para todos los bits de la modulación, vemos cómo sube a medida que se aumenta el número de puntos, consecuencia directa del efecto representado en la gráfica anterior. Esto se observa en la Figura 35.

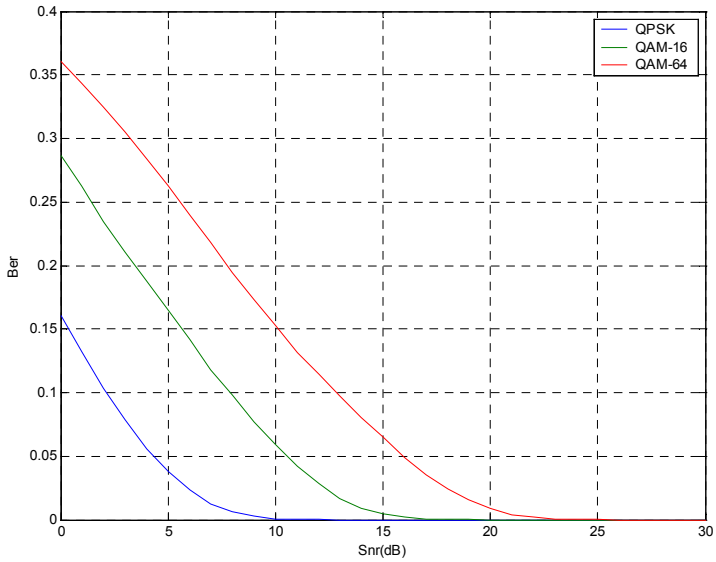



Figura 35: test_0: BER conjunta para las tres modulaciones.

A la vista de estos resultados, se ha podido concluir que en principio no existen erratas de código, hasta este punto de simulación.

# 6 CODIFICADOR CONVOLUCIONAL Y DECODIFICADOR DE VITERBI.

El propósito principal de este capítulo es explicar con detenimiento la técnica FEC, conocida como codificación convolucional y decodificación de Viterbi, y, más particularmente, se enfocará al decodificador de Viterbi y sus prestaciones, objeto principal de estudio de este proyecto.

Una vez terminada la introducción teórica del apartado, seguirán una serie de pruebas y resultados que van a ser discutidos en un capítulo posterior denominado **Conclusiones**.

## *6.1 INTRODUCCIÓN TEÓRICA.*

Las técnicas FEC se emplean para mejorar la capacidad del canal, añadiendo una serie de redundancias en los datos a transmitir a través de él. Este proceso mediante el cual se añade redundancia es llamado "codificación de canal", y se ha estudiado en el Capítulo 3.

Se va a dar una explicación detallada de la generación de los datos a transmitir por un canal mediante codificación convolucional, y, finalmente la recuperación de la cadena de datos inicial mediante la decodificación de Viterbi.

### 6.1.1 CODIFICADOR CONVOLUCIONAL.

Como recordatorio, podemos decir que un codificador convolucional es un generador de código en el cual la secuencia de información va pasando a través de un registro de estados, tal y como vemos de nuevo en la figura.

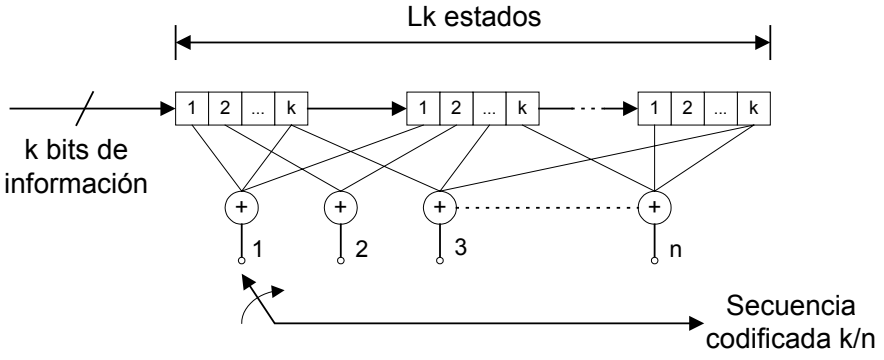



Figura 36: Codificador convolucional.

Para hacer la descripción de un codificador convolucional, tenemos varios métodos. El primero de ellos consiste en dar su matriz generadora de código. El problema de esto está en que el generador de la matriz de código se hace semi-infinito si la secuencia de datos de entrada también lo es. Es un método poco viable.

En general, los códigos convolucionales son descritos usando dos parámetros principales, además de los polinomios generadores. El primero de ellos, se denomina tasa del código, 'k/n', donde 'k' es el número de bits a la entrada y 'n' el número de bits a la salida. En segundo lugar, tenemos K, denominado "*constraint-length*" y denota el número de biestables necesarios por cada entrada, así como el número mínimo de bits necesarios para conseguir la primera salida. Los polinomios generadores son un conjunto de vectores que denotan las conexiones del codificador con cada salida. Se suelen dar en octal en vez de en binario porque aporta claridad.

Existen otros tres modos de representación: "Tree Diagram" (Diagrama de árbol), "Trellis Diagram" (diagrama enrejado), y "Diagrama de Estados". Se explican con un ejemplo que aportará claridad:

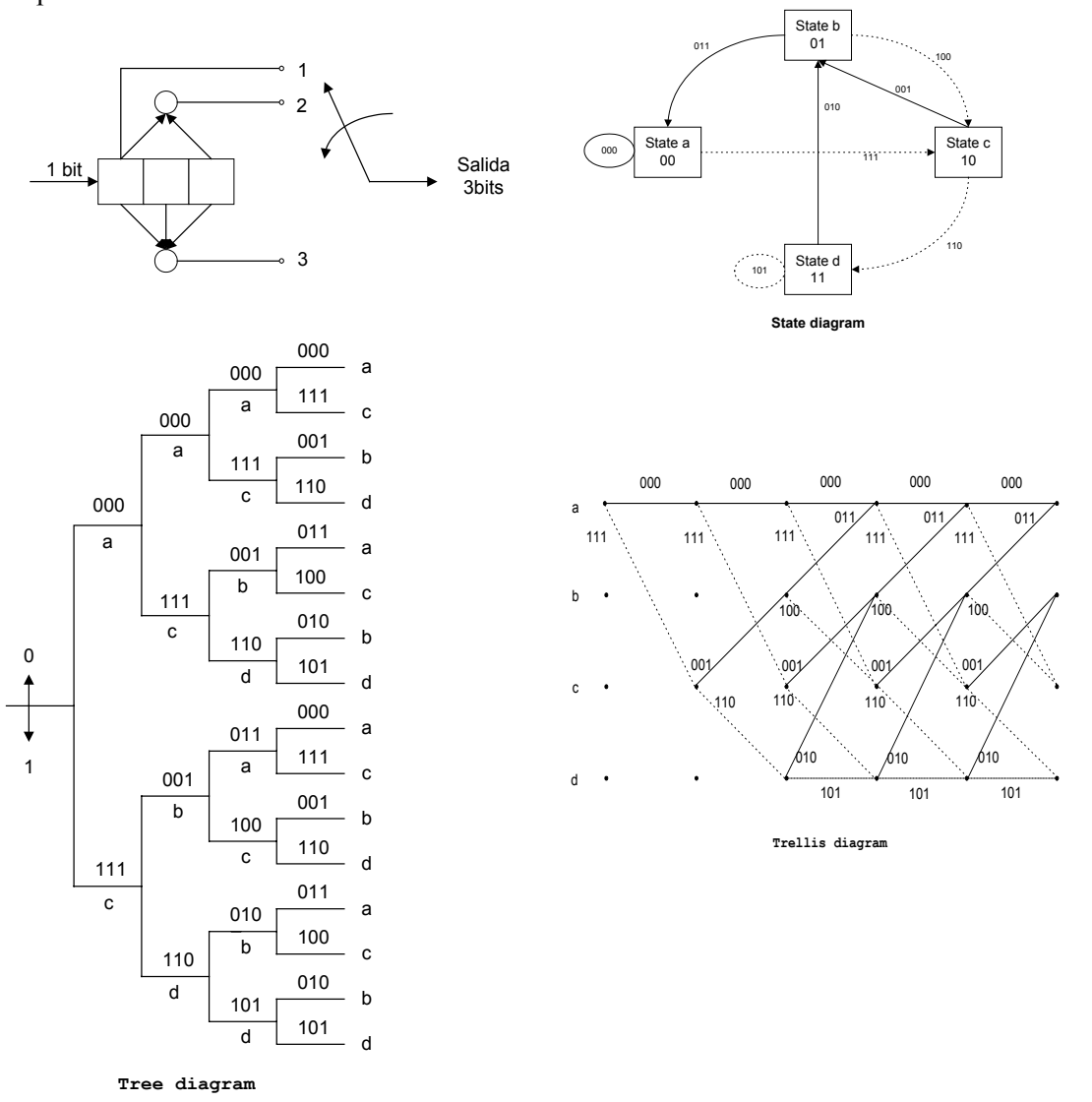



Figura 37: Distintas representaciones para el código 1/3 con K=3.

Una ventaja importante en codificación convolucional es que se puede demostrar, que la codificación convolucional de cualquier tasa 'k/n', se puede conseguir a partir del codificador '½', mediante el proceso de "*puncturing*" (punctureado) de los datos, en el cual, selectivamente, se van eliminando los datos que no deben ser transmitidos. Esto significa que se pueden conseguir todas las tasas de transmisión con distintos niveles de redundancia a partir de un solo codificador convolucional.

Según la norma para DVB-T [8], nos basamos en un código convolucional "madre", con una tasa '½' y 64 estados, a partir del cual podemos generar todos los demás. La matriz generadora de código, en este caso, cuenta con dos con dos polinomios, dados por los vectores G1=171oct y G2=133oct. De este modo se consiguen los distintos códigos convolucionales a través del "*puncturing*" de los datos de salida. [8]

### 6.1.2 EL ALGORITMO DE VITERBI.

#### 6.1.2.1 Introducción a los códigos FEC.

Para poder entender los principios de funcionamiento del algoritmo de Viterbi y comprobar su potencia en decodificación, es necesaria una explicación exhaustiva de lo que son las técnicas FEC y en que basan su funcionamiento.

Para afrontar este apartado, hacemos unas consideraciones previas de notación, que nos van a ayudar a seguir el desarrollo:

- $\{a_i\}$ Secuencia de datos de entrada, generalmente binaria (bits).
- $\{x_i\}$ Secuencia de símbolos tras el codificador convolucional. Tiene el mismo conjunto de valores permitidos que $\{a_i\}$, si esta última es binaria, entonces $\{x_i\}$ también lo es.
- $\{y_i\}$ Secuencia después del canal. Son los símbolos a decodificar.

Para cualquier tipo de canal, se puede demostrar que si todas las secuencias de entrada son equiprobables, entonces el decodificador que minimiza la probabilidad de error es aquel basado en la comparación de las funciones de probabilidad condicionales, esto es en las funciones de verosimilitud:

$$p\left(y / x^{(m)}\right)$$

donde '$y$' denota al símbolo recibido, y '$x^{(m)}$' son todas las posibles secuencias de símbolos transmitidos. Tras esta comparación, se decide a favor del máximo, esto es, a favor del símbolo más probable.

Si un algoritmo de decodificación sigue este método, entonces tenemos un decodificador de máxima verosimilitud (ML). Además como los datos que se van a recibir son aleatorios y no se conocen a priori lo que se hace es una predicción FEC.

Para poder comparar estas funciones ML, se necesita conocer las especificaciones del canal, para poder saber la pdf a la que responden.

En muchas ocasiones, en vez de comparar las funciones de máxima verosimilitud, lo que se hace es comparar el logaritmo de estas. Esto es $\log\left( p\left(y / x^{(m)}\right) \right)$, ya que el resultado resulta invariante.

Ahora necesitamos una expresión general de esta función ML, para ello, hacemos una serie de consideraciones que permiten trabajar con funciones estadísticas de forma cómoda.

- La probabilidad de que el canal altere un símbolo determinado viene dada por p< ½ .
- La secuencia recibida ' $y$ ' es una secuencia binaria N-dimensional.
- Todas las posibles secuencias transmitidas ' $x^{(m)}$ ' son también secuencias binarias N-dimensionales.
- Ambas secuencias, ' $x^{(m)}$ ' e ' $y$ ', difieren entre sí en ' $d_m$ ' símbolos. ( $d_m$ es la distancia Hamming entre ambas).

Con estas consideraciones, teniendo en cuenta un canal sin memoria, la probabilidad de que el símbolo ' $x^{(m)}$ ' se haya transformado en ' $y$ ' viene dada por:

$$p\left(y / x^{(m)}\right) = p^{d_m}.(1-p)^{N-d_m}$$

Si tomamos logaritmos en la expresión anterior, vemos que es equivalente a:

$$\log\left(\ p\left(y / x^{(m)}\right)\ \right) = -d_m.\log\left(1-p \middle/ p\right) + N.\log(1-p) = -\alpha.d_m - \beta \cong -d_m$$

donde $\alpha, \beta$ son constantes.

Por tanto, vemos cómo maximizar la función ML, equivale a minimizar la distancia Hamming, esto es precisamente lo que hacen los decodificadores con mínima probabilidad de error.

Para hacer la comparación de cada uno de los símbolos de la secuencia transmitida, y conseguir la distancia Hamming:

$$\log\left(\ p\left(y / x^{(m)}\right)\ \right) = \sum_{i}^{m}\sum_{j}^{N} \log\left(p\left(y_j / x_{ij}^{(m)}\right)\right)$$

donde '*i*' representa todos los posibles símbolos transmitidos, y '*j*' el número de símbolos.

A partir de esta función se consiguen los distintos pesos que van a servir en el cálculo de distancias mínimas para decodificar.

Con esto, se terminan las consideraciones generales para cualquier algoritmo de decodificación basado en FEC, el siguiente apartado se dedica a la explicación teórica del algoritmo objeto de estudio, el algoritmo de Viterbi.

### 6.1.2.2 Algoritmo de Viterbi.

El algoritmo de Viterbi es una técnica de ML para la decodificación de códigos convolucionales, con especial relevancia en canales sin memoria, que cuenta con una ventaja principal frente a otros algoritmos de decodificación secuenciales, también basados en técnicas FEC, ésta es su tiempo fijo de decodificación, factor clave para la realización hardware.

El decodificador de Viterbi fue desarrollado por Andrew J. Viterbi, uno de los miembros fundadores de la "Corporación Qualcomm". Su seminario sobre esta técnica se recoge en el artículo "Error bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm", publicado en "IEEE Transactions on Information Theory", volumen IT-13, en abril de 1967.

Desde entonces, otros investigadores han seguido sus pasos, aportando mejores codificadores convolucionales para los datos y explorando los límites de esta técnica variando los parámetros de diseño, de este modo se pretende encontrar una realización óptima tanto software como hardware.

Para poder afrontar la descripción de este algoritmo, son necesarias unas definiciones previas:

- *"Survivor-path (SP)"*: Es la secuencia de símbolos que nos da el camino a seguir.
- *"Path-metric (PM)"*: Coste del camino que se sigue.
- *"Branch-metric (BM)"*: Coste del camino desde el estado i al estado j con el siguiente símbolo recibido.
- *"Trace-back (TB)"*: Estado a partir del cual empezamos a decodificar, se parte del nodo con un '*Path-metric'* menor.
- *"Trace-back-length(TBL)"*: Nº de símbolos recibidos a partir de los que se realiza la decodificación. Es un parámetro clave en las necesidades de memoria.

Vemos el esquema seguido por el algoritmo de Viterbi en la gráfica de la siguiente figura:

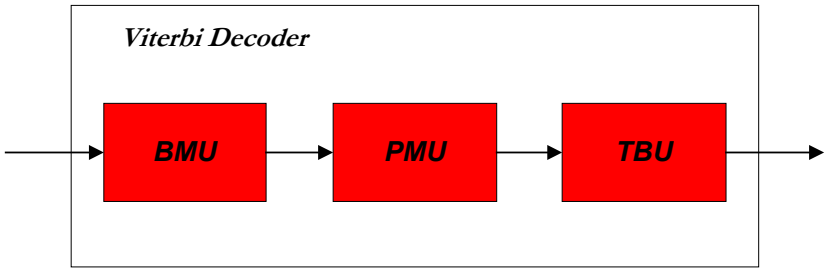



Figura 38: Bloques funcionales para el desarrollo de un Decodificador de Viterbi.

Como podemos observar se divide en tres pasos fundamentalmente. En estos pasos vamos a dividir su estudio.

- **Paso 1:** *Generación de la BM.*

Esta generación es básicamente el cálculo de las distancias que proporcionan el coste, o lo que es lo mismo, la probabilidad de que el símbolo recibido coincida con el que se está comparando. Cuanto más pequeña sea esta distancia entre símbolos, más probabilidad hay de que estos símbolos sean el mismo.

El cálculo de la BM responde a la siguiente fórmula general, que atiende a los distintos tipos de métricas (ver Anexo1):

$$B_{i,j,n} = (y_n - C_{i,j})$$

donde $C_{i,j}$ es el coste del camino del estado i al j (i= estado inicial, j= estado al que se dirige y n= nº de recursión en la que se encuentra). En el cálculo de estos costes $C_{i,j}$ es donde interviene la probabilidad condicional de ML. Este coste depende también de la probabilidad a priori de un determinado símbolo ya que estamos dentro de las técnicas FEC.

De este modo se generan todas las BM posibles desde todos los posibles estados hacia todas sus posibles transiciones, en cada una de las recursiones, hasta TBL+1.

- **Paso 2:** *Recálculo del SP y de las PM para todos los estados.*

En cada uno de los estados vamos calculando los Mi,n ("*Path-metric*"), y se van descartando los caminos de mayor coste. Como es de suponer, en el cálculo de las PM intervienen las BM calculadas en el paso anterior, y la PM de los estados anteriores. Su formulación matemática se corresponde con:

$$M_{i,n} = \min_j \left[ M_{j,n-1} + B_{j,i} \right]$$

En el caso de encontrarnos en el estado de partida, todas las PM de los estados son nulas. Esto es $M_{i,0} = 0$.

El cálculo de las PM lleva consigo la elección de distintos caminos con menor coste, se van descartando caminos. Esto procedimiento es denominado '*recálculo del SP'*.

- **Paso 3:** *Selección del camino más probable.*

Este último paso se realiza cuando el número de iteraciones de los pasos anteriores ha llegado a la longitud del TBL. Es ahora cuando se da la primera secuencia de datos decodificados.

A partir del nodo a partir del cual hacemos el TB, se toma el estado con SP menor y vamos hacia atrás según el camino con menores SP almacenados. De este modo, encontramos la traza más probable. Si partimos desde un nodo con varios estados de igual SP almacenado, se sigue hacia atrás con el mismo criterio, porque puede demostrarse que ambos caminos convergen hacia el mismo punto.

Se va a utilizar un método que mide la distancia mínima entre estados consecutivos. De esta manera se va consiguiendo un camino a través del "Trellis" con mínimo número de errores. Conceptualmente, el uso del mismo "Trellis" tanto en codificación como en decodificación nos lleva a una mínima probabilidad de error.

Una vez visto el procedimiento utilizado por el algoritmo, podemos hacer una serie de consideraciones generales, que influyen en la realización física, en un circuito integrado del esquema.

Desde el punto de vista Hardware, tendremos tres bloques funcionales, encargados de realizar las tres funciones descritas en los pasos anteriores, tal y como se ha visto en el esquema inicial.

Los dos bloques que consumen un mayor número de recursos son el PMU y el TBU. La **lógica consumida** depende directamente del ancho de las métricas elegidas así como del número de estados. En función de los requisitos de tiempo para el procesamiento, se puede considerar una arquitectura de procesamiento repetitivo (secuencial) o de procesamiento simultáneo (paralelo). La primera requiere menos lógica, pero requiere un mayor tiempo de procesamiento por dato de entrada. La segunda requiere una mayor lógica y permite un menor tiempo de procesamiento.

La **memoria consumida** depende directamente del *Survivor Path*, que es función del número de estados del "Trellis" utilizado y por el TBL, que es el que obliga a tener un número mayor o menor de datos almacenados. Los requerimientos de memoria vienen dados tanto por el número de estados almacenados y por la longitud de cada una de las palabras almacenadas.

Para aclarar los conceptos vistos, en el siguiente apartado se presenta un caso práctico.

### 6.1.3 CASO PRÁCTICO.

Se va a desarrollar un ejemplo que se corresponde con los siguientes datos:

- Codificador convolucional utilizado: K=3, (7, 5). (Por tanto k/n=1/2)
- Con puncturing=1/2.
- En decodificación se va a tomar "hard-decision"
- TBL=15.
- Suponemos una secuencia de datos a la entrada de $\{0\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\}_2$

Para ver de modo más claro las especificaciones de este codificador, se dan unas tablas. En la primera se representa el estado al que se pasa cuando se recibe un determinado bit, y la segunda el símbolo de salida a partir de un determinado estado.

| | **Siguiente estado** | |
|---|---|---|
| **Estado actual** | **Entrada '0'** | **Entrada '1'** |
| **00** | 00 | 10 |
| **01** | 00 | 10 |
| **10** | 01 | 11 |
| **11** | 01 | 11 |

Tabla 1: Siguiente estado del codificador convolucional según la entrada sea '0' ó '1'.

| | Símbolo de salida | |
|---|---|---|
| **Estado actual** | **Entrada '0'** | **Entrada '1'** |
| **00** | 00 | 11 |
| **01** | 11 | 00 |
| **10** | 10 | 01 |
| **11** | 01 | 10 |

Tabla 2: Símbolo de salida según la entrada sea '0' ó '1'.

Para hacer más inteligible este apartado numérico, se ve a través de una serie de pasos que mejoran la comprensión de la secuencia ordenada a seguir.

**1.** Código tras el codificador convolucional. Acudimos a las tablas anteriores, a partir de ellas se obtiene:

| **Entrada** | **0** | **1** | **0** | **1** | **1** | **1** | **0** | **0** | **1** | **0** | **1** | **0** | **0** | **0** | **1** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Estado inicial** | 00 | 00 | 10 | 01 | 10 | 11 | 11 | 01 | 00 | 10 | 01 | 10 | 01 | 00 | 00 |
| **Estado final** | 00 | 10 | 01 | 10 | 11 | 11 | 01 | 00 | 10 | 01 | 10 | 01 | 00 | 00 | 10 |
| **Salida** | 00 | 11 | 10 | 00 | 01 | 10 | 01 | 11 | 11 | 10 | 00 | 10 | 11 | 00 | 11 |

Por tanto, la secuencia de salida nos viene dada por:

$$\{00\ 11\ 10\ 00\ 01\ 10\ 01\ 11\ 11\ 10\ 00\ 10\ 11\ 00\ 11\}_2$$

ésta es la secuencia que atravesará el canal.

**2.** Paso por el canal. Suponemos que la secuencia es ruidosa, se han insertado errores, de esta forma la secuencia queda como:

$$\{00\ \ 11\ \ 10\ \ 00\ \ \textcircled{01}\ \ 11\ \ 01\ \ 11\ \ 11\ \ 10\ \ \textcircled{01}\ \ 10\ \ 11\ \ 00\ \ 11\}_2$$

a partir de ella estudiaremos la capacidad correctora del algoritmo de Viterbi. (Los símbolos donde se han insertado los errores aparecen rodeados en rojo.)

**3.** Algoritmo de Viterbi. Para aportar claridad se van a detallar los tres pasos por separado.

- Paso1: '*Cálculo de las BM*'.

Responde a la distancia 'Hamming' (ver Anexo1) entre el símbolo recibido y el peso de cada uno de los estados, según el símbolo recibido. ($BM_{i,j}$ donde '*i*' denota el estado al que se llega, y '*j*' el estado de partida.).

En realidad, desde el punto de vista hardware, sólo existen cuatro BM, que son las cuatro distancias desde el punto recibido, hasta las salidas ideales del codificador convolucional. Por tanto, la tabla lo que nos da es dónde se aplican cada una de ellas. El estado de partida, y al que se llega.

La tabla que se obtiene viene dada por:

| **SÍMBOLO RECIBIDO** | | **00** | **11** | **10** | **00** | **01** | **11** | **01** | **11** | **11** | **10** | **01** | **10** | **11** | **00** | **11** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Estado 0 '00'** | $\mathbf{BM_{0,0}}$ | 0 | 2 | 1 | 0 | 1 | 2 | 1 | 2 | 2 | 1 | 1 | 1 | 2 | 0 | 2 |
| | $\mathbf{BM_{0,1}}$ | 2 | 0 | 1 | 2 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 2 | 0 |
| **Estado 1 '10'** | $\mathbf{BM_{1,2}}$ | 2 | 0 | 1 | 2 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 2 | 0 |
| | $\mathbf{BM_{1,3}}$ | 0 | 2 | 1 | 0 | 1 | 2 | 1 | 2 | 2 | 1 | 1 | 1 | 2 | 0 | 2 |
| **Estado 2 '01'** | $\mathbf{BM_{2,0}}$ | 1 | 1 | 0 | 1 | 2 | 1 | 2 | 1 | 1 | 0 | 2 | 0 | 1 | 1 | 1 |
| | $\mathbf{BM_{2,1}}$ | 1 | 1 | 2 | 1 | 0 | 1 | 0 | 1 | 1 | 2 | 0 | 2 | 1 | 1 | 1 |
| **Estado 3 '11'** | $\mathbf{BM_{3,2}}$ | 1 | 1 | 2 | 1 | 0 | 1 | 0 | 1 | 1 | 2 | 0 | 2 | 1 | 1 | 1 |
| | $\mathbf{BM_{3,3}}$ | 1 | 1 | 0 | 1 | 2 | 1 | 2 | 1 | 1 | 0 | 2 | 0 | 1 | 1 | 1 |

Tabla 3: 'Branch-metric'.

- Paso 2: '*Cálculo de los PM y SP*'.

Como ya se ha visto en el apartado anterior, el SP es el brazo que llega a un estado con menor peso. En la siguiente tabla se representa el PM de cada estado y el estado de partida, esto es: 'PM / SP'.

| **SÍMBOLO RECIBIDO** | **00** | **11** | **10** | **00** | **01** | **11** | **01** | **11** | **11** | **10** | **01** | **10** | **11** | **00** | **11** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **PM Estado 0 '00' / SP** | 0/0 | 1/2 | 1/2 | 1/0 | 2/0 | 1/2 | 2/2 | 1/2 | 3/0 | 4/0 | 2/2 | 3/0 | 2/2 | 2/0 | 4/0 |
| **PM Estado 1 '10' / SP** | 0/2 | 0/0 | 0/2 | 0/2 | 2/0 | 2/0 | 2/2 | 2/0 | 1/0 | 4/2 | 2/2 | 3/0 | 3/0 | 4/2 | 2/0 |
| **PM Estado 2 '01' / SP** | 1/1 | 0/1 | 0/1 | 1/3 | 1/3 | 1/3 | 1/3 | 3/1 | 3/1 | 1/1 | 3/3 | 2/1 | 4/1 | 4/1 | 5/3 |
| **PM Estado 3 '11' / SP** | 1/1 | 0/1 | 0/3 | 1/1 | 0/1 | 1/3 | 2/1 | 3/3 | 3/1 | 3/3 | 4/1 | 4/1 | 4/1 | 4/1 | 5/1 |

Tabla 4: "Path-metric" & "Survivor-path".

- Paso 3: '*Calculo del TB*'.

El cálculo del TB tiene dos partes, la primera es el cómputo de la distancia, ver cuál es la menor distancia en el estado de partida, y el segundo, que es ver el camino a tomar, a partir del SP.

A partir de la tabla anterior, se va tomando el camino con menor coste, es el que aparece en la siguiente tabla señalado en color.

| **SÍMBOLO RECIBIDO** | **00** | **11** | **10** | **00** | **01** | **11** | **01** | **11** | **11** | **10** | **01** | **10** | **11** | **00** | **11** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **PM Estado 0 '00' / SP** | 0/0 | 1/2 | 1/2 | 1/0 | 2/0 | 1/2 | 2/2 | 1/2 | 3/0 | 4/0 | 2/2 | 3/0 | 2/2 | 2/0 | 4/0 |
| **PM Estado 1 '10' / SP** | 0/2 | 0/0 | 0/2 | 0/2 | 2/0 | 2/0 | 2/2 | 2/0 | 1/0 | 4/2 | 2/2 | 3/0 | 3/0 | 4/2 | 2/0 |
| **PM Estado 2 '01' / SP** | 1/1 | 0/1 | 0/1 | 1/3 | 1/3 | 1/3 | 1/3 | 3/1 | 3/1 | 1/1 | 3/3 | 2/1 | 4/1 | 4/1 | 5/3 |
| **PM Estado 3 '11' / SP** | 1/1 | 0/1 | 0/3 | 1/1 | 0/1 | 1/3 | 2/1 | 3/3 | 3/1 | 3/3 | 4/1 | 4/1 | 4/1 | 4/1 | 5/1 |

Tabla 5: Tabla que recoge el *Trace-Back*.

Fijándonos en la tabla anterior, vemos que los estados más probables son:

$$\{0\ 1\ 2\ 1\ 3\ 3\ 2\ 0\ 1\ 2\ 1\ 2\ 0\ 0\ 1\}$$

si estos estados se ponen en binario dan lugar a la secuencia:

$$\{00\ 10\ 01\ 10\ 11\ 11\ 01\ 00\ 10\ 01\ 10\ 01\ 00\ 00\ 10\}_2$$

si volvemos a la Tabla 1 vemos que la secuencia decodificada a la que da lugar es:

$$\{0\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\}_2$$

Si comparamos esta secuencia con la inicial, antes del codificador convolucional vemos que coinciden. Por tanto, el proceso de codificación de canal seguido, ha corregido los errores introducidos por este.

## *6.2 PRUEBAS Y SIMULACIONES.*

Este apartado se divide en dos grupos principales, el primero es el de pruebas de código, en el que nos centramos en verificar el correcto funcionamiento del código de simulación. Y el segundo, el de 'pruebas de sistema', en el que se hacen las pruebas del sistema completo, con precisión en los resultados obtenidos.

### 6.2.1 PRUEBAS DE VERIFICACIÓN DE CÓDIGO.

#### 6.2.1.1 Prueba TEST_1.

Se valida el código cuando tenemos punctureado de datos. Se introducen dos bloques nuevos en el esquema anterior, el punctureado de los datos en el transmisor, y el depunctureado en el receptor. El esquema final para esta prueba quedaría del siguiente modo:

Figura 39: test_1: Esquema simulado.

Los resultados que se obtengan no deben variar respecto al esquema del "test_0" anterior puesto que para el cálculo de la BER no se tiene en cuenta los datos de puncturing.

Vemos si estos resultados esperados son los que se obtienen realmente, para ello nos apoyamos en las siguientes gráficas obtenidas de las simulaciones.

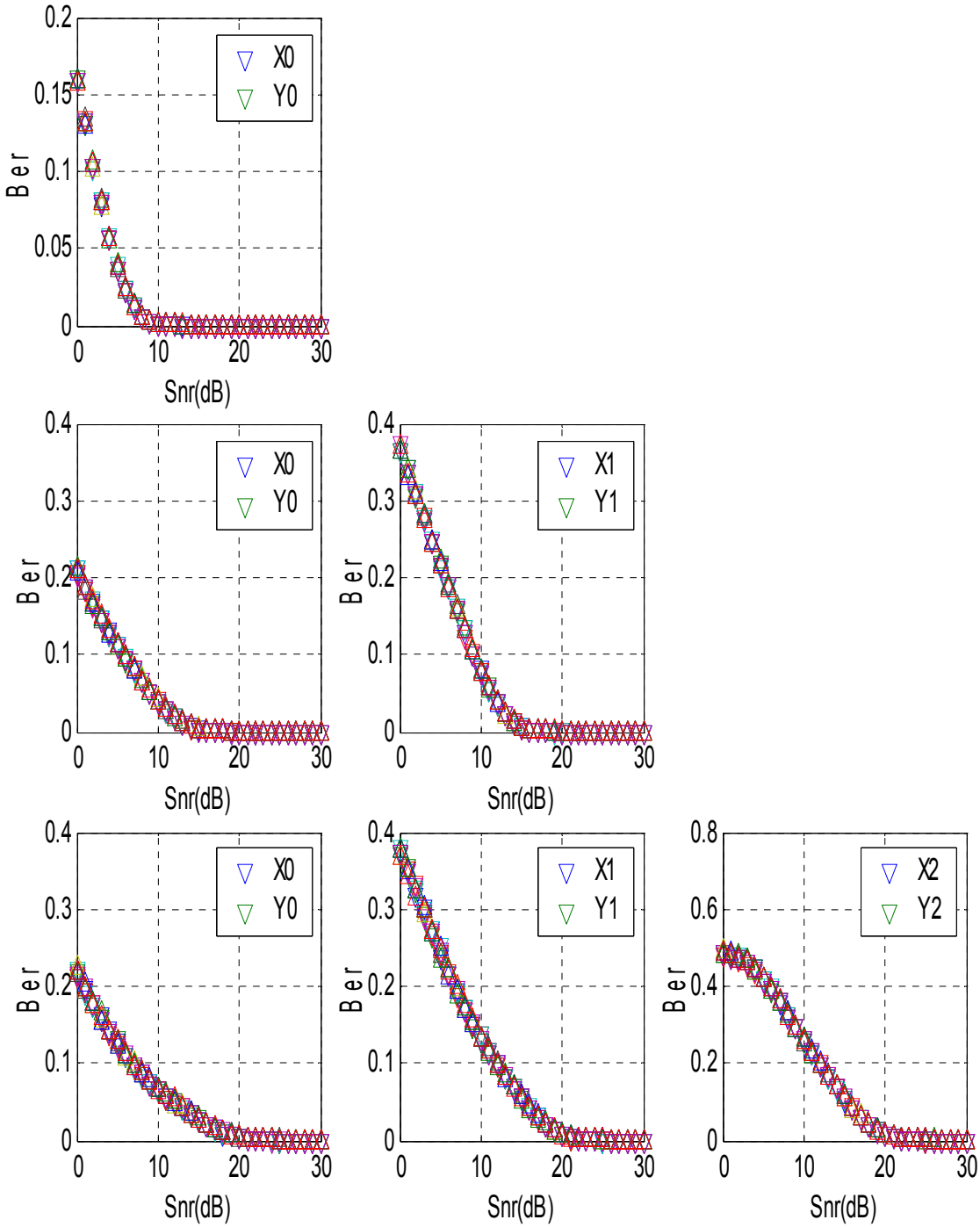



Figura 40: test_1: BER separada por parejas de bits.

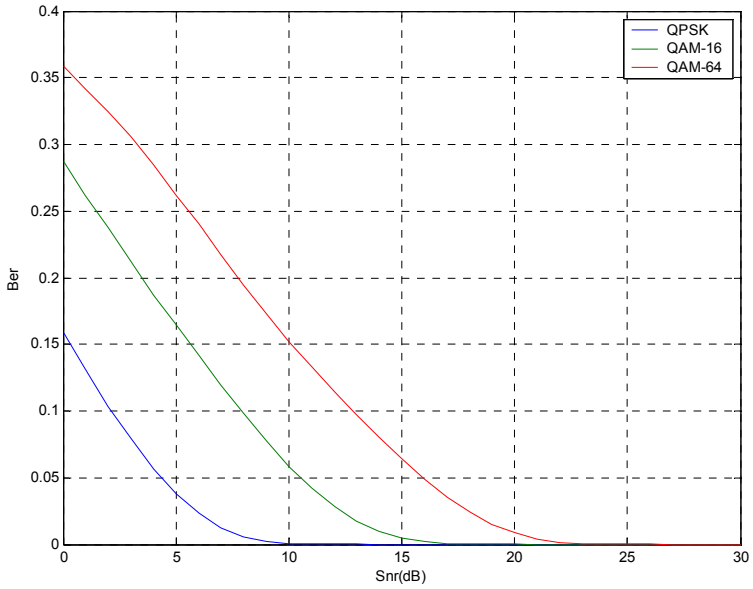



Figura 41 test_1: Probabilidad de error conjunta para cada una de las modulaciones empleadas.

Como podemos observar comparando los resultados de las Figura 40 y Figura 41, con las del caso anterior, *"test_0"* (Figura 33 y Figura 35) los datos obtenidos son los esperados, puesto que las probabilidades de error no varían. Esto es señal de que el código que se está utilizando sigue siendo correcto, o al menos no ofrece resultados erróneos visibles.

### 6.2.1.2 Prueba TEST_2.

Esta es la última prueba de validación, se introduce el esquema de simulación completo, con el codificador convolucional a la entrada, y el decodificador de Viterbi a la salida. Los datos se toman de una fuente aleatoria, antes del codificador. El esquema de simulación completo quedaría de la siguiente manera:

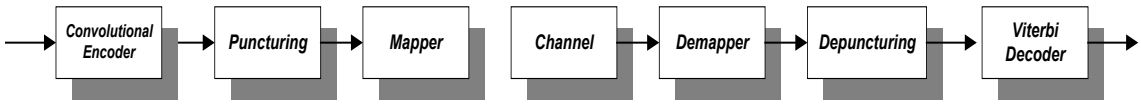



Figura 42. test_2: Esquema simulado.

Como podemos observar de las gráficas obtenidas, los resultados son muy diferentes a los obtenidos en los dos casos anteriores. Para bajas SNR la probabilidad de error es superior a la obtenida anteriormente (debido a que el algoritmo no funciona si el número de errores es excesivo), mejorando sensiblemente para SNR's medias y altas.

En las siguientes figuras vemos las gráficas obtenidas para las distintas modulaciones según los distintos patrones de puncturing. Se observa que a medida que disminuye la redundancia de los datos la probabilidad de error aumenta, pero que para SNR medias, mejora considerablemente en relación a los resultados obtenidos en casos anteriores.

En estas mismas gráficas (una para cada modulación, y dentro de cada una para las diferentes tasas de puncturing), se analiza asimismo la dependencia con el número de bits para *"soft-decision"*, que discutiremos en un apartado posterior.

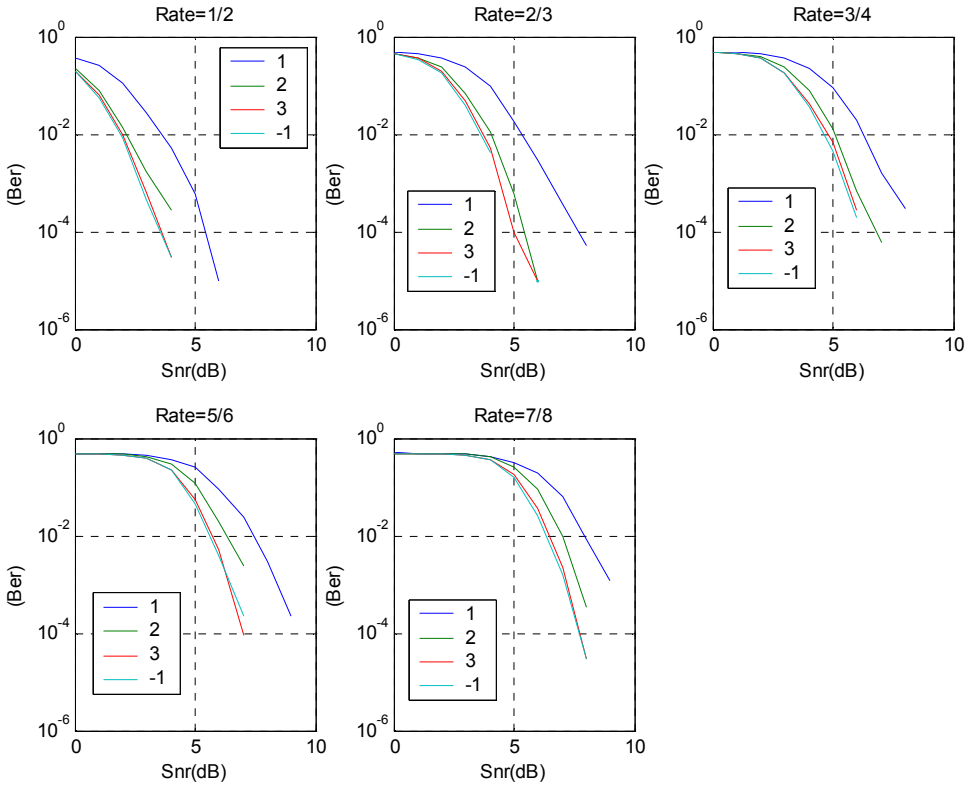



Figura 43. test_2: BER para una modulación QPSK según los distintos patrones de puncturing.

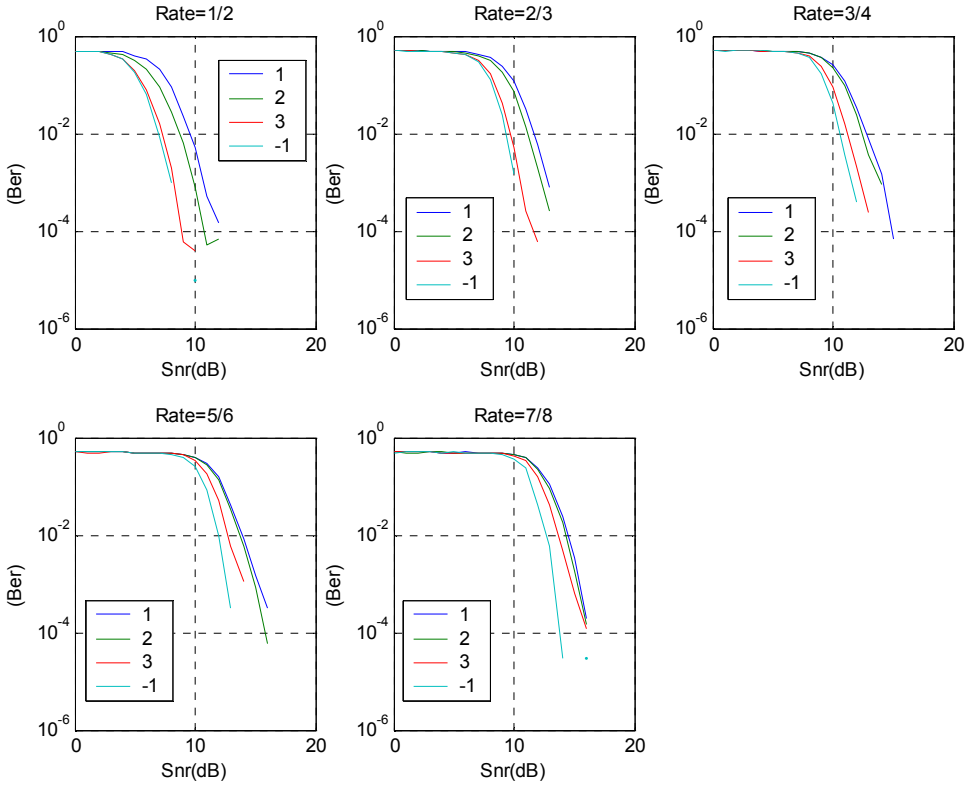



Figura 44 test_2: BER para una modulación 16-QAM según los distintos patrones de puncturing.

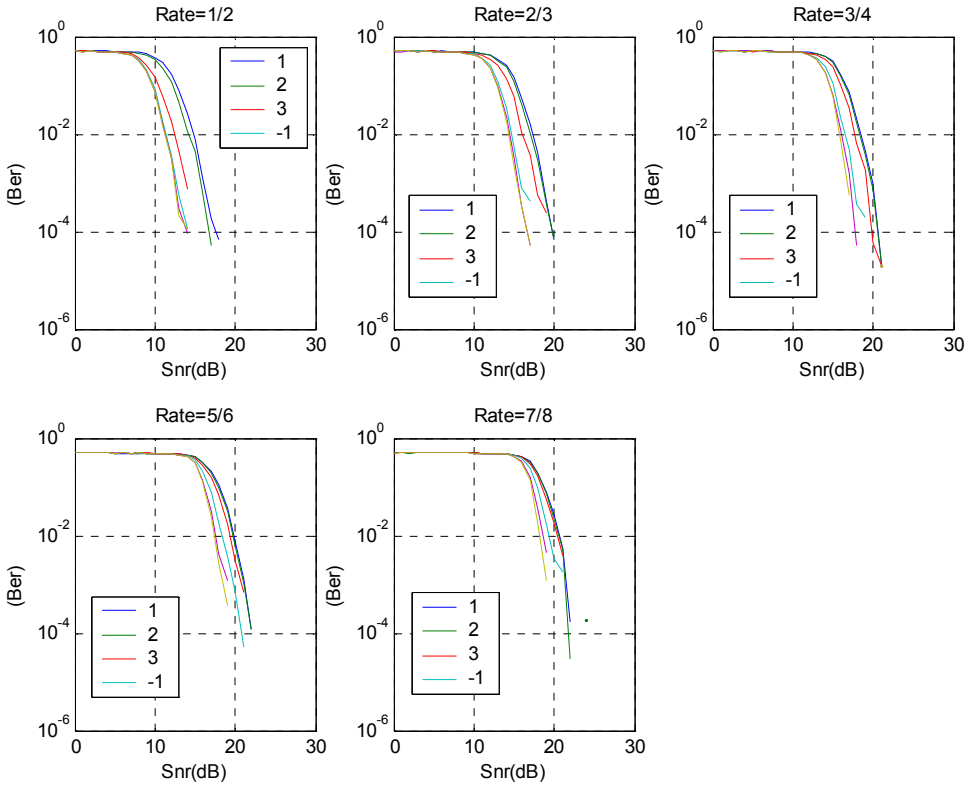



Figura 45 test_2: BER para una modulación 64-QAM según los distintos patrones de puncturing.

A la vista de estos resultados, podemos concluir que las pruebas de código son correctas y pasamos a la simulación del sistema.

### 6.2.2 PRUEBAS DE SIMULACIÓN DE SISTEMA.

Una vez hechas las comprobaciones pertinentes, pasamos a la simulación del sistema completo. Donde haremos especial hincapié en garantizar niveles de BER con suficiente precisión en el rango de 1e-4.

Se van a realizar dos pruebas. La primera, en la que vemos la BER del sistema según la modulación elegida, dividiendo los resultados según los patrones de puncturing y para distintos niveles de *"soft-decision"*. Es básicamente como la prueba anterior *"test_2"*, pero dotando a los datos de más precisión, en este resultado nos basamos para predecir el nivel de "*soft-decision*" necesario según cada modulación.

La segunda, en la que se va variando la longitud del *"trace-back"* en potencias de dos, tomando como nivel de "soft-decision" una precisión infinita. De aquí se predice la longitud TBL para el sistema.

A estas pruebas las vamos a denotar test_3 y test_4 siguiendo con la notación utilizada hasta ahora.

#### 6.2.2.1 Prueba TEST_3.

En esta prueba, el esquema de simulación utilizado es el mismo que en el caso anterior, para el "*test_2"*, dado en la Figura 42. Por tanto, en este caso, para ver los cambios respecto a la simulación anterior, se da el diagrama de flujo.

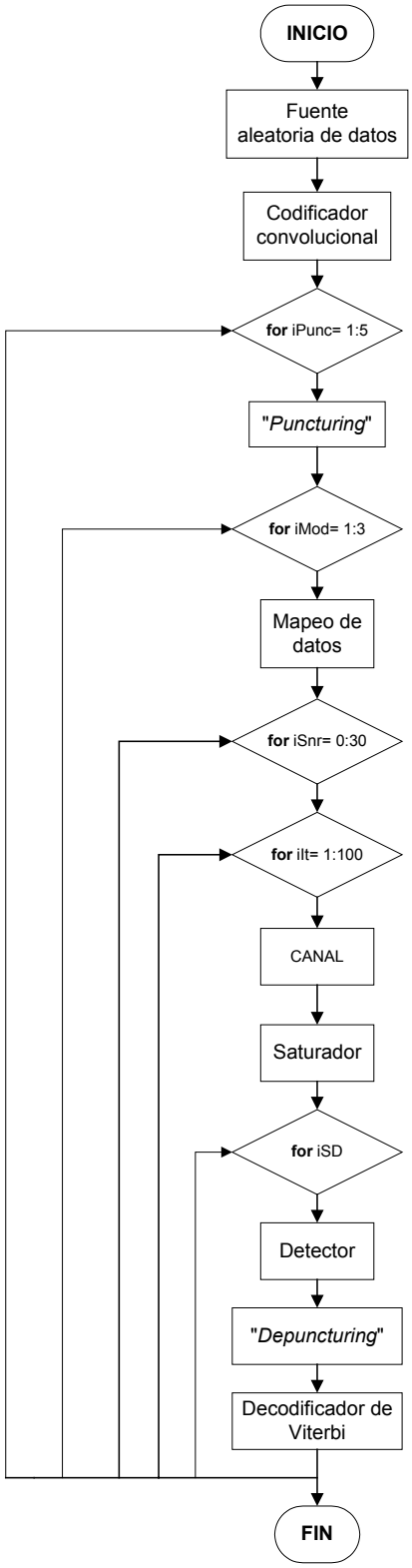



Figura 46 test_3: Diagrama de flujo.

Buscamos un valor para el número de bits en "soft-decision" que proporcione una BER correcta según la modulación.

Se trabaja con una precisión para la BER de 1e-7. En las zonas donde la probabilidad de error es nula (debido a que para la cantidad de datos procesada no se han obtenido errores), al ser el sistema de representación logarítmico, esta cantidad no aparece (-infinito).

En las siguientes figuras vemos los resultados obtenidos según las distintas modulaciones y los dos patrones de "puncturing" extremos, que nos van a servir para comparar resultados. En dichas gráficas hemos añadido el punto SNR/BER proporcionado por la norma DVB-T (para canal gausiano), que nos permite comprobar la veracidad de los resultados obtenidos.

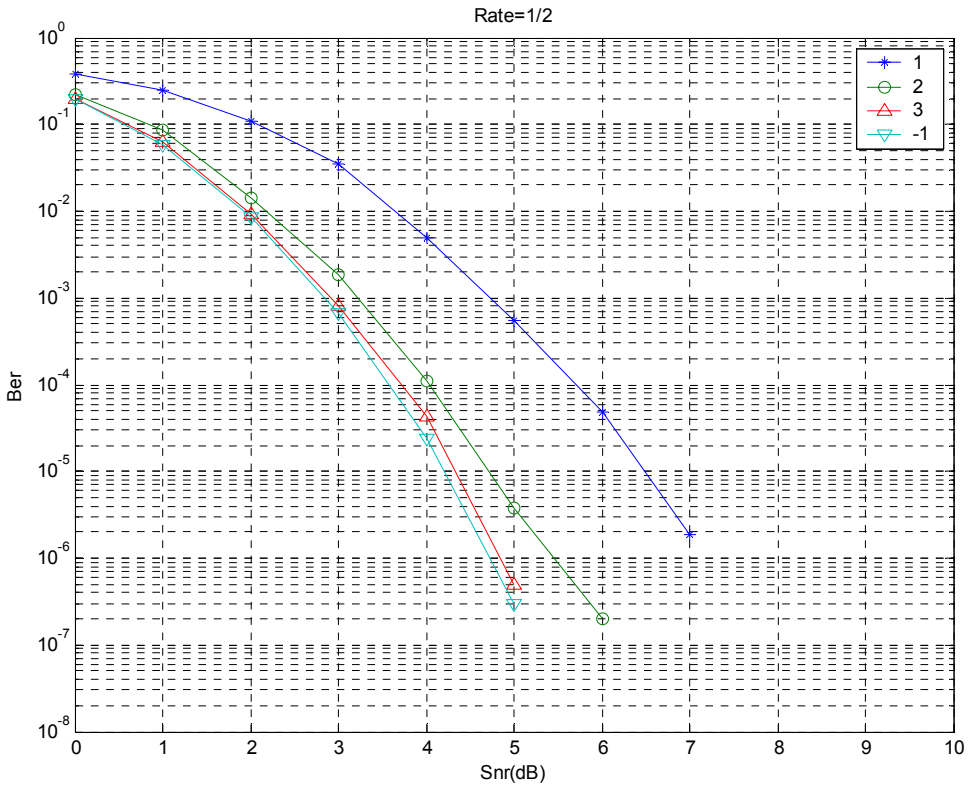



Figura 47 test_3: BER para distintos niveles de "*soft-decision*", modulación QPSK y tasa de "puncturing" ½ .

| NB_SD<br>SNR | 1 | 2 | 3 | -1 |
|---|---|---|---|---|
| **0** | 0.3811 | 0.2190 | 0.1934 | 0.1932 |
| **2** | 0.1072 | 0.0144 | 0.0092 | 0.0085 |
| **4** | 0.0050 | 1.111e-4 | 4.260e-5 | 2.350e-5 |
| **6** | 4.930e-5 | 2.000e-7 | - | - |
| **8** | 4.000e-8 | - | - | - |

Tabla 6: Tabla de resultados numéricos correspondiente a la Figura 47.

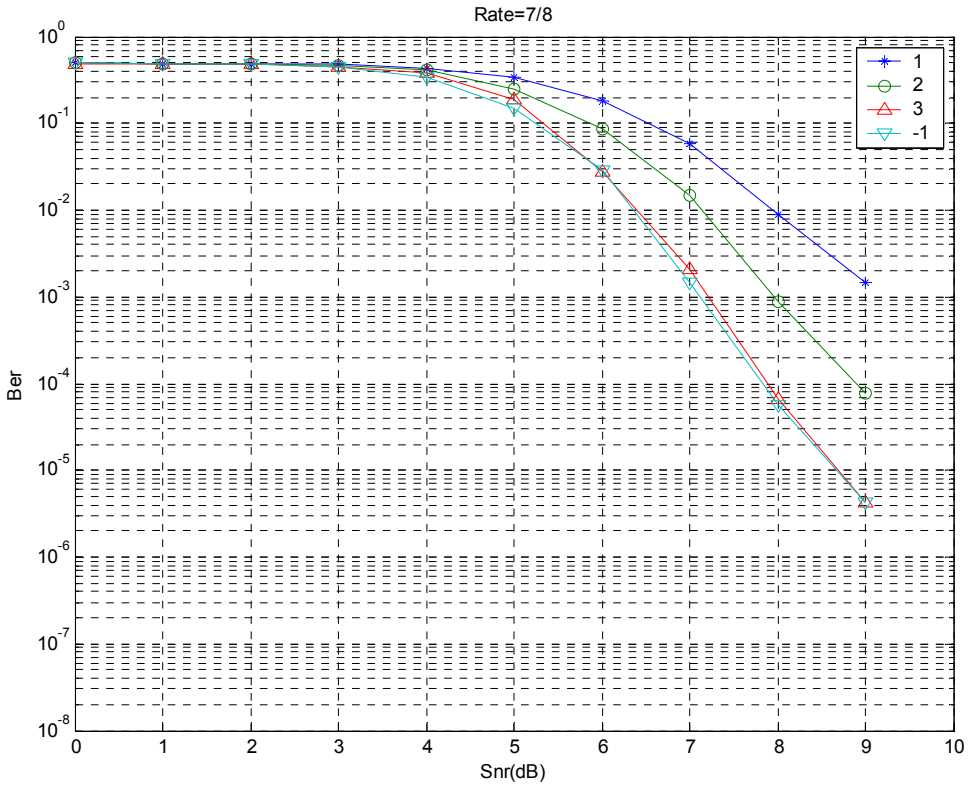



Figura 48: test_3: BER para distintos niveles de "*soft-decision*", modulación QPSK y tasa de "puncturing" 7/8.

| NB_SD<br>SNR | 1 | 2 | 3 | -1 |
|---|---|---|---|---|
| **0** | 0.4946 | 0.4946 | 0.4935 | 0.4935 |
| **2** | 0.4846 | 0.4846 | 0.4807 | 0.4800 |
| **4** | 0.4364 | 0.4060 | 0.3797 | 0.3436 |
| **6** | 0.1825 | 0.0866 | 0.0281 | 0.0281 |
| **8** | 0.0090 | 8.725e-4 | 6.640e-5 | 5.640e-5 |
| **10** | 2.003e-4 | 1.354e-5 | - | - |

Tabla 7: Tabla de resultados numéricos correspondiente a la Figura 48.

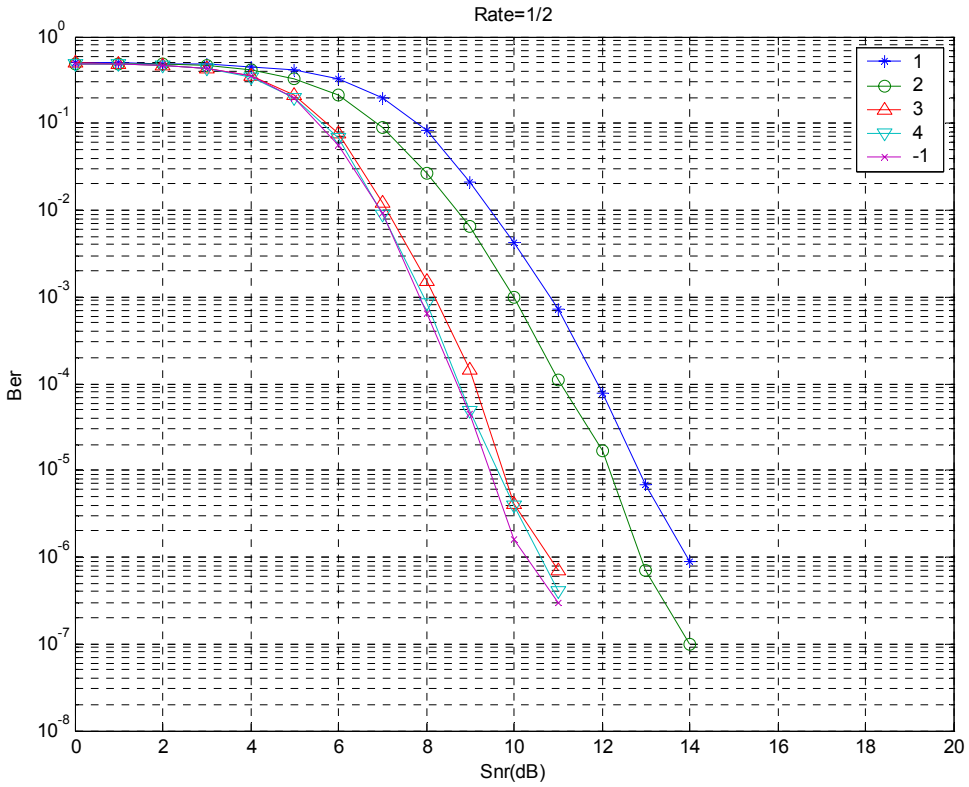



Figura 49: test_3: BER para distintos niveles de "*soft-decision*", modulación 16-QAM y tasa de "puncturing" ½.

| NB_SD<br>SNR | 1 | 2 | 3 | 4 | -1 |
|---|---|---|---|---|---|
| **4** | 0.4496 | 0.4078 | 0.3484 | 0.3419 | 0.3418 |
| **6** | 0.3220 | 0.2153 | 0.0761 | 0.0670 | 0.0562 |
| **8** | 0.0832 | 0.0266 | 0.0015 | 8.542e-4 | 6.431e-4 |
| **10** | 0.0042 | 0.0010 | 4.100e-6 | 3.900e-6 | 1.600e-6 |
| **12** | 7.640e-5 | 1.700e-5 | - | - | - |
| **14** | 9.000e-7 | 1.000e-7 | - | - | - |
| **16** | - | - | - | - | - |

Tabla 8: Tabla de resultados numéricos correspondiente a la Figura 49.

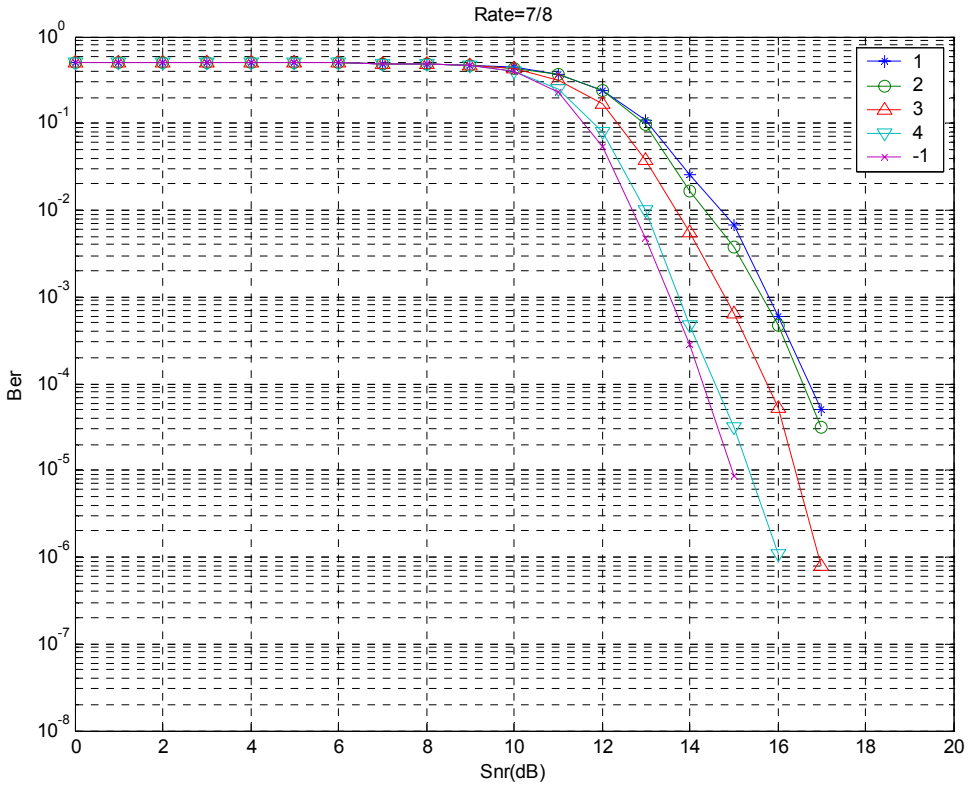



Figura 50: test_3: BER para distintos niveles de "*soft-decision*", modulación 16-QAM y tasa de "puncturing" 7/8.

| NB_SD<br>SNR | 1 | 2 | 3 | 4 | -1 |
|---|---|---|---|---|---|
| **4** | 0.4986 | 0.4986 | 0.4967 | 0.4967 | 0.4951 |
| **6** | 0.4963 | 0.4947 | 0.4947 | 0.4947 | 0.4947 |
| **8** | 0.4877 | 0.4868 | 0.4822 | 0.4797 | 0.4797 |
| **10** | 0.4411 | 0.4328 | 0.4294 | 0.4048 | 0.3932 |
| **12** | 0.2398 | 0.2415 | 0.1686 | 0.0805 | 0.0537 |
| **14** | 0.0255 | 0.0168 | 0.0055 | 4.623e-4 | 2.866e-4 |
| **16** | 6.012e-4 | 4.664e-4 | 5.180e-5 | 1.100e-6 | - |

Tabla 9: Tabla de resultados numéricos correspondiente a la Figura 50.

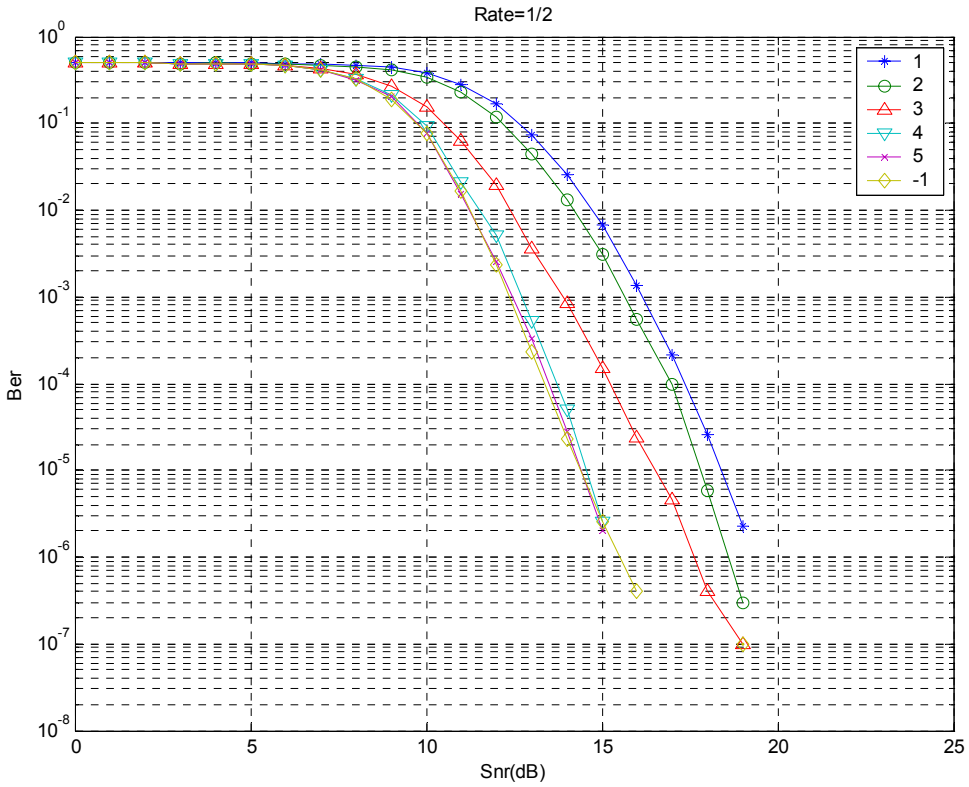



Figura 51: test_3: BER para distintos niveles de "soft-decision", modulación 64-QAM y tasa de "puncturing" ½.

| NB_SD<br>SNR | 1 | 2 | 3 | 4 | 5 | -1 |
|---|---|---|---|---|---|---|
| **5** | 0.4981 | 0.4938 | 0.4818 | 0.4794 | 0.4783 | 0.4780 |
| **7** | 0.4849 | 0.4731 | 0.4305 | 0.4097 | 0.4136 | 0.4099 |
| **9** | 0.4394 | 0.4117 | 0.2685 | 0.2139 | 0.2007 | 0.1895 |
| **11** | 0.2835 | 0.2281 | 0.0626 | 0.0214 | 0.0155 | 0.0155 |
| **13** | 0.0734 | 0.0440 | 0.0036 | 5.225e-4 | 3.345e-4 | 2.344e-4 |
| **15** | 0.0069 | 0.0031 | 1.521e-4 | 2.600e-6 | 2.000e-6 | 2.000e-6 |
| **17** | 2.110e-4 | 9.800e-5 | 4.600e-6 | - | - | - |

Tabla 10: Tabla de resultados numéricos correspondiente a la Figura 51.

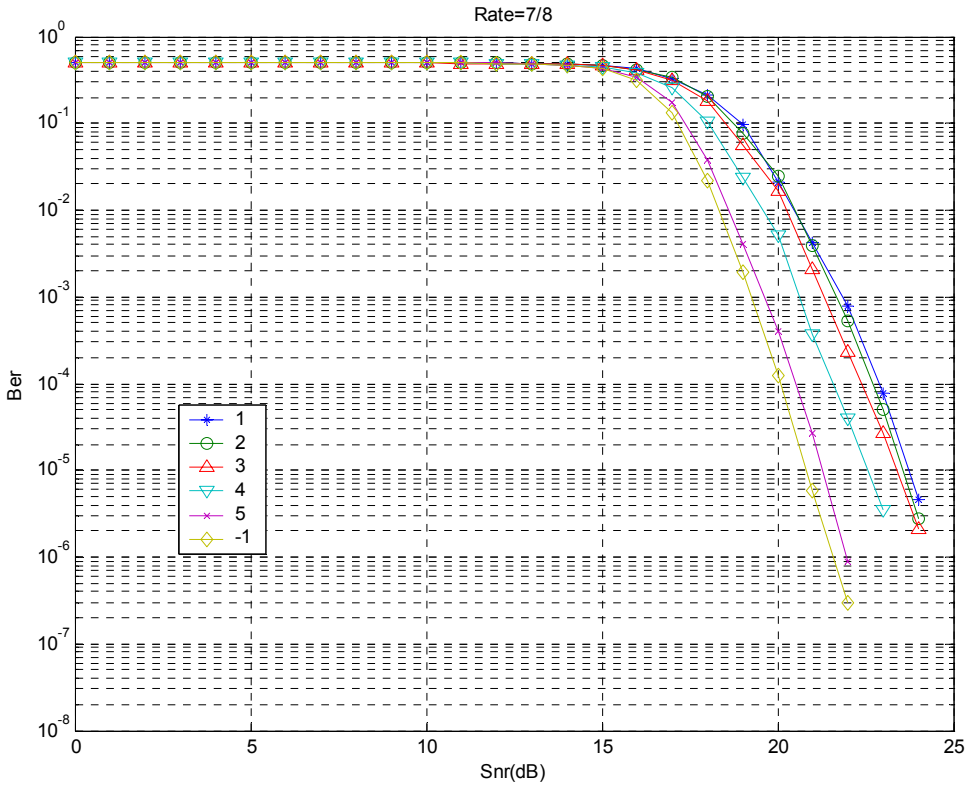



Figura 52: test_3: BER para distintos niveles de "soft-decision", modulación 64-QAM y tasa de "puncturing" 7/8.

| NB_SD<br>SNR | 1 | 2 | 3 | 4 | 5 | -1 |
|---|---|---|---|---|---|---|
| **5** | 0.4996 | 0.4985 | 0.4985 | 0.4985 | 0.4985 | 0.4985 |
| **10** | 0.4982 | 0.4978 | 0.4978 | 0.4968 | 0.4968 | 0.4968 |
| **15** | 0.4579 | 0.4579 | 0.4579 | 0.4503 | 0.4247 | 0.4242 |
| **17** | 0.3251 | 0.3251 | 0.3087 | 0.2562 | 0.1717 | 0.1318 |
| **19** | 0.0966 | 0.0769 | 0.0559 | 0.0242 | 0.0041 | 0.0019 |
| **21** | 0.0042 | 0.0039 | 0.0021 | 3.779e-4 | 2.710e-5 | 5.801e-6 |
| **23** | 7.721e-5 | 5.051e-5 | 2.660e-5 | 3.500e-6 | - | - |

Tabla 11: Tabla de resultados numéricos correspondiente a la Figura 52.

Una vez que tenemos los resultados de las simulaciones, se acude a la norma DVB-T [8] para ver cuales son los que producen una menor degradación de prestaciones, viendo por separado modulación y rate:

- Caso QPSK:

Para tasas bajas de "*puncturing*", con un nivel de "*soft-decision*" de 2 bits bastaría, pero conforme se sube a tasas superiores no es suficiente, por tanto un valor más apropiado sería tomar un nivel de 3 bits.

- Caso 16-QAM:

En este caso necesitaríamos un nivel de 4 bits para poder obtener resultados correctos tanto con tasas bajas, como con altas de "*puncturing*".

- Caso 64-QAM:

Aquí necesitamos otro nivel adicional, puesto que los errores han aumentado con esta modulación. Tomamos un nivel de 5 bits para "*soft-decision*".

En resumen:

Como hemos podido observar, el número de bits necesario va aumentando conforme aumenta el número de bits utilizado en la modulación y según la tasa de puncturing.

Para elevados niveles de modulación las distancias disminuyen y es necesario aumentar el número de bits de codificación para poder cuantificar de forma apropiada la pertenencia a la zona de decisión.

El aumento de la tasa de puncturing disminuye la capacidad correctora del algoritmo, por lo que es preciso garantizar con una mayor precisión en los datos disponibles, la eficacia del algoritmo.

### 6.2.2.2 Prueba TEST_4.

El esquema de simulación básicamente coincide con el del caso anterior, pero variando la TraceBack-Length (TBL). Las modulaciones utilizadas, son las mismas que en casos anteriores QPSK, 16-QAM y 64-QAM. El siguiente diagrama de flujo nos da las variaciones respecto a los casos anteriores.

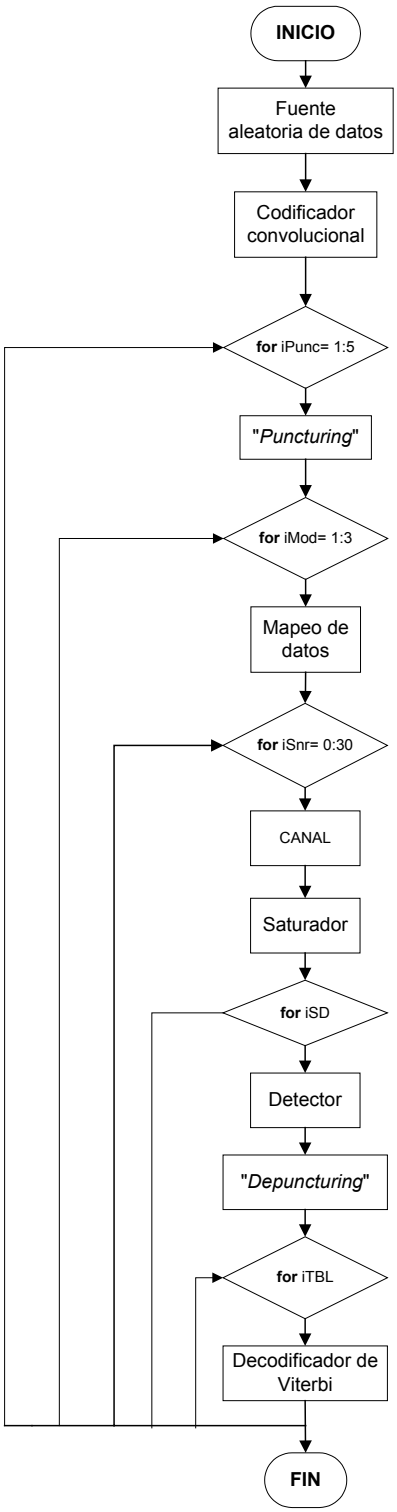



Figura 53: test_4: Diagrama de flujo.

En cada una de las siguientes gráficas se representa para una sola modulación y un sólo patrón de puncturing la diferencia ocurrida en la BER según los distintos TBL. Después de cada gráfica se adjunta la tabla correspondiente a la figura que nos va a servir como referencia para poder comparar resultados y sacar conclusiones.

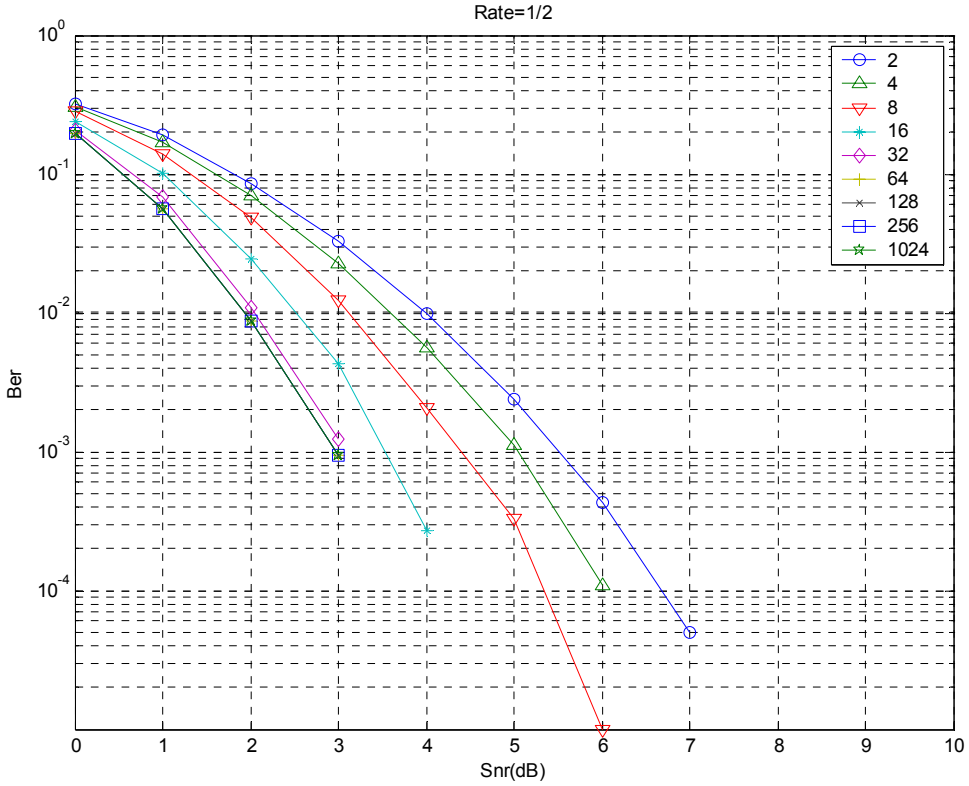



Figura 54: test_4: Influencia de TBL en una modulación QPSK con una tasa de "puncturing" ½.

| TBL<br>SNR | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 1024 |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.3205 | 0.3065 | 0.2801 | 0.2402 | 0.2065 | 0.1959 | 0.1959 | 0.1951 | 0.1949 |
| **2** | 0.0862 | 0.0702 | 0.0483 | 0.0242 | 0.0110 | 0.0087 | 0.0086 | 0.0086 | 0.0086 |
| **4** | 0.0098 | 0.0056 | 0.0021 | 7.00e-4 | 1.15e-4 | 1.12e-4 | 1.10e-4 | 1.00e-4 | 1.00e-4 |
| **6** | 4.30e-4 | 1.10e-4 | 1.00e-5 | - | - | - | - | - | - |
| **8** | 1.00e-5 | - | - | - | - | - | - | - | - |

Tabla 12: Tabla de resultados numéricos correspondiente a la Figura 54.

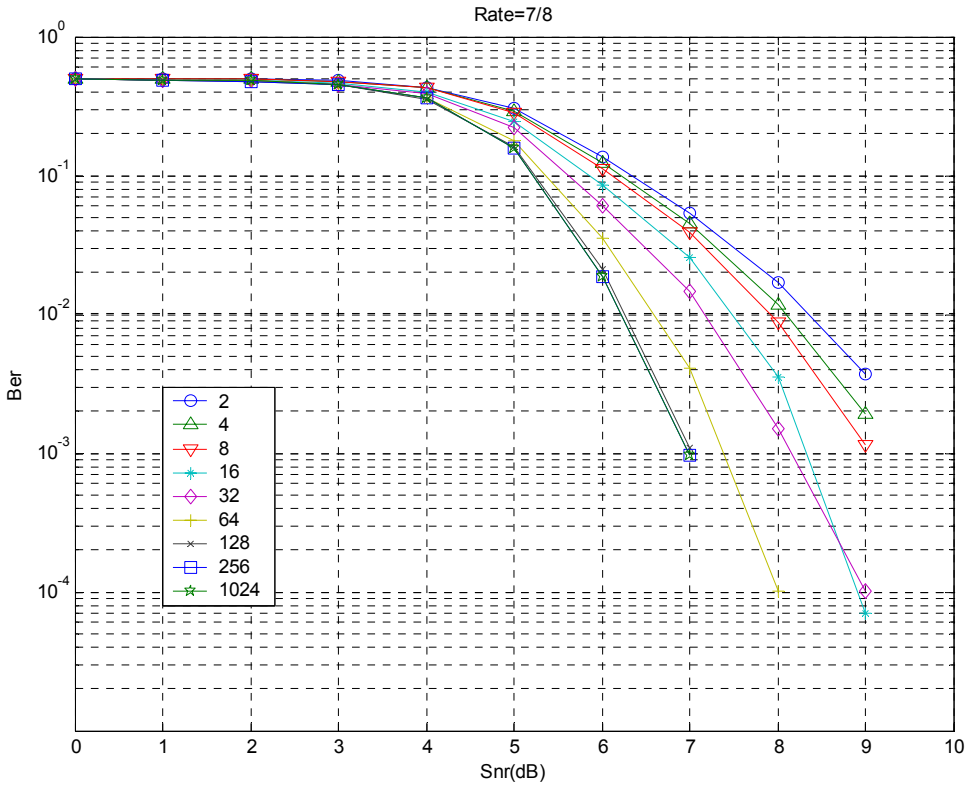



Figura 55: test_4: Influencia de TBL en una modulación QPSK con una tasa de "puncturing" de 7/8.

| TBL<br>SNR | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 1024 |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.4990 | 0.4990 | 0.4985 | 0.4938 | 0.4938 | 0.4936 | 0.4930 | 0.4928 | 0.4928 |
| **2** | 0.4938 | 0.4938 | 0.4938 | 0.4837 | 0.4832 | 0.4810 | 0.4789 | 0.4787 | 0.4785 |
| **4** | 0.4347 | 0.4307 | 0.4254 | 0.3992 | 0.3861 | 0.3627 | 0.3567 | 0.3565 | 0.3560 |
| **6** | 0.1361 | 0.1225 | 0.1121 | 0.0857 | 0.0605 | 0.0350 | 0.0210 | 0.0187 | 0.0186 |
| **8** | 0.0168 | 0.0116 | 0.0088 | 0.0036 | 0.0015 | 1.00e-4 | 1.00e-5 | 1.00e-5 | 1.00e-5 |

Tabla 13: Tabla de resultados numéricos correspondiente a la Figura 55.

Hasta aquí las gráficas para la modulación QPSK, vemos ahora la de una modulación 16-QAM con las mismas tasas de "puncturing" estudiadas.

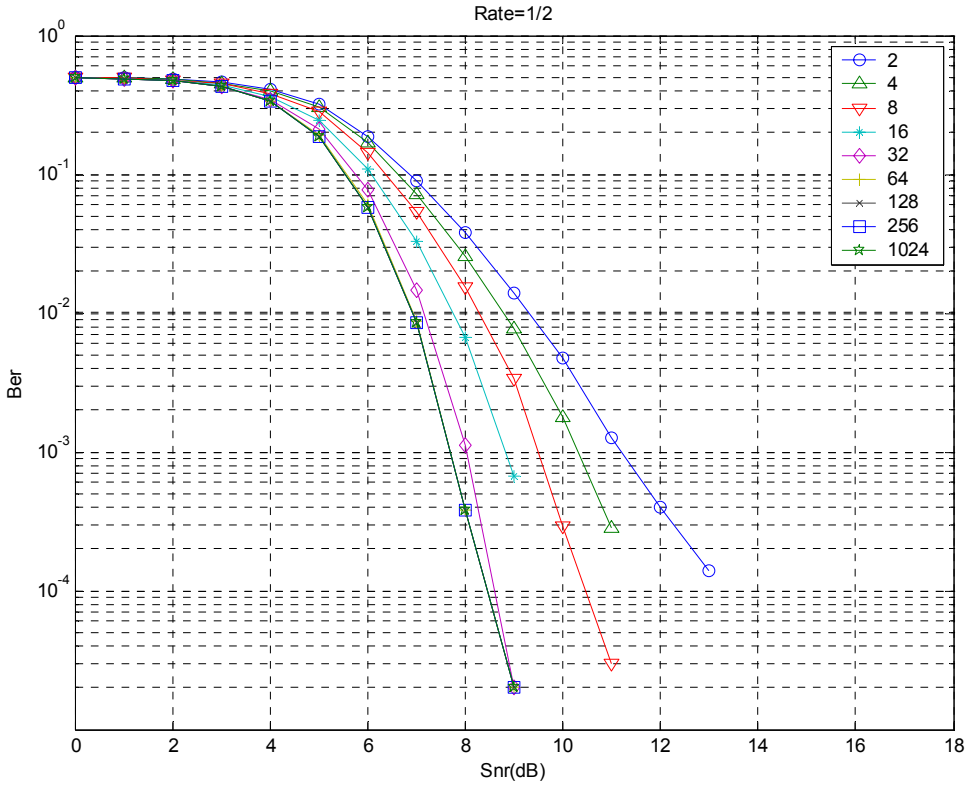



Figura 56: test_4: Influencia de TBL en una modulación 16-QAM con una tasa de "puncturing" de ½.

| TBL<br>SNR | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 1024 |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.4955 | 0.4937 | 0.4956 | 0.4932 | 0.4942 | 0.4927 | 0.4934 | 0.4930 | 0.4930 |
| **3** | 0.4595 | 0.4553 | 0.4492 | 0.4373 | 0.4296 | 0.4294 | 0.4292 | 0.4290 | 0.4289 |
| **6** | 0.1881 | 0.1690 | 0.1437 | 0.1087 | 0.0766 | 0.0607 | 0.0578 | 0.0579 | 0.0579 |
| **9** | 0.0139 | 0.0077 | 0.0033 | 6.60e-4 | 2.00e-5 | 2.00e-5 | 2.00e-5 | 2.00e-5 | 2.00e-5 |
| **12** | 4.00e-4 | 3.00e-5 | - | - | - | - | - | - | - |
| **15** | - | - | - | - | - | - | - | - | - |

Tabla 14: Tabla de resultados numéricos correspondiente a la Figura 56.

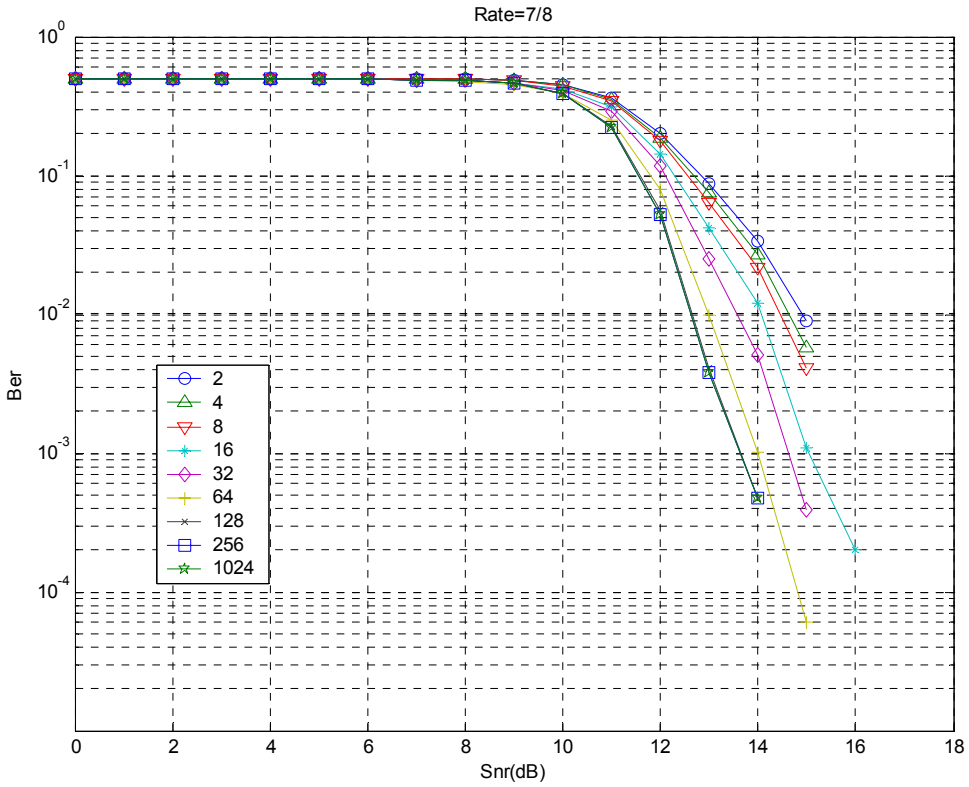



Figura 57: test_4: Influencia de TBL en una modulación 16-QAM con una tasa de "puncturing" de 7/8.

| TBL<br>SNR | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 1024 |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.4999 | 0.4999 | 0.4999 | 0.4999 | 0.4999 | 0.4998 | 0.4998 | 0.4995 | 0.4992 |
| **3** | 0.4999 | 0.4999 | 0.4999 | 0.4999 | 0.4992 | 0.4992 | 0.4987 | 0.4986 | 0.4985 |
| **6** | 0.4999 | 0.4999 | 0.4999 | 0.4940 | 0.4932 | 0.4930 | 0.4930 | 0.4928 | 0.4928 |
| **9** | 0.4874 | 0.4865 | 0.4865 | 0.4629 | 0.4608 | 0.4557 | 0.4552 | 0.4552 | 0.4552 |
| **12** | 0.2020 | 0.1878 | 0.1761 | 0.1439 | 0.1172 | 0.0787 | 0.0565 | 0.0526 | 0.0519 |
| **15** | 0.0090 | 0.0058 | 0.0041 | 0.0011 | 3.90e-4 | 6.00e-5 | 1.00e-5 | 1.00e-5 | 1.00e-5 |

Tabla 15: Tabla de resultados numéricos correspondiente a la Figura 57.

Si tenemos el caso de una modulación 64-QAM, vemos los resultados obtenidos con los mismos parámetros que en los casos anteriores.

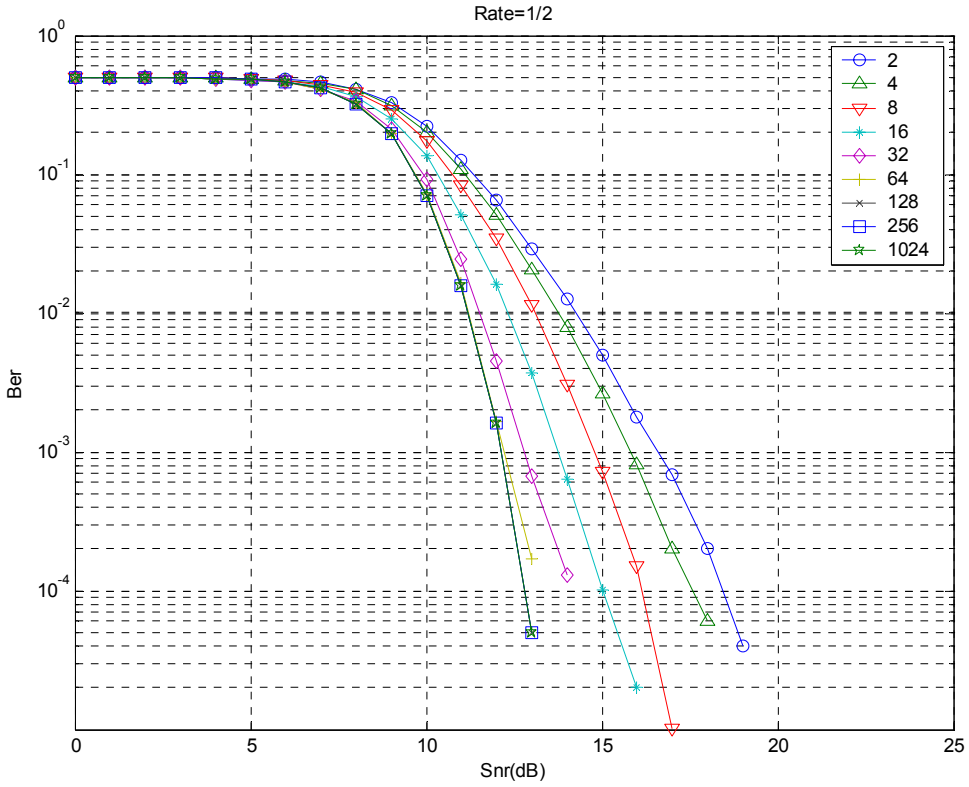



Figura 58: test_4: Influencia de TBL en una modulación 64-QAM con una tasa de "puncturing" de ½.

| TBL<br>SNR | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 1024 |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.4988 | 0.4982 | 0.4987 | 0.4983 | 0.4970 | 0.4995 | 0.4987 | 0.4985 | 0.4983 |
| **3** | 0.4981 | 0.4963 | 0.4962 | 0.4961 | 0.4961 | 0.4959 | 0.4958 | 0.4958 | 0.4958 |
| **6** | 0.4818 | 0.4777 | 0.4730 | 0.4663 | 0.4618 | 0.4618 | 0.4618 | 0.4618 | 0.4618 |
| **9** | 0.3285 | 0.3139 | 0.2903 | 0.2500 | 0.2103 | 0.1961 | 0.1955 | 0.1955 | 0.1954 |
| **12** | 0.0646 | 0.0507 | 0.0343 | 0.0163 | 0.0045 | 0.0016 | 0.0016 | 0.0016 | 0.0016 |
| **15** | 0.0050 | 0.0026 | 7.20e-4 | 1.00e-4 | - | - | - | - | - |
| **18** | 2.00e-4 | 6.00e-5 | - | - | - | - | - | - | - |

Tabla 16: Tabla de resultados numéricos correspondiente a la Figura 58.

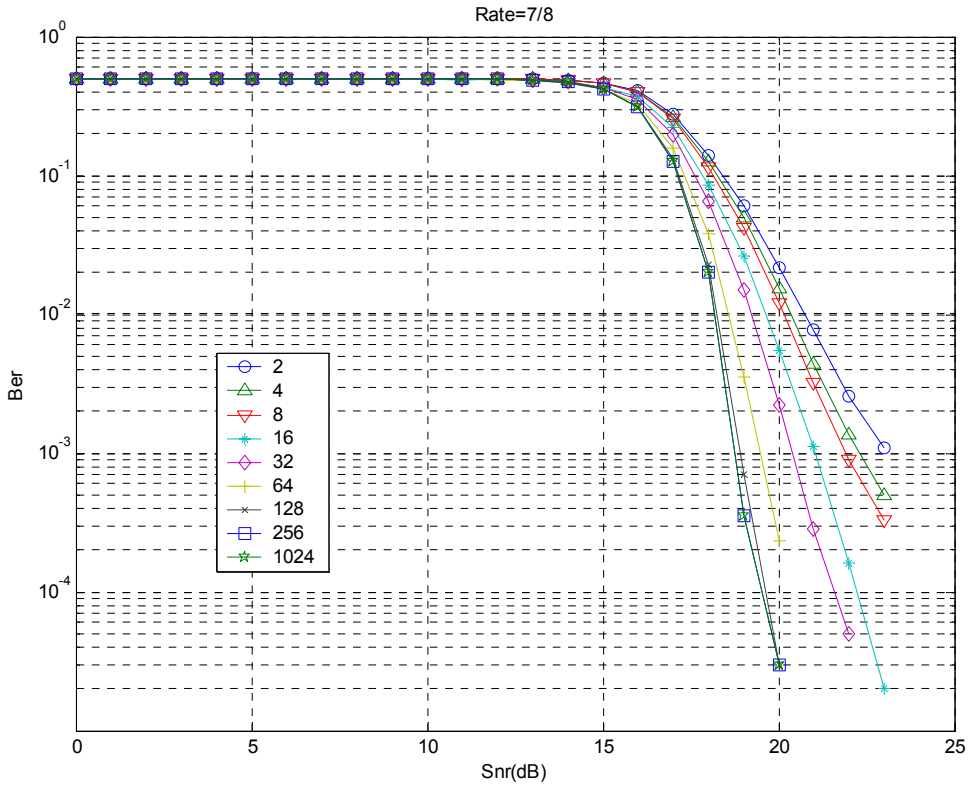



Figura 59: test_4: Influencia de TBL en una modulación 64-QAM con una tasa de "puncturing" de 7/8.

| TBL<br>SNR | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 1024 |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.4999 | 0.4999 | 0.4999 | 0.4999 | 0.4999 | 0.4999 | 0.4997 | 0.4997 | 0.4996 |
| **5** | 0.4999 | 0.4999 | 0.4999 | 0.4999 | 0.4985 | 0.4985 | 0.4985 | 0.4983 | 0.4981 |
| **10** | 0.4999 | 0.4999 | 0.4999 | 0.4999 | 0.4984 | 0.4957 | 0.4955 | 0.4955 | 0.4955 |
| **15** | 0.4622 | 0.4596 | 0.4580 | 0.4346 | 0.4306 | 0.4222 | 0.4203 | 0.4177 | 0.4177 |
| **18** | 0.1393 | 0.1250 | 0.1141 | 0.0863 | 0.0649 | 0.0382 | 0.0228 | 0.0199 | 0.0199 |
| **21** | 0.0077 | 0.0044 | 0.0032 | 0.0011 | 2.80e-4 | - | - | - | - |
| **24** | 4.50e-4 | 2.60e-4 | 1.80e-4 | - | - | - | - | - | - |

Tabla 17: Tabla de resultados numéricos correspondiente a la Figura 59.

Se han tomado las tasas ‘1/2’ y ‘7/8’ por ser los casos extremos, de esta manera tenemos el caso más favorable, y el menos favorable.

Para poder elegir uno u otro valor de TBL, debemos acudir a la norma de DVB-T [8] y ver para qué valores de TBL no existe degradación excesiva de prestaciones.

- QPSK:

Se puede decir que no existe degradación a partir de TBL=32, tanto para tasa de puncturing ‘1/2’ como para ‘7/8’

- 16-QAM:

En este caso para tasa de puncturing ‘1/2’ tambien con TBL=32 se cumple, pero es necesario una TBL=64 para la tasa de puncturing ‘7/8’ con objeto de no degradar en exceso las prestaciones.

- 64-QAM:

Por último, en este caso tanto para tasa de puncturing ‘1/2’ como ‘7/8’ es conveniente adoptar una TBL=64 para conseguir unas prestaciones adecuadas.

# 7 CONCLUSIONES / RESULTADOS.

Se ha analizado la influencia de distintos parámetros en la decodificación Viterbi, en particular el número de bits de entrada (‘*Soft-decision’*) y el recorrido de decodificación (‘*Trace-back-length*’). En este apartado debemos poner en común los resultados de ambas simulaciones, para sacar conclusiones sobre los parámetros a utilizar tanto en sistemas reales (Hardware) como en posteriores simulaciones Software.

Para la prueba en la que se ha variado el número de bits de entrada, test_3, el otro parámetro, TBL, se había mantenido fijo en un valor de 128. Si acudimos a la discusión final sobre los resultados, podemos descubrir que éstos dependen de la modulación utilizada y la tasa de ‘*puncturing*’. Se puede concluir que para una modulación QPSK el número de bits de ‘*soft-decision*’, no debe bajar de tres, para una 16-QAM de cuatro, y para una 64-QAM de cinco. Para dar estos resultados ha habido que comprobar que cumplen con la normativa DVB-T [8].

En la siguiente prueba, test_4, se estudia la influencia del TBL. Se escoge una precisión infinita en el número de bits de entrada para que el único factor influyente en la corrección de errores por parte del algoritmo sea este parámetro en estudio. Si acudimos a las tablas y gráficas que se adjuntan en ese apartado de simulación, se puede comprobar que si escogemos TBL=64, entonces se cumplen las especificaciones dadas por la normativa para las tres modulaciones y las cinco tasas de ‘*puncturing*’.

Como disponemos de diferentes valores de ‘*soft-decision*’ y ‘*Trace-back-length*’ en función de la modulación y puncturing, adoptamos el valor más restrictivo: cinco bits de ‘*soft-decision*’ y 64 de ‘*Trace-back-length*’.

Dada la capacidad de corrección de este algoritmo, es posible que existieran errores de código que se vieran ocultos por dicha capacidad, se ha realizado una validación parcial de cada módulo, comprobando su correcto funcionamiento. Esta comprobación de resultados parciales son las pruebas llevadas a cabo en test_0, test_1 y test_2. Por eso las gráficas adjuntas no tienen demasiada precisión en general, no son las que proporcionan resultados del sistema objeto de estudio. Los resultados de la simulación test_0 están tabulados, y se ha comprobado que son correctos con la bibliografía teórica de procesamiento de señal [6].

# 8 MATLAB.

## *8.1 INTRODUCCIÓN.*

Matlab nació a finales de los años 70, y tenía como finalidad ser utilizado en programas de teoría de matrices, álgebra lineal y análisis numérico.

Hoy en día, sus capacidades se extienden mucho más allá de las expectativas originales. Matlab se ha convertido en un programa de simulación matemática muy potente, tanto si se usa como lenguaje de programación, o como herramienta de visualización de datos. Es un sistema interactivo y un lenguaje de programación científico general y para cálculos técnicos, que ofrece al usuario un conjunto de facilidades o capacidades para la resolución de problemas.

La gran versatilidad de Matlab se debe a que las matemáticas es el lenguaje común para gran parte de la ciencia y la ingeniería, y que su diseño original, fue esto mismo, el procesamiento matemático de los datos. Por esto, Matlab puede ser utilizado en muchos campos diferentes, algunas de estas aplicaciones:

- Analizar y visualizar datos de experimentos con campos magnéticos en superconductores.
- Modelado de sistemas de control, de la trayectoria que sigue un objeto...
- Permite la investigación de esquemas de codificación, compresión, decodificación... en sistemas de telecomunicaciones.

Esta última aplicación es precisamente la nuestra, que utilizamos Matlab como herramienta de verificación de un sistema de comunicaciones específico. Para ello partimos del diseño mediante bloques funcionales del sistema, y luego los simulamos apoyándonos en un conjunto de funciones programadas en Matlab.

## *8.2 SISTEMA DE COMUNICACIONES SIMULADO.*

Nos centramos en un sistema de comunicaciones digital, COFDM, en el que se van variando una serie de parámetros. Según estos parámetros se van obteniendo los resultados y conclusiones analizados en el apartado anterior.

En este apartado se ven por separado cada uno de los bloques que constituyen un sistema de comunicaciones digital, cada uno de ellos simulado en Matlab por una función que se integra en cada una de las pruebas, según corresponda.

### 8.2.1 FUNCIONES.

Vemos cada una de estas funciones por separado, comentando que hace cada una de ellas dentro del sistema, parámetros que se le pasan por línea de comandos, resultados obtenidos y código Matlab.

#### 8.2.1.1 Codificador convolucional: conv_module.

Función que simula el codificador convolucional para una tasa '1/2' de entrada salida. Divide la cadena de entrada en dos vectores de salida que facilitan el punctureado de datos del siguiente bloque.

La llamada a la función:

[Yo, Trellis, LastState] = conv_module (Xi, Initstate)

Parámetros de entrada:

- Xi: Cadena de datos aleatorios de entrada. (1 x k)
- InitState: Estado inicial del codificador convolucional de Matlab.

Parámetros de salida:

- Trellis: Codificador convolucional generado, lo devuelve porque el bloque Viterbi de Matlab lo necesita como entrada para poder decodificar.
- Yo: Código que se le pasa al bloque de punctureado de datos. (2 x k)
- LastState: Estado final del codificador convolucional de Matlab, se pasa como parámetro para iteraciones posteriores dentro de una misma simulación.

Código Matlab:

```
function [Yo,Trellis,LastState]=conv_module(Xi, InitState)

%% ------------------------------------------------------------
%%  función que simula el codificador convolucional
%%  1. Se crea el trellis para una relación 1/2.
%%  2. Creamos el código convolucional que corresponde a los datos de
%%  entrada.
%% ------------------------------------------------------------
%%  Autor: Elisa Garzon
%%  Creation:   10 Marzo 2004
%%  Modified:   23 Abril 2004
%% ------------------------------------------------------------
%% Input:
%  Xi(1,k)
%  InitState: Nos da el estado inicial de los registros del codificador
%  convolucional.
%% ------------------------------------------------------------
%% Output:
%  Yo(2,k): Vector de salida. Es el código codificado.
%  Trellis: Es el trellis que utiliza el codificador, lo devolvemos porque
%  será entrada del Viterbi.
```

```
%  LastState: Ultimo estado de los registros, se devuelve por si tenemos
%  que volver a entrar dentro de un mismo bucle.
%% ------------------------------------------------------------
%% Internal variables:
a=0;    %trellis correcto o no.
cod=0;  %vector para almacenar el código del codificador convolucional
Trellis=0;  % entrada al codificador convolucional
%% ------------------------------------------------------------

%% Checks:
if (nargin ~=2)
   error('codificador convolucional ERROR: Numero de argumentos de entrada erróneo');
   return;
end


%% Convolutional Coding
%% -----------------------------------
% Trellis para una relación 1/2

%Trellis=poly2trellis(3,[5 7]);          % Polinomio generador básico.
Trellis=poly2trellis(7,[171 133]);      % Polinomio que especifica la norma DVBT.

% Comprobamos que el código generado es correcto. No es necesario.

a=istrellis(Trellis);
if (a ~= 1)
   error('codificador convolucional ERROR: Generación del "trellis"');
   return;
end

%% Codificación de la cadena de bits con el trellis generado.

[aux LastState]=convenc(Xi,Trellis,InitState);

% Dividimos el vector que nos da el codificador
% convolucional 1*k en el de salida que va a ser 2*(k/2)

N=length(aux);
Yo(1,1:1:N/2)=aux(1:2:N);
Yo(2,1:1:N/2)=aux(2:2:N);
```

### 8.2.1.2 Punctureado de los datos: punc_dvbt.

Esta función es la que se encarga de quitar redundancia a los datos que se van a enviar por el canal antes de ser mapeados. Le llega un vector bidimensional, y a la salida tendremos uno monodimensional que será la entrada del mapeador.

La llamada a la función:

[Yo] = punc_dvbt (Xi, PUNC_MODE)

Parámetros de entrada:

- Xi: Cadena de datos que viene del codificador convolucional. (2 x k)
- PUNC_MODE: Tasa de puncturing que se va a utilizar. {'1/2','2/3','3/4','5/6','7/8'} .

Parámetros de salida:

- Yo: Código que se le pasa al bloque de mapeado. Su longitud depende de la tasa de puncturing (1 x m)

Código Matlab:

```
function [Yo] = punc_dvbt (Xi, PUNC_MODE)

%% ------------------------------------------------------------
%%  función que simula el punctureado de datos a partir del código que le
%%  llega del codificador convolucional, esto es a partir de una tasa 1/2 y
%%  siguiendo la norma DVBT, obtiene los datos con las distintas tasas k/n.
%% ------------------------------------------------------------
%%  Author: Elisa Garzón
%%  Creation:   10 Marzo 2004
%%  Modified:   24 Abril 2004
%% ------------------------------------------------------------
%% Input:
%  Xi(2,k): Vector bidimensional con los datos del codificador
%          convolucional.
%  PUNC_MODE: Tasas de puncturing. {'1/2', '2/3', '3/4', '5/6', '7/8'}
%% ------------------------------------------------------------
%% Output:
%  Yo(1,m): Vector de salida, vuelve a ser monodimensional
%          como entrada del mapeador.
%% ------------------------------------------------------------
%% Internal variables:
%% ------------------------------------------------------------

%% Checks:
if (nargin ~=2)
   error('puncturing ERROR: Numero de argumentos de entrada erroneo');
   return;
end

N=size(Xi);
```

```
if(N(1)==2)
   NI_COLS=N(2);
else
   if(N(2)==2)
      NI_COLS=N(1);
   else
      error('punct ERROR: El vector de entrada no tiene dimensiones correctas');
   end
end

%% preparamos el puncturing
%% ------------------------------------------------------------

switch (PUNC_MODE)
   case '1/2'
      oMax=2;
      iMax=1;
      puncMatX=[1];
      puncMatY=[1];
   case '2/3'
      oMax=3;
      iMax=2;
      puncMatX=[1,0];
      puncMatY=[1,1];
   case '3/4'
      oMax=4;
      iMax=3;
      puncMatX=[1,0,1];
      puncMatY=[1,1,0];
   case '5/6'
      oMax=6;
      iMax=5;
      puncMatX=[1,0,1,0,1];
      puncMatY=[1,1,0,1,0];
   case '7/8'
      oMax=8;
      iMax=7;
      puncMatX=[1,0,0,0,1,0,1];
      puncMatY=[1,1,1,1,0,1,0];
end

% numero de bloques completos a los que se le aplica el puncturing

NBLOCK_OUT =floor(NI_COLS/(iMax));

% Número de parejas que no forman un bloque completo de puncturing y no se
% le aplica el puncturing

N_REMAIN=NI_COLS-iMax*NBLOCK_OUT;
```

```
% sale por pantalla el numero de parejas de datos no punctureados.

if ( N_REMAIN ~= 0)
        disp (sprintf ('Warn %d data remains',N_REMAIN))
end

%% Puncturing
%% -----------------------------------------------------------

oIdx=1;
for iIdx=1:iMax        %% nos movemos por las columnas de las matrices de puncturing
   if(puncMatX(iIdx)==1)
      Yo(oIdx:oMax:NBLOCK_OUT*oMax)=Xi(1,iIdx:iMax:NBLOCK_OUT*iMax);
      oIdx=oIdx+1;
   end
   if(puncMatY(iIdx)==1)
      Yo(oIdx:oMax:NBLOCK_OUT*oMax)=Xi(2, iIdx:iMax:NBLOCK_OUT*iMax);
      oIdx=oIdx+1;
   end
end
```

#### 8.2.1.3 Mapeo de datos: mapper_dvbt.

Esta función transforma la cadena de bits en coordenadas para ser enviadas por el canal. Estas coordenadas dependen del tipo de modulación utilizada, pero la salida siempre va a estar comprendida entre [-1,1].

La salida va a ser un vector de valores complejos, donde el valor real corresponde a la coordenada "x", y el valor complejo a la coordenada "y".

Los tipos de modulación estudiados van a ser: 'BPSK', 'QPSK', '16-QAM' y '64-QAM'.

La llamada a la función:

[Ymap] = mapper_dvbt (Code,MOD)

Parámetros de entrada:

- Code: Es una cadena de bits monodimensional (1 x m)
- MOD: Tipo de modulación: 'BPSK', 'QPSK', '16-QAM', '64-QAM'.

Parámetros de salida:

- Ymap: Vector complejo de coordenadas. Está cuantificado entre [-1,1]. Su longitud depende de la modulación utilizada. (1 x n)

Código Matlab:

```
function [Ymap]= mapper_dvbt(Code,MOD)

%% ------------------------------------------------------------
%%   Función que nos da el mapeo de la cadena de bits.
%%   Mapeamos segun el tipo de modulacion.
%%   Los valores de salida del mapeo se basan en la norma
%%   "ETSI EN 300 744 v1.4.1 (2001-01)
%% ------------------------------------------------------------
%%  Author: Elisa Garzon
%%  Creation:   10 Marzo 2004
%%  Modified:   22 Abril 2004
%% ------------------------------------------------------------
%% Input:
%   Code = Cadena de bits que nos ha devuelto el codificador convolucional
%   MOD = Tipo de modulacion que va a utilizar para la simulacion:
%              {'BPSK', 'QPSK', '16-QAM', '64-QAM'}.
%% ------------------------------------------------------------
%% Output:
%   Ymap = Codigo mapeado (mapped code).
%% ------------------------------------------------------------
%  Internal variables:
%  xCode = Vector auxiliar.
%  x = Vector auxiliar en que se van almacenando los valores reales del vector de
%       salida.
%  y = Vector auxiliar en que se van almacenando los valores imaginarios del vector de
%       salida.
%% ------------------------------------------------------------

% checks
if (nargin ~=2)
   error('mapper ERROR: Numero de argumentos de entrada erroneo')
   return;
end

NIC=length(Code);

% Mapeamos segun la modulacion.
%% ------------------------------------------------------------

switch (MOD)
   case ('BPSK') %%  Modulacion BPSK.

      NBIT=1;           % Nº de bits que tomamos para el mapeo.
      NOC=NIC;          % Longitud del vector de salida.
      NIC_TR = NOC*NBIT;    % Nº de datos mapeados del vector de entrada.

      x=ones(1,NOC)-2*Code(1:NIC_TR);
      y=zeros(1,NOC);
```

```
   case ('QPSK') %% Modulacion QPSK.

      NBIT=2;
      NOC = floor (NIC/NBIT);
      NIC_TR = NOC*NBIT;

%%  Tomamos los valores impares y los almacenamos en la coordenada x
%%  los valores pares se almacenan en la coordenada y.

      x=ones(1,NOC)-2*Code(1:2:NIC_TR);
      y=ones(1,NOC)-2*Code(2:2:NIC_TR);

   case ('16-QAM') %% Modulacion 16-QAM

      NBIT=4;
      NOC=floor(NIC/NBIT);
      NIC_TR=NOC*NBIT;

      xCode=ones(1,NIC_TR)-2*Code(1:NIC_TR);
      x=sign(xCode(1:4:NIC_TR)).*((2/3)+(1/3)*sign(xCode(3:4:NIC_TR)));
      y=sign(xCode(2:4:NIC_TR)).*((2/3)+(1/3)*sign(xCode(4:4:NIC_TR)));

   case('64-QAM') %% Modulacion 64-QAM

      NBIT=6;
      NOC=floor(NIC/NBIT);
      NIC_TR=NOC*NBIT;

      xCode=ones(1,NIC_TR)-2*Code(1:NIC_TR);

      x = sign (xCode(1:6:NIC_TR)).* ((4/7) + (2/7)*sign(xCode(3:6:NIC_TR)) +
          (1/7)* (sign(xCode(3:6:NIC_TR)).* sign (xCode (5:6:NIC_TR)))) ;

      y = sign (xCode(2:6:NIC_TR)).* ((4/7) + (2/7)*sign(xCode(4:6:NIC_TR)) +
          (1/7)* (sign(xCode(4:6:NIC_TR)).* sign (xCode (6:6:NIC_TR)))) ;


   otherwise

      error ( 'mapper ERROR: El tipo de modulacion elegida no es correcta.');
      return;

end

% Parejas de datos que no han sido mapeados, se dan por pantalla.

N_REMAIN=NIC-(NOC*NBIT);

if (N_REMAIN ~= 0)
   disp (sprintf ( 'mapper WARN: %d bits not mapped',N_REMAIN));
end



%%  Formamos el vector complejo, salida de la funcion.
```

```
i=sqrt(-1);
Ymap=x+i*y;
```

#### 8.2.1.4 Canal: channel.

Función que se encarga de la simulación del canal de comunicaciones, introduce el ruido en el sistema. Nosotros vamos a utilizar un canal AWGN para las simulaciones.

La llamada a la función:

[CodeCANAL] = channel (CodeIFFT,SNR)

Parámetros de entrada:

- CodeIFFT: Vector que viene del modulador, es un vector de valores complejos.
- SNR: Relacion señal a ruido del canal

Parámetros de salida:

- CodeCANAL: Vector complejo salida del canal, valores ruidosos.

Código Matlab:

```
function [CodeCANAL]=channel(CodeIFFT,SNR)

%% ------------------------------------------------------------
%%  AWGN Channel simulation
%% ------------------------------------------------------------
%%  Author: Elisa Garzon
%%  Creation:    1 Abril 2004
%%  Modified:   1 Abril 2004
%% ------------------------------------------------------------
%% Input:
%   CodeIFFT: Modulated signal
%   SNR: Signal/Noise rate.
%% ------------------------------------------------------------
%% Output:
%   CodeCANAL: Signal with AWGN noise
%% ------------------------------------------------------------
%% Internal variables:
%  se van detallando en el codigo.
%% ------------------------------------------------------------

%% Checks
if (nargin ~=2)
   error('canal ERROR: Numero de argumentos de entrada erroneo');
   return;
end


% Energia de la señal
```

```
p=sum(CodeIFFT.*conj(CodeIFFT))/length(CodeIFFT);

%paso de la relacion señal a ruido que nos llega en dB a naturales.
snr=10^(SNR/10);

% calculo de la desviacion tipica de la señal
desv=(sqrt(p/snr)/sqrt(2));

% generacion aleatoria de ruido AWGN.
n=desv*(randn(size(CodeIFFT))+j*randn(size(CodeIFFT)));

% generacion de codigo ruidoso
CodeCANAL=CodeIFFT+n;
```

#### 8.2.1.5 Saturador: Saturador.

A la salida del canal, necesitamos una función que mantenga la señal en el rango deseado, esto es, trunca la señal a [-1,1].

La llamada a la función:

[CodeSAT]=saturador(CodeFFT)

Parámetros de entrada:

- CodeFFT: Vector complejo que contiene los datos tras la salida del canal.

Parámetros de salida:

- CodeSAT: Coordenadas ruidosas, van a ser la entrada del demapeador. Podemos asegurar que se encuentran en un rango [-1,1].

Código Matlab:

```
function [CodeSAT]=saturador(CodeFFT)

%% ------------------------------------------------------------
%%  satura los valores de entrada ruidosos a un rango [-1,1]
%% ------------------------------------------------------------
%%  Author: Elisa Garzon
%%  Creation:   22 Abril 2004
%%  Modified:   22 Abril 2004
%% ------------------------------------------------------------
%% Input:
%  CodeFFT = Vector complejo con datos ruidosos.
%% ------------------------------------------------------------
%% Output:
%  CodeSAT = Vector complejo de datos ruidosos, pero en el rango [-1,1]
%% ------------------------------------------------------------

%% Internal variables:
```

```
%   x = Vector auxiliar donde se almacenan los valores reales del vector de entrada.
%   y = Vector auxiliar donde se almacenan los valores imaginarios del vector de entrada.
%% ------------------------------------------------------------

%% comprobamos el numero de argumentos
if (nargin ~=1)
   error('saturador ERROR: Numero de argumentos de entrada erroneo')
   return;
end

x=real(CodeFFT);
y=imag(CodeFFT);

x(find(x<-1))=-1;
x(find(x>1))=1;
y(find(y<-1))=-1;
y(find(y>1))=1;

CodeSAT=x+i*y;
```

#### 8.2.1.6 Demapeo de datos: demapper_dvbt.

Función que realiza la transformación inversa al mapeador, esto es, la conversión de coordenadas en datos susceptibles de ser entendidos por el decodificador. Esta conversión va a depender del tipo de modulación utilizada. Las modulaciones contempladas son: 'BPSK', 'QPSK', '16-QAM' y '64-QAM'.

En cada uno de los casos vamos a tener salidas diferentes con un vector de longitud variable según el tipo de modulación. Vemos los distintos casos:

- Para 'BPSK': Tenemos un valor del vector binario de salida para cada dato complejo de la entrada.
- Para 'QPSK': Tenemos dos valores binarios por cada entrada del vector de coordenadas.
- Para '16_QAM': Tenemos cuatro valores binarios por cada entrada del vector de coordenadas.
- Para '64-QAM': Se tienen seis valores binarios por cada una de las entradas del vector de coordenadas.

La llamada a la función:

[CodeDEM] = demapper_dvbt (CodeFFT, MOD, Ns)

Parámetros de entrada:

- CodeFFT: Vector de coordenadas ruidoso.
- MOD: Tipo de modulación que se ha utilizado: 'BPSK', 'QPSK', '16-QAM', '64-QAM'.
- Ns: Número de bits para "*soft decision"*.

Parámetros de salida:

- CodeDEM: Vector que va a ser la entrada del Viterbi. No va a ser un vector binario como en su inverso del transmisor.
  Los valores van a depender del número de bits utilizados para 'soft' o 'hard' decision.

Código Matlab:

```
function [CodeDEM] = demapper_dvbt (CodeFFT, MOD, Ns)

%% ------------------------------------------------------------
%%  Funcion que demapea segun el tipo de modulacion que se ha utilizado en el
%%  transmisor.
%%  No se comprueba que MOD sea correcto porque ya se hizo en las
%%  funciones de transmision.
%%  La funcion se divide entres partes:
%%      1. Demapeo: Norma ETSI EN 300 744 v1.4.1 (2001-01)
%%      2. Vectores que componen la salida segun el tipo de modulacion
%%      3. Relleno del vector de salida, tambien segun el tipo de
%%      modulacion.
%% ------------------------------------------------------------
%%  Author: Elisa Garzon
%%  Creation:   10 Marzo 2004
%%  Modified:   22 Abril 2004
%% ------------------------------------------------------------
%% Input:
%   CodeFFT= codigo despues de la demodulacion.
%   MOD= tipo de modulacion, es una cadena de caracteres.
%   Ns= numero de bits para 'soft'.

%% ------------------------------------------------------------
%% Output:
%% ------------------------------------------------------------
%% Internal variables:
% NB_SAL = Nos dice segun la modulacion, el numero de salidas por cada entrada
% Nc = Longitud del vector de entrada.
% x = Parte real del vector de entrada.
% y = Parte imaginaria del vector de entrada.
Q_Ns=2^(Ns-1);      % Valor de la cuantificacion.
Max_Ns=(2^Ns)-1;    % Valor maximo con Ns bits.
%% ------------------------------------------------------------

%% comprobamos el numero de argumentos
if (nargin ~=3)
   error('demapeo ERROR: Numero de argumentos erroneo')
   return;
end

%%  1ºpaso: DEMAPEO.
%% ------------------------------------------------------------
% Demapeamos segun tipo de modulacion, y asignamos un numero de bits a
% cada simbolo.
Nc=length(CodeFFT);
```

```
x=real(CodeFFT);
y=imag(CodeFFT);


switch (MOD)
   case ('BPSK')

%% A partir de 1 datos de entrada,
%% tenemos 1 dato de salida.
       demx0=x;
       NB_SAL=1;

   case ('QPSK')

%% A partir de 1 dato de entrada,
%% tenemos 2 datos en la salida.
       demx0=x;
       demy0=y;
       NB_SAL=2;

   case ('16-QAM')

%% A partir de 1 datos de entrada,
%% tenemos 4 datos de salida.
       demx0=x;
       demy0=y;
       demx1=abs(x)-(2/3)*ones(1,Nc);
       demy1=abs(y)-(2/3)*ones(1,Nc);
       NB_SAL=4;

   case ('64-QAM')

%% A partir de 1 datos de entrada,
%% tenemos 6 datos de salida.
       demx0=x;
       demy0=y;
       demx1=abs(x)-(4/7)*ones(1,Nc);
       demy1=abs(y)-(4/7)*ones(1,Nc);
       demx2=abs(demx1)-(2/7)*ones(1,Nc);
       demy2=abs(demy1)-(2/7)*ones(1,Nc);
       NB_SAL=6;

   otherwise

       error('demapeo ERROR: Tipo de modulacon erronea')
       return;
end


%%  2ºpaso: CUANTIFICACION.
%% ------------------------------------------------------
%%  Vemos si el valor de Ns dado es un valor valido.
```

```
if ((Ns==0)||(Ns<=-2))
   error(sprintf('demapeo ERROR: Nºde bits erroneo NS=%d',Ns));
   return;
end


switch (Ns)
   case (-1)

   % No hace nada, se queda con infinitos valores.

   otherwise

      switch (NB_SAL)

         case (1)    %% BPSK

            %demx0: tiene valores entre 0 y 1
            demx0=(demx0+1)/2;

            %demx0: Max_Ns posibles valores enteros entre 0 y Max_Ns
            demx0=round(Max_Ns*demx0);

            %demx0: Max_Ns posibles valores entre -Max_Ns/2 y Max_Ns/2
           demx0=demx0-(Max_Ns/2);

           %demx0: Max_Ns posibles valores entre -1 y 1 (el 0 no existe)
           demx0=2*demx0/Max_Ns;

         case (2)    %% QPSK

           demx0=(demx0+1)/2;
           demx0=round(Max_Ns*demx0);
           demx0=demx0-(Max_Ns/2);
           demx0=2*demx0/Max_Ns;

           demy0=(demy0+1)/2;
           demy0=round(Max_Ns*demy0);
           demy0=demy0-(Max_Ns/2);
           demy0=2*demy0/Max_Ns;

         case (4)    %% 16-QAM

           demx0=(demx0+1)/2;
           demx0=round(Max_Ns*demx0);
           demx0=demx0-(Max_Ns/2);
           demx0=2*demx0/Max_Ns;

           demx1=(demx1+1)/2;
           demx1=round(Max_Ns*demx1);
           demx1=demx1-(Max_Ns/2);
           demx1=2*demx1/Max_Ns;

           demy0=(demy0+1)/2;
           demy0=round(Max_Ns*demy0);
```

```
            demy0=demy0-(Max_Ns/2);
            demy0=2*demy0/Max_Ns;

            demy1=(demy1+1)/2;
            demy1=round(Max_Ns*demy1);
            demy1=demy1-(Max_Ns/2);
            demy1=2*demy1/Max_Ns;


         case(6)     %% 64-QAM

            demx0=(demx0+1)/2;
            demx0=round(Max_Ns*demx0);
            demx0=demx0-(Max_Ns/2);
            demx0=2*demx0/Max_Ns;

            demx1=(demx1+1)/2;
            demx1=round(Max_Ns*demx1);
            demx1=demx1-(Max_Ns/2);
            demx1=2*demx1/Max_Ns;

            demx2=(demx2+1)/2;
            demx2=round(Max_Ns*demx2);
            demx2=demx2-(Max_Ns/2);
            demx2=2*demx2/Max_Ns;

            demy0=(demy0+1)/2;
            demy0=round(Max_Ns*demy0);
            demy0=demy0-(Max_Ns/2);
            demy0=2*demy0/Max_Ns;

            demy1=(demy1+1)/2;
            demy1=round(Max_Ns*demy1);
            demy1=demy1-(Max_Ns/2);
            demy1=2*demy1/Max_Ns;

            demy2=(demy2+1)/2;
            demy2=round(Max_Ns*demy2);
            demy2=demy2-(Max_Ns/2);
            demy2=2*demy2/Max_Ns;

         otherwise
            error ('demapeo ERROR: Relacion entrada/salida no valida');
            return;

      end
end




%%  3ºpaso: RELLENO DEL VECTOR DE SALIDA (CodeDEM).
%% ------------------------------------------------------------------------
```

```
switch(NB_SAL)
   case (1)    %% BPSK

      CodeDEM=demx0;

   case (2)    %% QPSK

      CodeDEM(1:2:2*Nc)=demx0;
      CodeDEM(2:2:2*Nc)=demy0;

   case (4)    %% 16-QAM

      CodeDEM(1:4:4*Nc)=demx0;
      CodeDEM(2:4:4*Nc)=demy0;
      CodeDEM(3:4:4*Nc)=demx1;
      CodeDEM(4:4:4*Nc)=demy1;


   case (6)    %% 64-QAM

      CodeDEM(1:6:6*Nc)=demx0;
      CodeDEM(2:6:6*Nc)=demy0;
      CodeDEM(3:6:6*Nc)=demx1;
      CodeDEM(4:6:6*Nc)=demy1;
      CodeDEM(5:6:6*Nc)=demx2;
      CodeDEM(6:6:6*Nc)=demy2;


   otherwise
      error ('demapeo ERROR: Relacion entrada/salida, relleno del vector de salida no valido');
      return;

end
```

#### 8.2.1.7 Depunctureado: depunc_dvbt.

Añade los datos de depuncturing al vector de datos demapeados del bloque anterior. Depende de la misma relación K/N del transmisor.

Los datos de depuncturing son “0”, puesto que indican al decodificador que no se sabe nada sobre el dato que va en esa posición.

La llamada a la función:

[Yo, Sel] = depunc_dvbt ( Xi,  PUNC_MODE)

Parámetros de entrada:

- Xi: Código demapeado.
- PUNC_MODE: Relación de codificación convolucional.

Parámetros de salida:

- Yo: Código que está listo para ser decodificado y atacar al bloque decodificador convolucional de Matlab.
- Sel: Vector que indica cual de los datos es bueno, y cual es de "punturing", se le pasa al bloque Viterbi.

Código Matlab:

```
function [Yo,Sel] = depunc_dvbt (Xi, PUNC_MODE)

%% ------------------------------------------------------------
%%  funcion que simula el punctureado de datos a partir del codigo que le
%%  llega del codificador convolucional
%% ------------------------------------------------------------
%%  Author: Elisa Garzon
%%  Creation:   10 Marzo 2004
%%  Modified:   25 Abril 2004
%% ------------------------------------------------------------
%% Input:
%  Xi(1,k) = Vector unidimensional, con los datos demapeados.
%  PUNC_MODE = Tasa de puncturing.
%% ------------------------------------------------------------
%% Output:
%  Yo(2,m) = Vector de salida, es bidimensional. Tiene un '0'
%           en las posiciones de puncturing
%  Sel =
%% ------------------------------------------------------------
%% Internal variables:
%% ------------------------------------------------------------

%% Checks:
if (nargin ~=2)
   error('puncturing ERROR: Numero de argumentos de entrada erroneo');
   return;
end

N=length(Xi);

%% preparamos el puncturing
%% ------------------------------------------------------------
switch (PUNC_MODE)
   case '1/2'
      oMax=2;
      iMax=1;
      puncMatX=[1];
      puncMatY=[1];
```

```
   case '2/3'
      oMax=3;
      iMax=2;
      puncMatX=[1,0];
      puncMatY=[1,1];
   case '3/4'
      oMax=4;
      iMax=3;
      puncMatX=[1,0,1];
      puncMatY=[1,1,0];
   case '5/6'
      oMax=6;
      iMax=5;
      puncMatX=[1,0,1,0,1];
      puncMatY=[1,1,0,1,0];
   case '7/8'
      oMax=8;
      iMax=7;
      puncMatX=[1,0,0,0,1,0,1];
      puncMatY=[1,1,1,1,0,1,0];
end

% numero de bloques completos a los que se le aplica el puncturing
NBLOCK_OUT =floor(N/oMax);

% numero de parejas que no forman un bloque completo de puncturing y no se
% le aplica el puncturing
N_REMAIN=N-(oMax*NBLOCK_OUT);

if( N_REMAIN ~= 0)      % sale por pantalla el numero de parejas de datos no punctureados
   disp(sprintf('Warn %d data remains',N_REMAIN))
end


%DEPUNCTURING
%% ------------------------------------------------------------

oIdx=1;
Yo=zeros(2,iMax*NBLOCK_OUT);
Sel=zeros(2,iMax*NBLOCK_OUT);

for iIdx=1:iMax %% for each column in punct matrix
   if(puncMatX(iIdx)==1)
      Yo(1,iIdx:iMax:NBLOCK_OUT*iMax)=Xi(oIdx:oMax:NBLOCK_OUT*oMax);
      Sel(1,iIdx:iMax:NBLOCK_OUT*iMax)=1;
      oIdx=oIdx+1;
   end
   if(puncMatY(iIdx)==1)
      Yo(2,iIdx:iMax:NBLOCK_OUT*iMax)=Xi(oIdx:oMax:NBLOCK_OUT*oMax);
      Sel(2,iIdx:iMax:NBLOCK_OUT*iMax)=1;
      oIdx=oIdx+1;
   end

end
```

### 8.2.1.8 Decodificador: vit_module.

Función que simula un decodificador de Viterbi mediante la función específica de Matlab.

Utiliza los parámetros 'unquant' tanto para "*hard"* como para "*soft-decision"*, puesto que consideramos la primera como caso especial dentro de la segunda

La llamada a la función:

[ Xdec ] = vit_module ( Trellis, NTB_SIZE, Xi, Xsel)

Parámetros de entrada:

- Trellis : Parámetro necesario para la decodificación, da el código convolucional utilizado en la entrada.
- NTB_SIZE : "Trace-back-length".
- Xi : Vector de entrada con los datos.
- Xsel : Vector selector, nos dice cuales de los datos de entrada son válidos.

Parámetros de salida:

- Xdec: Vector de salida con los datos decodificados, tiene que ser igual a la entrada de la fuente.

Código Matlab:

```
function [ Xdec ] = vit_module (Trellis, NTB_SIZE, Xi, Xsel)
%% ------------------------------------------------------------
%%  Viterbi's decoder simulation
%% ------------------------------------------------------------
%%  Author: Elisa Garzon
%%  Creation:   12 Abril 2004
%%  Modified:   26 Abril 2004
%% ------------------------------------------------------------
%% Input:
%   Trellis
%   NTB_SIZE= "Trace-back-length"
%   Xi= Vector de datos de entrada.
%   Xsel= Vector de seleccion de datos de entrada. En las posiciones donde
%           hay un '0' indica que el dato en el vector de entrada es de puncturing.
%% ------------------------------------------------------------
%% Output:
%   Xdec= Vector decodificado. Salida del sistema de comunicaciones.
%% ------------------------------------------------------------
%% Internal variables:
%% ------------------------------------------------------------

%% Checks
if (nargin ~=4)
   error('Viterbi ERROR: Numero de argumentos de entrada erroneo');
   return;
end
```

```
N_PTS=length(Xi);

%% Matrix --> Vector & Puncturing
depXi=Xi(1,:);

%% Puncturing coordinates
idxPunc=find(Xsel(1,:)==0);
depXi(idxPunc)=0;

%% Convert Matrix --> Vector
xMatIn(1:2:2*N_PTS)=depXi(1:N_PTS);

depXi=Xi(2,:);

%% Puncturing coordinates
idxPunc=find(Xsel(2,:)==0);
depXi(idxPunc)=0;

%% Convert Matrix --> Vector
xMatIn(2:2:2*N_PTS)=depXi(1:N_PTS);

%% Llamada a la funcion de Matlab.
Xdec=vitdec(xMatIn, Trellis, NTB_SIZE, 'trunc', 'unquant');
```

# 9 ANEXOS.

## *9.1 TIPOS DE MÉTRICAS.*

A continuación, se hace un estudio para el cálculo de distancias basándonos en tres métricas concretas, métrica "*Euclídea*", métrica "*Hamming*" y métrica "*city-block*".

- Métrica "*Euclídea*": Nos da la distancia real entre el dato recibido, y el ideal. La representación gráfica se ve en la Figura 60**.**

$$d^2 = (y_2-y_1)^2 + (x_2-x_1)^2$$

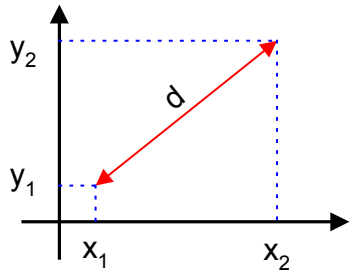



Figura 60: Métrica "*Euclídea*", formulación y gráfica.

- Métrica "*Hamming*": Nos da una distancia en función del número de cambios que hay entre el dato recibido y el ideal esperado en recepción.

- Métrica "*city-block*": Nos da la suma de las distancias de las coordenadas del dato recibido al dato ideal esperado. Su formulación matemática y su gráfica se ven en la Figura 61.

$$d = (y_2-y_1) + (x_2-x_1)$$

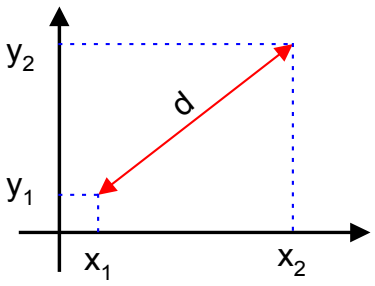



Figura 61: Métrica "*city-block*", formulación y gráfica.

Para ver la métrica a utilizar, tomamos varios ejemplos, y observamos resultados.

**Ejemplos:**

**1.** Suponemos que no existe el *puncturing*, tomamos *'hard-decision'* para facilitar los cálculos, con el número de bits en cada una de las entradas igual a uno. Por tanto, podemos tomar este caso como uno particular de "*soft-decision*" pero con número de bits igual a uno.

Vemos el resultado en la Tabla 18.

| | | Euclídea | | | | Hamming | | | | City-block | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Entradas: I0** | **I1** | **Bm00** | **bm01** | **bm10** | **bm11** | **Bm00** | **bm01** | **bm10** | **bm11** | **Bm00** | **bm01** | **bm10** | **bm11** |
| **0** | **0** | 0 | 2 | 2 | $\sqrt{8}$ | 0 | 1 | 1 | 2 | 0 | 2 | 2 | 4 |
| **0** | **1** | 2 | 0 | $\sqrt{8}$ | 2 | 1 | 0 | 2 | 1 | 2 | 0 | 4 | 2 |
| **1** | **0** | 2 | $\sqrt{8}$ | 0 | 2 | 1 | 2 | 0 | 1 | 2 | 4 | 0 | 2 |
| **1** | **1** | $\sqrt{8}$ | 2 | 2 | 0 | 2 | 1 | 1 | 0 | 4 | 2 | 2 | 0 |

Tabla 18: Tabla comparativa de las distintas métricas para "hard-decision".

Observando el resultado obtenido, vemos cómo los resultados son similares en los tres casos, sólo que aparecen escalados. Por tanto, resulta obvio concluir que es indiferente la métrica a utilizar, porque el resultado no variará con ella. A continuación, estudiamos cada una de ellas por separado para ver cual nos ofrece mejores prestaciones y elegir en consecuencia.

- Si se utiliza la métrica "*Euclídea*", necesitamos sumadores, multiplicadores y raíces cuadradas para cada uno de los datos, además de obtener números reales no enteros.

- Si se utiliza la métrica "*city-block*", se necesitan sólo sumadores, pero el resultado puede no ser un número entero y es difícil de cuantizar.

- Si nos quedamos con la métrica "*Hamming*", siempre va a ser un número entero, y para conseguirla, sólo se necesitan comparadores y sumadores, de menor coste computacional que los multiplicadores.

En conclusión, para este caso concreto, la que nos ofrece mejores resultados es la métrica "*Hamming*", seguida por la "*city-block*", y en último caso la "*Euclídea"* de mayor coste computacional.

**2.** En este segundo caso, tomamos el número de entradas igual a dos, esto es dos bits de entrada para un sólo bit a decodificar. Tampoco se tiene en cuenta el *puncturing*.

El resultado puede verse en la siguiente tabla:

| | | Euclídea | | | | Hamming | | | | City-block | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Entradas: I0 | I1 | Bm00 | bm01 | bm10 | bm11 | Bm00 | bm01 | bm10 | Bm11 | Bm00 | Bm01 | bm10 | bm11 |
| 00 | 00 | 0 | 3 | 3 | $\sqrt{18}$ | 0 | 1 | 1 | 2 | 0 | 3 | 3 | 6 |
| 00 | 01 | 1 | 2 | $\sqrt{10}$ | $\sqrt{13}$ | 1 | 2 | 2 | 3 | 1 | 2 | 4 | 5 |
| 00 | 11 | 2 | 1 | $\sqrt{13}$ | $\sqrt{10}$ | 2 | 1 | 3 | 2 | 2 | 1 | 5 | 4 |
| 00 | 10 | 3 | 0 | $\sqrt{18}$ | 3 | 1 | 0 | 2 | 1 | 3 | 0 | 6 | 3 |
| 01 | 00 | 1 | $\sqrt{10}$ | 2 | $\sqrt{13}$ | 1 | 2 | 2 | 3 | 1 | 4 | 2 | 5 |
| 01 | 01 | $\sqrt{2}$ | $\sqrt{5}$ | $\sqrt{5}$ | $\sqrt{8}$ | 2 | 3 | 3 | 4 | 2 | 3 | 3 | 4 |
| 01 | 11 | $\sqrt{5}$ | $\sqrt{2}$ | $\sqrt{8}$ | $\sqrt{5}$ | 3 | 2 | 4 | 3 | 3 | 2 | 4 | 3 |
| 01 | 10 | $\sqrt{10}$ | 1 | $\sqrt{13}$ | 2 | 2 | 1 | 3 | 2 | 4 | 1 | 5 | 2 |
| 10 | 00 | 3 | $\sqrt{18}$ | 0 | 3 | 1 | 2 | 0 | 1 | 3 | 6 | 0 | 3 |
| 10 | 01 | $\sqrt{10}$ | $\sqrt{13}$ | 1 | 2 | 2 | 3 | 1 | 2 | 4 | 5 | 1 | 2 |
| 10 | 11 | $\sqrt{13}$ | $\sqrt{10}$ | 2 | 1 | 3 | 2 | 2 | 1 | 5 | 4 | 2 | 1 |
| 10 | 10 | $\sqrt{18}$ | 3 | 3 | 0 | 2 | 1 | 1 | 0 | 6 | 3 | 3 | 0 |
| 11 | 00 | 2 | $\sqrt{13}$ | 1 | $\sqrt{10}$ | 2 | 3 | 1 | 2 | 2 | 5 | 1 | 4 |
| 11 | 01 | $\sqrt{5}$ | $\sqrt{8}$ | $\sqrt{2}$ | $\sqrt{5}$ | 3 | 4 | 2 | 3 | 3 | 4 | 2 | 3 |
| 11 | 11 | $\sqrt{8}$ | $\sqrt{5}$ | $\sqrt{5}$ | $\sqrt{2}$ | 4 | 3 | 3 | 2 | 4 | 3 | 3 | 2 |
| 11 | 10 | $\sqrt{13}$ | 2 | $\sqrt{10}$ | 1 | 3 | 2 | 2 | 1 | 5 | 2 | 4 | 1 |

Tabla 19: Tabla comparativa de las distintas métricas para un nivel de "soft-decision" de 2 bits.

En esta tabla, ya no se ven tan claros los resultados obtenidos, porque sólo cuatro de las dieciséis posibles entradas son los puntos hasta los que se mide la distancia, señalados en rojo. Los demás, señalados en gris, son todos lo posibles valores de la entrada. Por tanto, en este caso, se va a hacer un esquema gráfico que va a ayudar a comprender la tabla, se ve en la Figura 62.

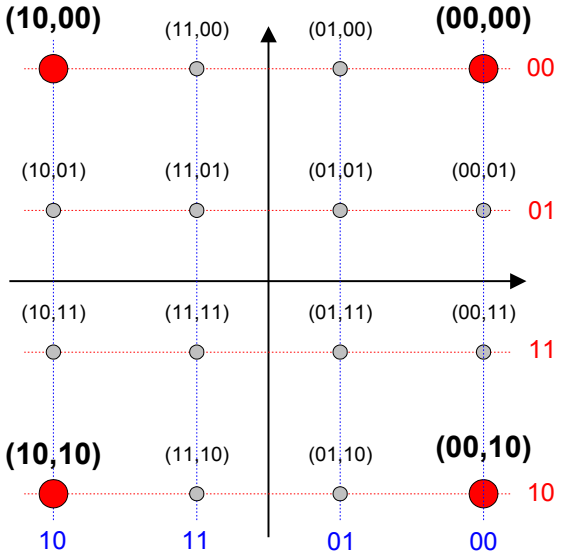



Figura 62: Referencia para las distancias.

En este caso, no es indiferente la métrica a utilizar, puesto que cambian los resultados según la métrica utilizada. Vemos cada uno de los casos por separado:

- Si se utiliza la métrica "*Euclídea*", obtenemos distancias reales, por tanto nos va a dar un resultado válido. Se necesitan sumadores, multiplicadores y raíces cuadradas para cada uno de los datos, además de obtener números reales no enteros.

- Si se utiliza la métrica "*city-block*", también es válida, se siguen obteniendo distancias reales. Se necesitan sumadores y comparadores.

- Si nos quedamos con la métrica "*Hamming*", no obtenemos en este caso distancias reales, como podemos observar en la Tabla 19, con ayuda del gráfico de la Figura 62. A distancias mayores, le corresponden métricas menores. Es una métrica que hay que descartar.

En conclusión, para este caso concreto, la que nos ofrece mejores resultados es la métrica "*city-block*", seguida de la métrica "*Euclídea*" y descartando la "*Hamming"*, por no ofrecer resultados correctos.

**Conclusión:**

A la vista de los resultados obtenidos en los ejemplos anteriores, se recomienda el uso de la métrica "*city-block*", en el bloque para el cálculo de la "*branch-metric*". Puesto que es la métrica que mejores prestaciones nos ofrece de las estudiadas. Una métrica "*Hamming*" sólo puede ser utilizada en el caso de "*hard-decision*", ya que en otro caso lleva a confusión, y la "*Euclídea*" lleva asociado un mayor coste computacional.

# 10 BIBLIOGRAFÍA.

[1] "Digital Communications" Ian Glover (Universidad de Bradford), Peter Grant (Universidad de Edinburgh) Editorial: Prentice Hall, 2000.

[2] "Mastering Matlab6, A Comprehensive Tutorial and Reference" Duane Hanselman, Bruce Littlefield. Editorial: Prentice Hall, 2001.

[3] "Matlab, edición de estudiante" Duane Hanselman, Bruce Littlefield. Editorial: Prentice Hall, 1995.

[4] "Principles of digital communication and Coding" Andrew J. Viterbi, Jim K. Omura. Editorial: McGraw-Hill, 1979.

[5] "Mobile Radio Communications" Raymond Steele. Editoral: Wiley (John Wley & Sons), 1996.

[6] "Digital Communications", 3ª Edición. John G. Proakis. Editorial: McGraw-Hill, 1995.

[7] "Implementing the Viterbi Algorithm", Hui-Ling-Lou. Fundamentals and real-time issues for processor designers. IEEE Signal Processing Magazine (1053-5888), Sep 1995.

[8] ETSI-EN-300-744-V1.4.1 (2001-01): Digital Video Broadcasting (DVB); Framming structure, channel coding and modulation for digital terrestrial television.

[9] "Explaining some of the magic of COFDM", JH Stott. Proceedings of 20h International Television Symposium 1997, Montreux. BBC Research & Development Department Report.

[10] "www.PlanetMath.org"

[11] "Técnicas de Sincronización en Sistemas OFDM para comunicaciones por Línea de Potencia.", Joaquín Granado Romero. Trabajo de Investigación, 2002.

[12] "Televisión Digital Terrestre". Jaime Sancho, Marcelo Pandolfo, informe técnico televisión chilena.

[13] "A Tutorial on Convolutional Coding with Viterbi Decoding", Chip Fleming. Spectrum Applications, 2003.