Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN TECNOLOXÍAS DA INFORMACIÓN

Implementación software de un Simulador de un Sistema de Comunicaciones Inalámbricas

Estudante: Javier Mejuto Vázquez
Dirección: Óscar Fresnedo Arias
José Pablo González Coma

A Coruña, xuño de 2020.

A mis padres y hermano

Agradecimientos

Primero, me gustaría agradecer a mis directores de proyecto, Óscar Fresnedo Arias y José Pablo González Coma, su dedicación y ayuda en todo momento, así como su magnífico trato.

Quiero dar las gracias a mis padres y mi hermano por su apoyo incondicional, sobretodo en los momentos más difíciles.

A todos los amigos que he conocido durante la carrera. Gracias a vosotros este duro camino ha resultado mucho más ameno.

Resumen

La sociedad actual no podría concebirse sin los sistemas de comunicaciones inalámbricas. Dado su gran interés, tanto desde el punto de vista académico como por parte de la industria, se propone en este trabajo de fin de grado la implementación *software* de un simulador de un sistema de comunicaciones completo.

Este simulador permite simular, de forma fiable, el envío de información desde un emisor a un receptor a través de una canal de comunicaciones. Para ello, se implementan las diferentes etapas por las que deben pasar los datos, entre las que se encuentran la modulación y demodulación, técnicas avanzadas de codificación o la ecualización del canal. Se hace especial énfasis en las técnicas de codificación, ya que permiten minimizar el impacto de los errores que se producen debido a las distorsiones introducidas por el canal.

Además, los usuarios serán capaces de ver y comparar los resultados para diferentes configuraciones del sistema de comunicaciones a través de la interfaz gráfica desarrollada. En ella se incluyen gráficas que muestran de forma visual el número de errores en el receptor, así como la información más relevante en cuánto a la simulación realizada.

Abstract

Nowadays, society could not be conceived without wireless communication systems. Given its great interest, from both an academic point of view and on the part of industry, the software implementation of a complete communications system simulator is proposed in this final degree project.

This simulator allows to reliably simulate the sending of information from a sender to a receiver over a communication channel. To achieve this, different stages are implemented where required data must be transmitted, including modulation and demodulation operations, advanced coding techniques and channel equalization. Special emphasis is made to these coding techniques, since they allow to minimize the impact of errors due to distortions introduced by the channel.

In addition, users will be able to observe and compare the results of different configurations of communication systems, thanks to the developed graphical interface. It includes graphs that show the number of errors at the receiver, as well as the most relevant information regarding the performed simulation.

Palabras clave:

- Codificación
- Modulación
- Demodulación
- Decodificación
- Comunicaciones inalámbricas
- Sistemas MIMO
- Interfaz Gráfica
- Python

Keywords:

- Coding
- Modulation
- Demodulation
- Decoding
- Wireless comunications
- MIMO systems
- Graphical interface
- Python

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1 Introducción

La comunicación es el proceso por el cuál un emisor y un receptor establecen una conexión mediante la transmisión de un mensaje, por un canal dado y empleando un código preestablecido, que les permite intercambiar o compartir ideas e información. Esto la convierte en un instrumento social de cambio muy importante.

En el ámbito informático, esto no es una excepción, dado que el número de personas que se conectan a Internet es cada vez mayor. Hoy por hoy, las Tecnologías de la Información y Comunicaciones Tecnologías de la Información y la Comunicación (TIC)) juegan un papel muy importante en la vida cotidiana y profesional del hombre, dado que estas herramientas permiten obtener información de forma más rápida junto con la posibilidad de definir nuevos modelos de comunicación que sirvan como soporte en el nuevo milenio de la informatización. Todas estas herramientas tecnológicas se basan en la posibilidad de intercambiar información de forma fiable entre dos nodos cualquiera, independientemente de la distancia que los separa. Por esa razón, es tan importante diseñar sistemas de comunicaciones digitales que garanticen una transmisión fiable de la información.

En el contexto de las comunicaciones digitales, los fundamentos en los que se asienta la teoría de la comunicación fueron establecidos en 1948, cuando Claude Elwood Shannon publica *"A Mathematical Theory of Communication"*[1]. En este trabajo, aborda el principal problema de la comunicación: transmitir un mensaje de la manera más fidedigna posible desde la fuente de información –las cuáles demostró que son medibles– hasta el receptor a través de un medio o canal que puede distorsionar el mensaje original añadiendo ruidos o interferencias. Además, estableció de forma teórica como determinar la velocidad máxima a la que es posible transmitir la información por un medio físico con una probabilidad de error arbitrariamente pequeña, es decir, de forma fiable. Esta idea se ha demostrado clave en el diseño de los sistemas de comunicaciones digitales y se conoce con el nombre de capacidad de canal.

Por otro lado, en los últimos años, el crecimiento de las comunicaciones inalámbricas ha sido exponencial y ha supuesto un gran beneficio para toda la sociedad. Han ido apareciendo multitud de estándares inalámbricos como LTE, WiMax o WiFi; los cuáles utilizan esquemas de codificación sofisticados como turbo códigos o códigos LDPC e incorporan la posibilidad de transmitir o recibir la información usando varias antenas.

En el presente proyecto, haremos el mismo trayecto que estableció Shannon, siguiendo todas las fases que recorre un mensaje para ser enviado desde un emisor hasta un receptor considerando, además, el caso de que la comunicación se realice a través del medio inalámbrico. Para iniciar el recorrido, empezaremos explicando en que consiste la operación de codificación, así como, los diferentes tipos que se utilizan y como han evolucionado y mejorado a lo largo de los últimos años. Seguidamente, abordaremos los esquemas de modulación y como la información, una vez codificada, debe ser transformada en señales físicas adecuadas para poder ser transmitida a través del canal. Este elemento constituye el medio físico por donde es transmitido el mensaje. Esta fase es una de las más importantes ya que, en este punto, es donde el mensaje que se transmite sufre la gran mayoría de modificaciones en su contenido debido a las perturbaciones y limitaciones de cada canal. Se explicará cómo funcionan los principales canales usados en comunicaciones, haciendo especial énfasis en el caso de comunicaciones inalámbricas.

Una vez la señal alcanza el destino, toca deshacer los cambios introducidos en el mensaje original. Para ello, es necesario realizar una operación de ecualización y, a continuación, se llevará a cabo la demodulación de la señal recibida para determinar cuales han sido los símbolos que se han transmitido con mayor probabilidad. Por último, la decodificación, fase muy importante también porque aquí es dónde se corrigen los posibles errores que se producen en la transmisión. Se analizarán los diferentes tipos de decodificación que se pueden aplicar dependiendo del esquema de codificación empleado y se explicará su funcionamiento. El objetivo final es que el mensaje recibido por el receptor sea igual al mensaje original enviado por el emisor.

Para poder entender todo este proceso de una manera más visual, se implementará un simulador de comunicaciones, con su correspondiente interfaz gráfica, en el que se podrán elegir diferentes opciones y parámetros para los esquemas de codificación y modulación, y se podrá simular la transmisión del mensaje resultante a través de diferentes tipos de canales. La herramienta gráfica mostrarán además una serie de gráficas muy informativas tras la simulación, incluyendo curvas de probabilidad de error, constelaciones de los símbolos transmitidos y recibidos, diagramas de bloques de ciertas componentes del sistema y pantallas con información relevante sobre la transmisión y del sistema de comunicaciones. Este tipo de información permite comparar de forma muy visual y didáctica el rendimiento y características de diferentes esquemas de comunicación.

1.2 Otros simuladores de comunicaciones

En esta sección se presentan dos simuladores que han servido como fuente de inspiración a la hora de realizar el simulador de comunicaciones desarrollado.

Simulador de la Universidad INCCA

En primer lugar, tenemos un sistema de comunicaciones que utiliza la modulación *Quadrature Amplitude Modulation* (QAM) y *Orthogonal Frequency Division Multiplexing* (OFDM) [2]. Su interfaz gráfica, realizada en MatLab, es muy sencilla, ya que únicamente presenta cuatro parámetros que deben ser elegidos por el usuario en los que se incluyen el número de subportadoras, el número de niveles para la modulación QAM; la frecuencia de muestreo; y un único valor para la relación señal a ruido del canal o *Signal-to-Noise Ratio* (SNR). En este caso, además se asume que el canal por el que se transmite la información modulada es un canal *Additive White Gaussian Noise* (AWGN).

Tras seleccionar los parámetros deseados, si ejecutamos la aplicación se mostrarán dos gráficas como las se pueden apreciar en la figura 1.1.

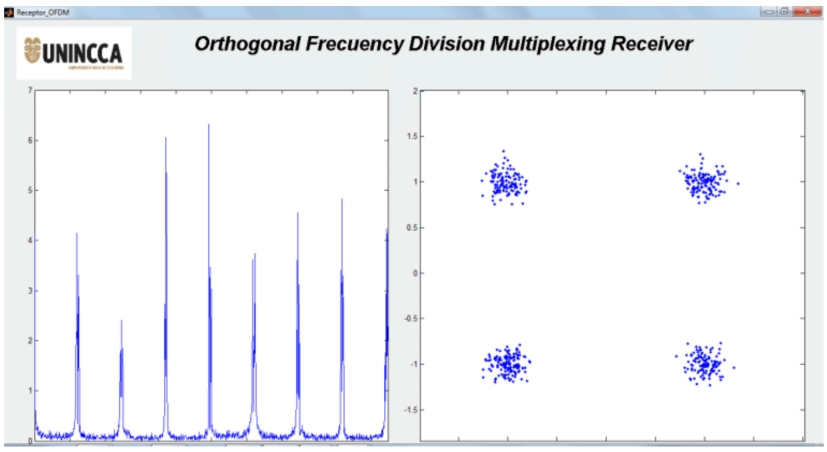

Figura 1.1: Gráficas del simulador tras ejecutar la app

La primera de ellas muestra el espectro de la señal enviada, mientras que la segunda muestra la constelación de los símbolos recibidos, donde se puede apreciar como el ruido afecta a los símbolos originalmente transmitidos.

Este simulador presenta numerosas limitaciones con respecto a la posibilidad de elegir diferentes tipos de parámetros del sistema de comunicaciones, considerar diferentes esquemas de transmisión o simular el envío de la información por diferentes tipos de canales. Además, solo se permite elegir un valor de SNR concreto y no un rango de ellos. Esta circunstancia impide que se pueda generar una gráfica que muestre la tasa de errores (de bit) que obtiene el

esquema de transmisión en función de las diferentes condiciones del canal que, generalmente, viene determinada por el valor de SNR. Este tipo de curvas que relacionan la tasa de error con un rango de valores de SNR resultan muy útiles para comparar experimentalmente las prestaciones de los diferentes componentes de un sistema de comunicaciones.

Además, se puede observar que en este simulador no se aplica ninguna técnica de corrección de errores ya que la información a transmitir es directamente modulada, transmitida por el canal, y demodulada.

Simulador basado en Simulink

Este simulador está realizado con Simulink, un lenguaje de alto nivel basado en Matlab. En él se implementan algunas de las principales modulaciones digitales: 16QAM, 64QAM, ... y multiplexación OFDM. Cuenta con varias pestañas para configurar el sistema, entre las que destaca una pestaña en la que se eligen los parámetros para obtener los resultados teóricos de la comunicación, como se muestra en la figura 1.2. Además, dispone de una pestaña en la que se eligen los parámetros para realizar la simulación, como se puede apreciar en la figura 1.3.

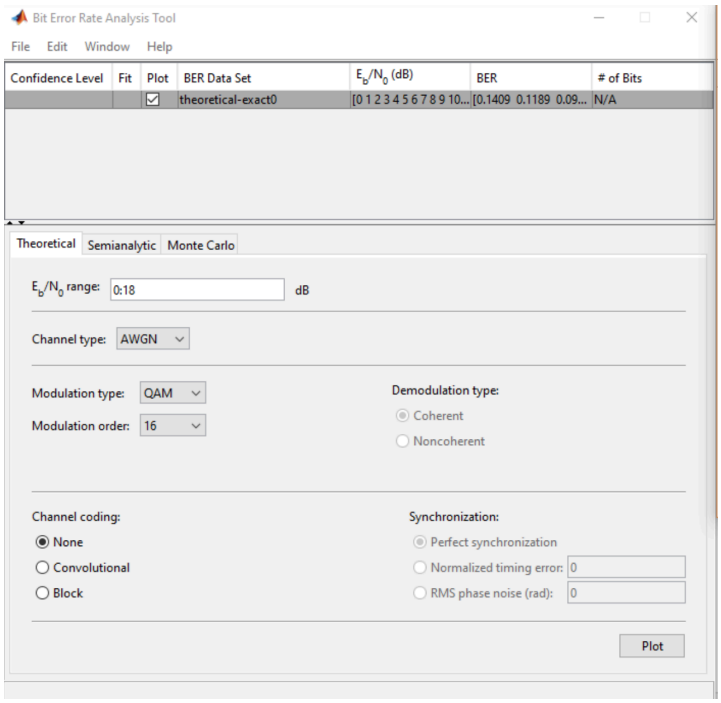

Figura 1.2: Pestaña de la aplicación para resultados teóricos

Como salida se obtienen gráficas en función de la relación señal a ruido, en la que se muestran las interferencias y errores que se dan en la señal transmitida tras el paso por un canal AWGN, incluyendo la posibilidad de comparar los resultados de simulación con los resulta-

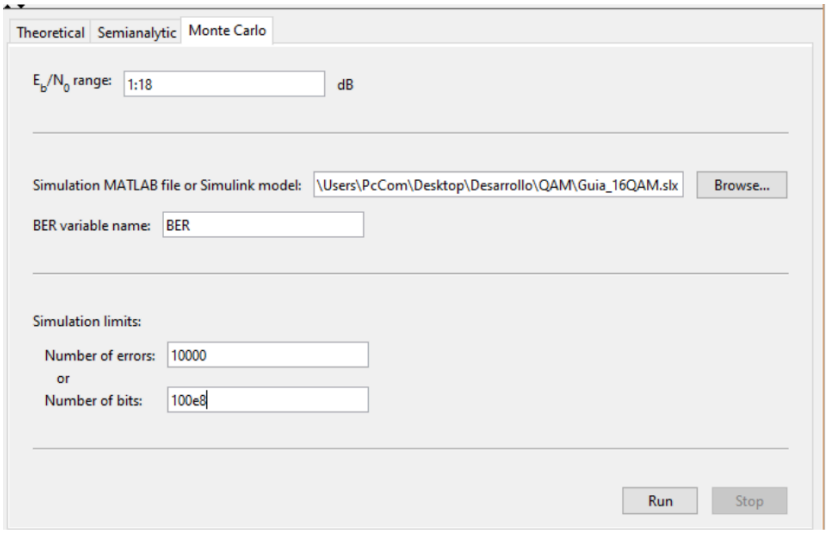

Figura 1.3: Pestaña de la aplicación para la simulación

dos teóricos. Además de esto, se muestra una gráfica con la constelación para la modulación elegida y de los símbolos recibidos, como se puede apreciar en la figura 1.4.

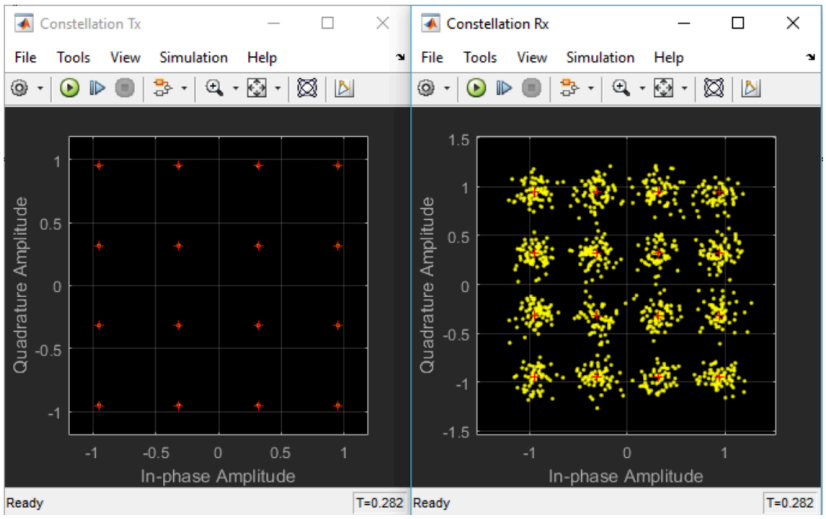

Figura 1.4: Gráfica de la constelación

En comparación con el simulador desarrollado en este proyecto, esta aplicación no considera el uso de técnicas modernas de comunicación ni el uso de técnicas basadas en el uso de *Multiple-Input Multiple-Output* (MIMO). Además, no muestra información relevante acerca de la simulación ni es posible comparar el rendimiento entre diferentes esquemas de transmisión.

1.3 Motivación

El motivo de realizar este proyecto surge por la gran importancia que tienen y han tenido siempre los sistemas de comunicaciones en la vida de las personas. Hoy en día, en el mundo informatizado en el que vivimos es clave que los sistemas de comunicaciones inalámbricas sean capaces de llevar a cabo una transferencia de datos lo más rápido posible y, sobre todo, con la mayor veracidad que se pueda. Todavía más si cabe, tras la situación vivida recientemente con la aparición del COVID-19, puesto que estos sistemas han permitido las comunicaciones entre familiares, la posibilidad del trabajo a distancia, compartir información médica, etc.

Por esa razón, es importante estudiar y comprender los diferentes componentes que conforman un sistema de comunicaciones inalámbrico y que permiten transmitir información a grandes distancias, así como las diferentes técnicas que se utilizan en la actualidad para detectar y corregir los errores que se dan durante este proceso. Esta comprensión resulta fundamental para ser capaces de diseñar esquemas de comunicación eficientes que cumplan con las crecientes demandas de la sociedad en general. En este punto, aparece además la necesidad de contar con simuladores avanzados que nos permitan evaluar de forma práctica y con un mínimo coste el rendimiento de un sistema de comunicación, verificar en la práctica los resultados teóricos que nos ofrecen herramientas como la teoría de la información o comprobar el impacto de los diferentes parámetros involucrados en el diseño de estos sistemas.

Por estas razones y debido a la ausencia de herramientas que cubran en profundidad los aspectos comentados, se ha decidido llevar a cabo un simulador de un sistema de comunicaciones inalámbricas que incorpore la posibilidad de configurar de forma sencilla diferentes componentes del sistema y mostrar visualmente resultados interesantes e ilustrativos sobre su funcionamiento. Además, el diseño de la aplicación se ha realizado valorando la posibilidad de utilizar este simulador desde un punto de vista didáctico debido a la presentación de los resultados e información relevante de forma visual y muy comprensible.

1.4 Objetivos

Los principales objetivos del proyecto se pueden resumir en:

- Analizar y comprender en detalle el funcionamiento de los principales elementos que forman un sistema de comunicación digital actual y los modelos de canal que se utilizan para modelar transmisiones por el medio inalámbrico.

- Desarrollar una herramienta software que permita simular las distintas etapas y operaciones que son necesarias para transmitir una trama de datos de manera inalámbrica, desde que se genera hasta que es decodificada en el dispositivo receptor.

- Implementar una aplicación gráfica fácil de usar e intuitiva para que el usuario pueda realizar estas simulaciones con las opciones que desee, y de manera que los resultados e información más relevantes sean mostrados de forma visual para ayudar a que el usuario sea capaz de entender las diferencias entre los esquemas de comunicación considerados y el impacto de los diferentes parámetros que se pueden configurar.

1.5 Estructura de la memoria

La memoria de este proyecto se divide en los siguientes capítulos:

- **Introducción**. En este capítulo se introduce el proyecto, se describen los motivos para llevarlo a cabo y se explican los objetivos del mismo.
- **Fundamentos de los sistemas de comunicaciones inalámbricas**. Se explican todos los conceptos y técnicas necesarios para entender el alcance y desarrollo del proyecto.
- **Fundamentos tecnológicos**. En este capítulo se describen las diferentes herramientas y tecnologías que ha sido necesario utilizar para llevar a cabo el proyecto.
- **Metodología, recursos y planificación**. Se explica la metodología elegida para el desarrollo del proyecto, así como los recursos utilizados y la planificación inicial que se ha realizado para el proyecto.
- **Análisis**. Se especifican los diferentes requisitos de la herramienta desarrollada y la arquitectura que va a seguir el proyecto.
- **Desarrollo**. Se explica la implementación llevada a cabo en cada una de las iteraciones en los que se ha divido el proyecto.
- **Funcionamiento de la herramienta**. En este capítulo se muestra la interfaz gráfica y su funcionamiento.
- **Conclusiones y trabajo futuro**. Se explican las conclusiones finales que se pueden extraer tras la realización del proyecto y las posibles líneas de trabajo a explorar en un futuro.

Capítulo 2

Fundamentos de sistemas de comunicaciones inalámbricas

Cuando nos referimos a un sistema de telecomunicación hablamos de un proceso que consta de cinco elementos básicos, que son el emisor, el medio, el mensaje, el lenguaje o protocolo de transmisión y el receptor 2.1. Si alguno de estos elementos deja de existir no será posible establecer correctamente la comunicación [3].

- Emisor: es el encargado de generar y enviar el mensaje de interés. Prepara la información para que pueda ser enviada por el canal, tanto en calidad (adecuación a la naturaleza del canal) como en cantidad (amplificando la señal).
- Medio: canal o medio físico por el cuál se realiza el proceso de comunicación.
- Mensaje: es la información que tratamos de transmitir, y su naturaleza puede ser analógica o digital. Lo importante es que llegue integro y con fidelidad para que puede ser interpretado de forma correcta en el destino.
- Lenguaje o protocolos de transmisión: son el conjunto de códigos, símbolos y reglas que gobiernan la transmisión de la información. Por ejemplo, en la transmisión oral entre personas se puede usar el español, el inglés o cualquier otro idioma.
- Receptor: es la entidad a la cual el mensaje está destinado. Puede ser una persona, grupo de personas, un dispositivo artificial, etc.

En la figura 2.1 se aprecia el proceso por el cual un emisor envía un mensaje con información a través de un canal de comunicación y que llega hasta el destinatario de la información.

Para simular este tipo de sistema de comunicación se llevará a cabo el desarrollo de una herramienta *software* que se centrará en la implementación de las operaciones relacionadas

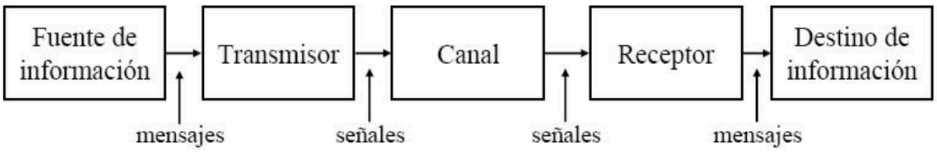

Figura 2.1: Esquema de un sistema de comunicaciones básico

con la capa física del modelo *Open System Interconnection* (OSI): codificación de canal, modulación, transmisión de la señal sobre el medio físico, demodulación y decodificación, cuyo esquema se puede ver en la figura 2.2.

Para comenzar, la información que se quiere transmitir será modulada, enviada en forma de señal, para adaptarse al medio por el que se enviarán los datos. Esta señal transmitida, al ser enviada a través de un canal, puede sufrir alteraciones, lo que provocará que el receptor le asigne un mensaje erróneo a la señal recibida, ocasionando errores irreversibles en el destino. El número de errores dependerá de las características del canal considerado.

Por este motivo, se impone la necesidad de emplear algún mecanismo de protección sobre las señales transmitidas que reduzca la probabilidad de error. Este mecanismo es conocido como codificación de canal. Esta operación consiste añadir información redundante estructurada a la secuencia binaria fuente antes de ser transmitida. De esa forma, los bits redundantes pueden ser utilizados en el receptor para detectar y corregir errores. La ventaja de esta solución es el aumento de la fiabilidad del sistema de transmisión, debido a la ya citada reducción de errores en las señales. Sin embargo, presenta el inconveniente de que, al añadir redundancia, se reduce la proporción de información útil respecto a la cantidad de información transmitida. Esto supone que la velocidad de transmisión efectiva se reduce, mientras que la cantidad total de energía que se necesita utilizar es mayor. Por ello, es importante la elección de un determinado código dependiendo de las características del canal, tratando de maximizar el beneficio con la mínima cantidad de información redundante.

En las siguientes secciones, se explicarán en detalle los principales componentes de la capa física de un sistema de comunicaciones digital: codificador de canal, esquema de modulación, demodulador y decodificador de canal. Además, se describirán los diferentes canales que se pueden considerar dependiendo del medio físico elegido. Finalmente, se procederá a comentar la métrica que se suele utilizar para evaluar el rendimiento y prestaciones de los sistemas de comunicaciones digitales.

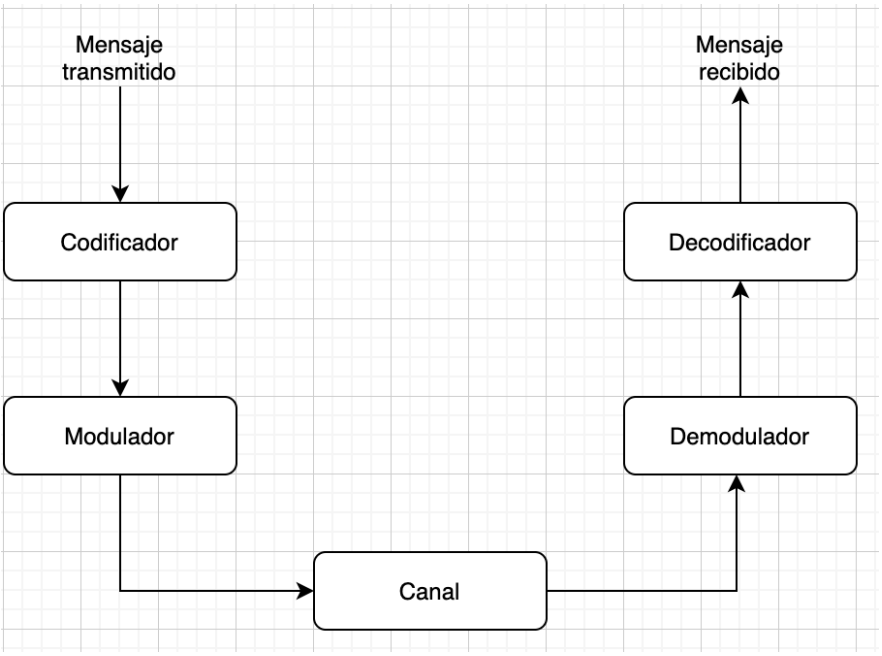

Figura 2.2: Principales elementos que conforman los distintos bloques de un sistema de comunicaciones.

2.1 Codificación de canal

En este apartado se hablará de la codificacion de canal, técnica utilizada para la corrección de errores en la transmisión de información, así como de diferentes tipos y de cómo funcionan cada uno.

2.1.1 Perspectiva histórica

La teoría de la información surgió a finales de la Segunda Guerra Mundial, en los años cuarenta. Se puede considerar como su punto de inicio, la publicación del artículo de Claude E. Shannon en el Bell System Technical Journal en 1948, titulado "Una teoría matemática de la comunicación". En esta época se buscaba utilizar de manera más eficiente los canales de comunicación, enviando una cantidad de información por un determinado canal y midiendo su capacidad, es decir, la velocidad máxima a la que se puede transmitir un mensaje con una probabilidad de error despreciable. En definitiva, se buscaba la transmisión óptima de los mensajes. Esta teoría fue el resultado de trabajos comenzados en la década 1910 por Andrei A. Markovi, a quien le siguió Ralp V. L. Hartley en 1927, quien fue el precursor del lenguaje binario. A su vez, Alan Turing en 1936, realizó el esquema de una máquina capaz de tratar información con emisión de símbolos, y finalmente Claude Shannon, matemático, ingeniero electrónico y criptógrafo estadounidense, conocido como "el padre de la teoría de la informa-

ción", junto a Warren Weaver, contribuyó en la culminación y el asentamiento de la Teoría Matemática de la Comunicación de 1949, que hoy es mundialmente conocida por todos como la Teoría de la Información [1]. Weaver consiguió mejorar el planteamiento inicial, creando un modelo simple y lineal: Fuente/codificador/mensaje-canal/decodificador/destino.

Con el modelo de la teoría de la información se estudia la forma más eficiente, rápida y segura de codificar un mensaje, sin que la presencia de ruido o las distorsiones introducidas por el canal impidan la correcta recepción del mensaje. El estudio de esquemas de codificación eficientes para lograr comunicaciones fiables a través de canales con ruido se denomina teoría de la codificación.

El estudio realizado por Shannon nos ofrece un mecanismo eficaz para evaluar las prestaciones de un esquema de codificación: cuanto más se aproxime a la capacidad del canal, más adecuado resulta ese esquema. Sin embargo, el teorema de Shannon no ofrece una descripción de cómo construir códigos eficientes [4]. A lo largo de los años se han desarrollado una gran variedad de esquemas de codificación que buscaban alcanzar la capacidad predicha por Shannon en su trabajo. Aunque estos códigos presentan características claramente diferenciadas, se pueden agrupar en dos grandes bloques: códigos bloque y códigos convolucionales.

En los sistemas actuales de comunicación, especialmente en los estándares inalámbricos, se utilizan principalmente dos tipos de códigos: los códigos turbo y los códigos LDPC. Como se explicará en las siguientes secciones, los códigos turbo pueden verse realmente como una versión avanzada de los códigos convolucionales, mientras que los códigos LDPC son códigos bloque lineales con unas características específicas que les proporcionan una gran capacidad correctora.

En las siguientes secciones, se presentarán las características de los dos grandes tipos de códigos y se detallarán los diferentes esquemas de codificación que se han considerado para el desarrollo del simulador.

2.1.2 Códigos bloque

Un código bloque, $C(n, k)$, consiste en dividir el mensaje fuente en bloques separados de k bits que son codificados de forma independiente de acuerdo a una función de asignación. En el caso de códigos bloque binarios, consiste en asignar una palabra o vector de n bits a cada una de las 2^k posibles palabras fuente de k bits.

Debido a que los distintos códigos añaden bits redundantes para la posterior corrección de errores, es importante definir el concepto de *rate* o tasa de un código bloque, que es la proporción de bits de información, es decir, que no son redundantes, respecto del total de bits de la palabra codificada. Por lo que se define como,

$$R = k/n. \tag{2.1}$$

Desde un punto de vista práctico, un código bloque puede interpretarse como una tabla en la que, para cada posible mensaje de entrada, tenemos almacenado su palabra código correspondiente, como se puede apreciar en la figura 2.3.

Mensajes K = 3 bits		Palabras código n = 6 bits	
x_0	000	0 0 0 0 0 0	c_0
x_1	001	1 1 0 1 1 0	c_1
x_2	010	0 1 0 1 0 1	c_2
x_3	011	1 0 0 0 1 1	c_3
x_4	100	1 0 1 0 1 0	c_4
x_5	101	0 1 1 1 0 0	c_5
x_6	110	1 1 1 1 1 1	c_6
x_7	111	0 0 1 0 0 1	c_7

Figura 2.3: Ejemplo de una tabla de asignación para un código bloque (6,3)

Códigos bloque lineales

Un código bloque es lineal si los bits de cada palabra código resultante se pueden obtener como combinación lineal de los bits de la palabra fuente correspondiente. Es decir,

$$c_j^{(i)} = \sum_{z=1}^{K} g_{z,i} m_z^{(i)} \qquad \forall j = 1, \ldots, n; \quad i = 1, \ldots, 2^k, \tag{2.2}$$

donde $c^{(i)} = [c_1^{(i)}, \ldots, c_n^{(i)}]$ representa la palabra código correspondiente a la palabra fuente $m^{(i)} = [m_1^{(i)}, \ldots, m_k^{(i)}]$, y la operación sumatorio implica la suma binaria. Además, los pesos $g_{z,i}$ determinan las palabras código válidas del conjunto de todas las posible palabras binarias de longitud n.

En el caso de códigos bloque lineales, la suma de dos palabras código cualesquiera es también una nueva palabra código [5]. Esta propiedad se puede comprobar considerando el código bloque mostrado en la figura 2.3. En este caso, observamos que tras la suma de cualquier combinación de palabras código obtendremos una nueva palabra que forma parte del código. Como ejemplo, cogiendo las palabras código c_1 y c_2 obtendremos:

$$c_1 + c_2 = [110110] + [010101] = [100011] = c_3.$$

De esta forma, podemos afirmar que este código es un código bloque lineal.

Una particularidad de todos los códigos bloque lineales es que una de sus palabras código debe ser compuesta por todo 0s. Esto se debe a que, como ya hemos explicado, la suma de dos palabras código cualesquiera debe dar como resultado otra palabra código, y en el caso de sumar una palabra código consigo misma siempre obtendremos la palabra compuesta por todo 0s. Por ejemplo:

$$c_4 + c_4 = [101010] + [101010] = [000000] = c_0.$$

Viendo este ejemplo podría pensarse que todos los códigos bloque son a su vez códigos bloque lineales, pero eso no es cierto. Siguiendo con el anterior ejemplo, si se sustituyera la palabra código $c_1 = [110110]$ por $[111110]$, entonces la suma anterior $c_1 + c_2$ daría como resultado $[101011]$ que no se correspondería con ninguna otra palabra código. Por tanto, estaríamos hablando de un código bloque no lineal. En ese caso, los bits de las palabras del código son obtenidos aplicando operaciones no necesariamente lineales sobre los bits de la palabra fuente correspondiente.

Matriz generadora del código

Teniendo en cuenta que un código bloque lineal, $C(n, k)$, forma un subespacio k-dimensional del espacio de vectores de F_2^n, se pueden encontrar k palabras código linealmente independientes que conformarán la base del subespacio del código.

Para determinar los elementos de la base de un código, es necesario recurrir al concepto de base canónica, que es una colección de vectores donde cada uno está compuesto por todo 0 exceptuando uno de ellos que toma el valor 1. Teniendo en cuenta esto, las palabras código que formarán la base del código serán aquellas que se corresponden con las palabra fuente de la base canónica. En nuestro código de ejemplo, se puede apreciar la base canónica y las palabras código correspondientes en la figura 2.4.

PALABRAS FUENTE(BASE CANÓNICA)	PALABRAS CÓDIGO
[1 0 0]	[1 0 1 0 1 0]
[0 1 0]	[0 1 0 1 0 1]
[0 0 1]	[1 1 0 1 1 0]

Figura 2.4: Palabras código que representarían la base del código bloque lineal usado de ejemplo.

Ahora ya podemos hallar la matriz generadora de código, G, con dimensión $k \times n$ y cuyas filas se corresponden con las k palabras código que forman partde de la base. Con esta matriz, se pueden obtener todas las palabras código de C de forma sencilla. Para ello, hay que multiplicar cada palabra fuente m por la matriz generadora, G, dando como resultado su palabra código asociada c:

$$c_i = m_i G \qquad \forall\, c_i \in C(n,k). \tag{2.3}$$

De esta forma, la función generadora se puede utilizar para realizar directamente la operación de codificación. Por ejemplo:

$$[011]\, G = [100011].$$

Además existe la llamada matriz de control de paridad, H, con dimensión $(n–k)\times n$, que nos permite comprobar si una palabra es una palabra código o no y es utilizada para la decodificación, detección y corrección de errores. Se cumple que todo vector de la matriz generadora, G, es ortogonal a las filas de H; y que todo vector que es ortogonal a las filas de H está en G, es decir, $GH^t = 0$. Teniendo en cuenta esta propiedad, se cumple que:

$$cH^t = 0, \tag{2.4}$$

donde c se corresponde con una palabra código cualquiera y H^t con la matriz control de paridad traspuesta, cuya dimensión sería $n\times(n–k)$. El resultado de este producto es un vector fila con todos sus elementos a 0 ya que la palabra c es una palabra código. Sin embargo, cuando se multiplica una palabra que no es código por H^t, el resultado da distinto de cero. De esta forma, la matriz control de paridad puede ser utilizada para detectar y corregir errores en la palabra recibida.

Si la palabra código de un código bloque contiene los k bits de la palabra de entrada al codificador en sus k primeras o últimas posiciones, entonces estamos hablando de un código bloque que es sistemático. Estos k bits correspondientes con los de la palabra fuente se denominan bits de información, mientas que los $n–k$ bits restantes de la palabra código son los llamados bits de paridad o redundancia. En este tipo de códigos bloque, la matriz generadora tiene la forma siguiente:

$$G = [I_{k\times k}\ P_{k\times(n-k)}], \tag{2.5}$$

donde $I_{k\times k}$ es la matriz identidad de dimensión k, y $P_{k\times(n-k)}$ representa los bits de redundancia.

Otros aspectos importantes relacionados con los códigos bloque lineales son el peso y su distancia mínima. El peso de una palabra código es el número de elementos que difieren

de 0. La distancia entre dos palabras código es la distancia de Hamming entre ellos, que se traduce en el número de elementos en los que se distinguen ambas palabras código. Por tanto, la distancia mínima de un código lineal es la mínima distancia de Hamming entre dos palabras código cualesquiera.

$$d_{\text{min}} = d_H(c_i, c_j) \qquad \forall c_i, c_j \in C(n,k), \tag{2.6}$$

donde $d_H(\cdot)$ representa la distancia de Hamming entre palabras binarias. En el caso particular de los código bloque lineales, se puede demostrar que la distancia mínima se puede calcular como el peso mínimo entre todas las palabras código distintas de la palabra con todos sus valores a 0. Es decir:

$$d_{\text{min}} = w(c_i) \qquad \forall c_i \in C(n,k),\ c_i \neq 0. \tag{2.7}$$

Códigos Hamming

El código de Hamming, cuyo inventor fue Richard Hamming, es un ejemplo de código bloque lineal sencillo y práctico. Es un código cuya distancia mínima es de 3, esto significa que el número mínimo de cambios de bits necesarios para pasar de cualquier palabra código a cualquier otra palabra código es 3, lo que le permite detectar hasta dos errores de bit pero solo corregir uno [6].

En los códigos de Hamming, las variables k y n están directamente relacionadas a través de una tercera variable, que llamaremos l, y que determina el número de bits de redundancia que se añaden, i.e., $l = n - k$. De hecho, los códigos de Hamming se definen a partir de la variable l a diferencia de otros códigos bloque que utilizan k y n. En el caso de los códigos de Hamming, para un determinado valor de l, se cumple de forma general lo siguiente:

$$n = 2^l – 1$$

$$k = n – l$$

$$d_{\text{min}} = 3.$$

Al igual que en cualquier código, el *rate* de los códigos de Hamming se calcula de la siguiente manera:

$$R = \frac{k}{n} = \frac{n-l}{l}. \tag{2.8}$$

Como se puede apreciar en la ecuación anterior, a medida que aumentamos la variable l, aumentará el *rate* de código. Esto se traduce en que aumenta la proporción de bits de información enviados respecto a los bits de redundancia. Es decir, las capacidades de detección y

corrección del código se mantienen a pesar de añadir menos redundancia. Como contrapartida, el tamaño de las palabras fuentes y código aumenta significativamente, lo que supone un mayor coste computacional tanto en la operación de codificación como decodificación. Además, al transmitir palabras código más largas, el número de errores que pueden afectar a cada palabra transmitida será en general mayor, superando las capacidades del código. Por esa razón, se suelen utilizar los códigos de Hamming con valores de l relativamente pequeños.

Hoy en día cuando hablamos del código de Hamming nos estamos refiriendo, normalmente, al código Hamming (7, 4) introducido en 1950. Siguiendo la relación entre variables en los códigos de Hamming anteriormente comentada y teniendo palabras código de $n = 7$ bits y palabras fuente de $k = 4$ bits, se puede deducir que $l = 3$. Este código añade, por tanto, tres bits de redundancia por cada cuatro bits de información. Los bits de redundancia o paridad se ubican en las posiciones 1, 2 y 4. Como ya se ha comentado, la distancia mínima del código es 3 y, por lo tanto, puede corregir errores en un solo bit. Cuando hay más de un error, la palabra código se corrige de manera incorrecta confundiéndose con otra, por tanto, no se sabrá si ha ocurrido de esta manera o si la palabra código se ha corregido correctamente.

A diferencia de la mayor parte de códigos bloque lineales que se definían a partir de la matriz generadora, los códigos de Hamming se definen a partir de la matriz control de paridad, donde cada columna se corresponde con la representación binaria de la posición que ocupa, excluyendo la columna todo ceros. Esto se traducirá en una serie de ventajas de este tipo de códigos en la fase de decodificación, como veremos en la sección 2.4. Para el caso de un código de Hamming (7, 4), su matriz control de paridad sería la siguiente:

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{2.9}$$

Sin embargo, estos códigos tiene el inconveniente de que su matriz control de paridad no se corresponde con la matriz control de paridad estándar para un código bloque lineal sistemático, ya que las primeras columnas de la matriz de estos códigos debería coincidir con la matriz identidad. Esto supone un problema a la hora de obtener la matriz generadora necesaria para la codificación. Sin embargo, se puede hallar la matriz generadora de cualquier código de Hamming a partir de la matriz control de paridad de un código bloque lineal sistemático equivalente.

2.1.3 Códigos convolucionales

Los códigos convolucionales fueron introducidos en 1955 por Peter Elias. En 1961, Wozencraft y Reiffen describieron el primer algoritmo práctico de decodificación para códigos convolucionales. Más tarde, en 1967, Andrew Viterbi, inventó el algoritmo que lleva su apelli-

do, y que usa una decodificación basada en enrejado invariante en el tiempo para decodificar códigos convolucional con un coste y complejidad razonables [7].

Estos códigos se presentan como una alternativa a los códigos bloque y, al igual que ellos, los códigos convoluciones también pueden ser códigos lineales. De hecho, muchos sistemas de comunicaciones utilizan simultáneamente códigos bloque y códigos convolucionales. Actualmente, pueden ser encontrados en diferentes perfiles de comunicación para sistemas WiFi, sistemas de telefonía móvil, etc. La diferencia con los códigos bloque es que, en los códigos convolucionales, los bits código se obtienen, además de con la información actual, con parte de la información anterior en tiempo. Por ello, estos códigos incluyen registros de desplazamiento en las cuáles habrá datos anteriores que influirán en las siguientes operaciones. De esta forma, una misma secuencia de entrada puede generar una secuencia código diferente debido al estado del registro, que depende de las entradas anteriores.

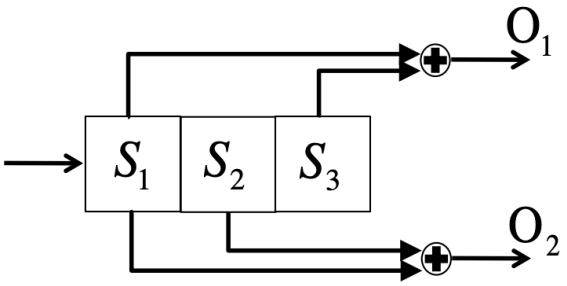

Figura 2.5: Diagrama de bloques de código convolucional

La figura 2.5 se corresponde con el diagrama de bloques de un código convolucional. La primera flecha de la izquierda representa por donde se introducen los bits de entrada mientras que O_1 y O_2 serían los bits de salida. En este ejemplo, se asume que entra un único bit fuente en cada instante de tiempo, es decir, $k = 1$ y $n = 2$. Además, en el diagrama de bloques se puede observar un registro de desplazamiento con tres posiciones, S_1, S_2, S_3, donde S_1 representa la entrada actual, mientras S_2 y S_3 serían las posiciones correspondientes a la memoria del código y almacenan valores correspondientes a entradas previas. Los bits de salida se calculan a partir de los bits del registro, es decir, como combinaciones lineales del bit de entrada y los bits almacenados en memoria. Para este ejemplo, como indica el diagrama, O_1 sería el resultado de sumar los bits de S_1 y S_3; mientras que O_2 resultaría de la suma de S_1 y S_2. Como por cada bit de entrada se obtienen dos bits de salida, este código convolucional tendría un rate:

$$R = k/n = 1/2. \tag{2.10}$$

Además, a partir de los parámetros anteriores, se puede definir la matriz generadora de código, donde las filas se corresponden con el número de registros de desplazamiento, mien-

tras que las columnas, con el número de salidas. Lo habitual es representar esta matriz en octal, de manera que la representación de la matriz generadora del código de la figura 2.5 es $G = (5, 6)_8$.

Otro parámetro muy importante de un código convolucional es lo que se llama la longitud restringida, que se corresponde con la longitud del registro de desplazamiento o, de forma más precisa, el número de instantes de tiempo que pasa desde que un bit entra al registro hasta que desaparece. En el ejemplo presentado, la longitud restringida del código convolucional sería $M = 3$.

Para ilustrar como se realizaría la operación de codificación en un código convolucional, vamos a utilizar el código de ejemplo correspondiente al diagrama de bloques de la figura 2.5. Vamos a suponer que se necesita codificar la secuencia fuente 10011. En el momento que se inicia la codificación estamos en el estado 000 o estado inicial. El primer bit que se introduce es un 1, por tanto, pasamos al estado 100, obteniendo como salida 11. El siguiente bit fuente se corresponde con un 0, el estado pasa a ser 010, y la salida correspondiente 01. Si continuamos el proceso llegaríamos a la siguiente salida del codificador 11 01 10 11 10. El estado final del codificador sería 110, habría que introducir tres 0 para reiniciar el estado y que se pueda aplicar en futuras codificaciones. Esto es lo que se llama codificación con terminación y, es habitualmente, empleada cuando se usan códigos convolucionales.

Los diferentes estados con sus correspondientes salidas para este ejemplo se pueden ver de manera muy sencilla en la siguiente tabla:

Registro	Secuencia Salida
000	00
100	11
010	01
001	10
110	10
011	11
101	01
111	00

Otro aspecto muy interesante en los códigos convolucionales es el diagrama de estados. Un código convolucional se puede interpretar como una máquina de estados finita con su correspondiente representación en forma de diagrama de estados. En nuestro ejemplo, el diagrama de estados asociado a ese código convolucional se representa en la figura 2.6. Como se puede apreciar, este diagrama está compuesto por cuatro posibles estados y dos transiciones para cada estado. En general, el número de potenciales estados y de transiciones depende de

los parámetros k y M del código. En general, se cumple que:

$$\text{nº de posibles estados} = 2^{(M-1)k},$$
$$\text{nº de transiciones} = 2^{k}.$$

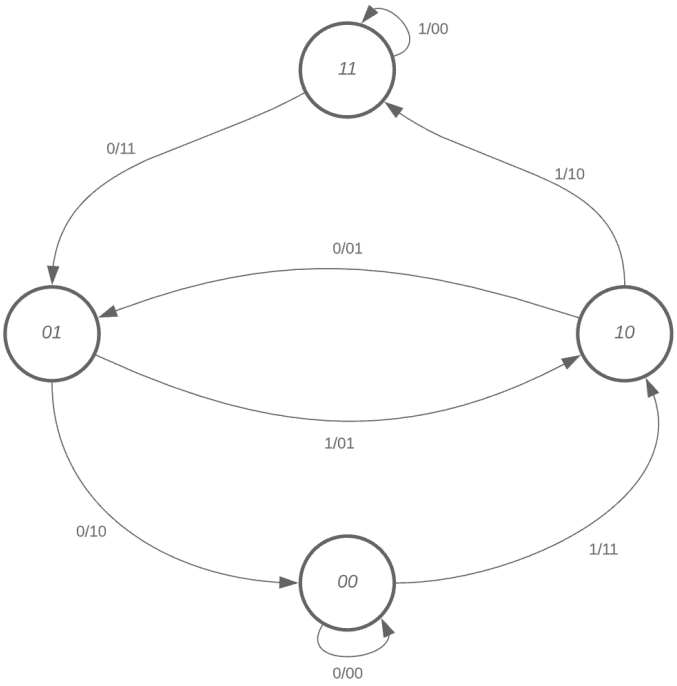

Figura 2.6: Diagrama de estados correspondiente al codificador convolucional de la figura 2.5

La ventaja de este tipo de diagramas es que se puede apreciar de forma muy visual y sencilla los posibles estados en los que se puede encontrar el sistema, representados con un círculo con sus valores en el interior, siendo en este caso 00, 01, 10 u 11. Cada vez que se introduce un nuevo bit al codificador, el valor de los registros se actualiza, pudiéndose producir una transición entre estados. Estas transiciones son indicadas por medio de una flecha del estado en el que estamos hacia el nuevo estado, junto con la siguiente notación X/XX, en donde el primer bit se corresponde con el bit de entrada y los otros dos bits con la salida que se produciría con esos valores en los registros. Por poner un ejemplo, imaginemos que el estado actual del registro es de 01 y el próximo bit de entrada es un 1. Siguiendo el diagrama de estados, la salida correspondiente, después de que el bit de entrada sea 1 con un registro de 01, sería de 01, y el nuevo valor del registro sería 10.

Otra posible opción para representar los diferentes estados del codificador convolucional de una forma más gráfica, así como las posibles transiciones entre estados, es el llamado diagrama de trellis, representado en la Figura 2.7. El diagrama de trellis se puede ver como un

diagrama de estados que se despliega en el tiempo, de tal forma que cada etapa del trellis se corresponde con un instante de tiempo donde se indican los posibles estados del codificador con sus correspondientes transiciones y salidas asociadas. De esa forma, es posible observar los posibles caminos en la fase de codificación a través del tiempo y a medida que entra nueva información en el registro del código.

Como se puede apreciar en la figura 2.7, los nodos en el diagrama de trellis representan diferentes estados del registro del código convolucional, y se suelen ordenar verticalmente. Las transiciones entre estados son representadas mediante flechas con su correspondiente secuencia de salida. Para este caso, cuando el bit de entrada es un 0, se emplea una flecha continua mientras que, cuando la entrada se trata de un 1, la flecha es discontinua.

Como ya se ha explicado anteriormente, el estado inicial del registro es necesariamente 00. A medida que se introducen los bits fuente al codificador, se van desplegando las diferentes etapas del diagrama de trellis. Esto nos permite indicar los estados alcanzables en cada etapa del trellis en función de los estados de la etapa anterior y de las posibles transiciones. Finalmente, llegaremos a un punto en que todos los estados son alcanzables y el diagrama de trellis se repite para cada transición.

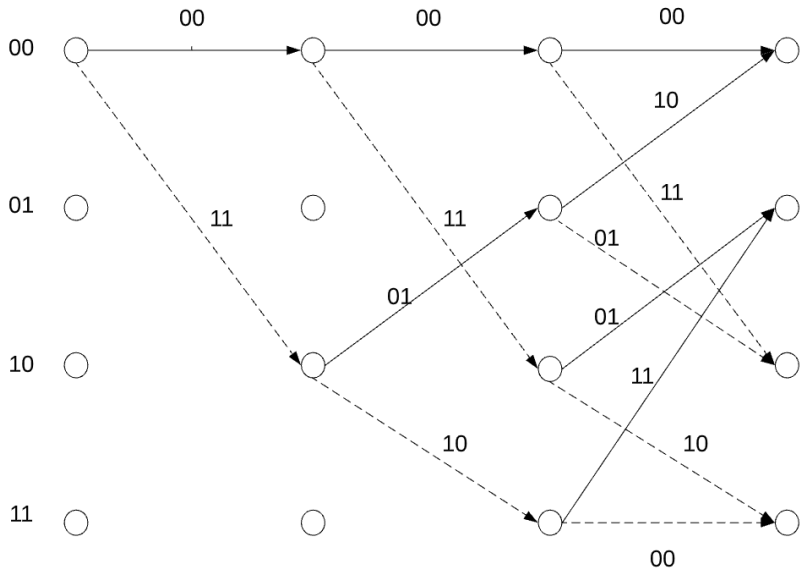

Figura 2.7: Diagrama de trellis correspondiente con el codificador convolucional de la figura 2.5

El diagrama de trellis se puede utilizar además para calcular otro de los parámetros fundamentales de un código convolucional: la distancia libre, cuyo interpretación es equivalente al concepto de distancia mínima de un código de bloque. Simplemente, tenemos que determinar el peso mínimo de las palabras códigos que se generan como consecuencia de un bucle del

diagrama de trellis que comienza en el estado inicial 00 y retorna a ese mismo estado tras un número finito de transiciones. En el ejemplo de código convolucional de la figura 2.7, la distancia libre sería:

$$d_{\text{free}} = 4. \tag{2.11}$$

Este parámetro es muy importante ya que, como se ha comentado para códigos bloque, determina las capacidades detectores y correctoras del código.

2.1.4 Turbo códigos

Surgen en 1993 cuando un grupo de investigadores franceses presentan una nueva clase de códigos que conseguían alcanzar una probabilidad de error muy próxima al límite establecido por Shannon [1]. Posteriores investigaciones ratificaron estos resultados preliminares y se inició una carrera para mejorar las prestaciones y viabilidad práctica de estos códigos. De esta forma, a finales de los 90, los turbo códigos empezaron a adaptarse a los diferentes sistemas de comunicaciones existentes siendo, hoy en día, utilizados en la mayoría de estándares de comunicación inalámbrica de largo y medio alcance, incluyendo comunicaciones por satélite, redes de telefonía móvil o WiFi [8].

Los turbo códigos están formados por dos codificadores convolucionales de tasa 1/2 concatenados en paralelo y separados por un entrelazador. El esquema habitual para este tipo de códigos se puede observar en la figura 2.8. Como se puede apreciar, uno de los dos codificadores, $r_i^{(0)}$, opera directamente sobre los datos de entrada, mientras que el segundo codificador, $r_i^{(1)}$, lo hace sobre una versión permutada de los datos. Esta permutación es llevada a cabo por el entrelazador que cambia bloques de bits de posición de acuerdo con una permutación generada de manera aleatoria.

A la salida del codificador, obtenemos tres vectores de bits que se concatenan antes de ser transmitidos. Por un lado, $x_i^{(0)}$ se corresponde con la salida sistemática (salida directa de información sin pasar por el codificador), mientras que $x_i^{(1)}$ y $x_i^{(2)}$ son las secuencias de bits de paridad generadas por los dos codificadores convolucionales que conforman el turbo código. De esta forma, la tasa de un turbo código es de 1/3 debido a que los dos codificadores que lo forman son de tasa 1/2.

2.1.5 Códigos LDPC

Los códigos LDPC (del inglés low density parity check) originalmente fueron propuestos por Robert Gallager en 1963 en su tesis doctoral en el MIT [9]. Estos códigos fueron olvidados debido a la insuficiente capacidad de procesamiento, ya que, su rendimiento era muy bueno cuando se consideraban tramas muy largas de datos. Sin embargo, el coste computacional para codificar esas tramas de datos era inabordable en aquella época.

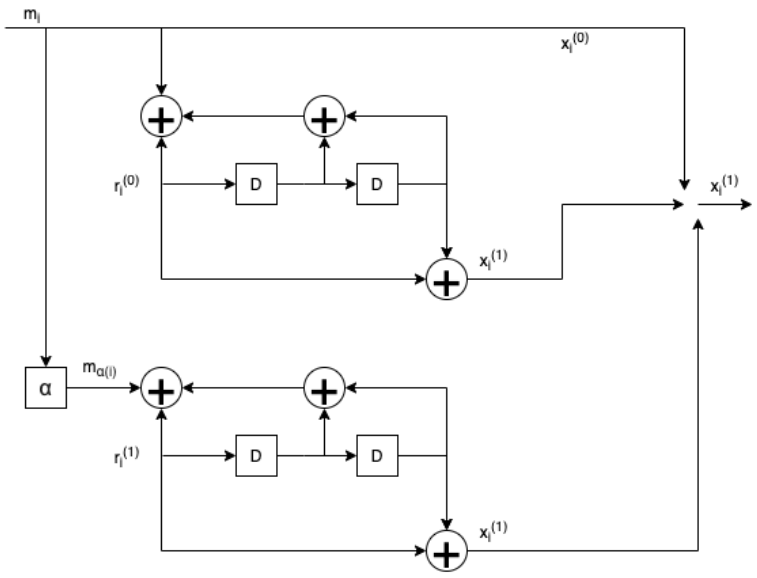

Figura 2.8: Diagrama de bloques de un turbo código

Tras la aparición de los turbo códigos en 1995, los códigos LDPC fueron redescubiertos en 1995 por David J.C. MacKay y Radford M. Neal [10]. Estos códigos han mostrado unas prestaciones prácticas en términos de corrección de errores que se aproximan al límites teórico definido por Shannon para cualquier tasa y para tamaños de palabra fuente relativamente grandes. El hecho de utilizar una algoritmo de decodificación con una complejidad aceptable (basado en la idea de *belief propagation*) y el aumento de la capacidad de cómputo de los sistemas actuales han hecho posible su implementación práctica. Hoy en día, se utilizan en aplicaciones como la recuperación de los paquetes perdidos en la transmisión de datos a través de Internet y en diferentes sistemas de comunicación inalámbrica como WiMax o en el estándar para televisión digital terreste, i.e., DVB-T (*Digital Video Broadcasting- Terrestrial*).

Los códigos LDPC son códigos bloque lineales (N, k), cuya propiedad esencial es que están definidos por una matriz control de paridad dispersa. Esto implica que esta matriz está compuesta por muy pocos elementos distintos de 0, en donde, las posiciones de los valores no nulos son normalmente escogidos aleatoriamente. Un aspecto muy importante en este tipo de códigos es que pueden ser representados mediante un grafo bipartito, ya que a la hora de la decodificación, los mejores algoritmos y más eficientes se basan en estas representaciones. Este grafo bipartito es representado con N nodos en el lado izquierdo, denominados nodos variables (*variable nodes*) y $l = N - k$ nodos a la derecha, llamados nodos de paridad (*check nodes*). Estos dos tipos de nodos pueden conectarse por medio de una arista. Las conexiones entre estos dos tipos de nodos determinan las l ecuaciones de paridad que deben cumplir los bits de una palabra binaria de longitud N para ser una palabras código y vienen dadas direc-

tamente por las entradas de la matriz control de paridad que define el código. En la figura 2.9 puede verse un ejemplo de representación de un código LDPC en forma de grafo bipartito.

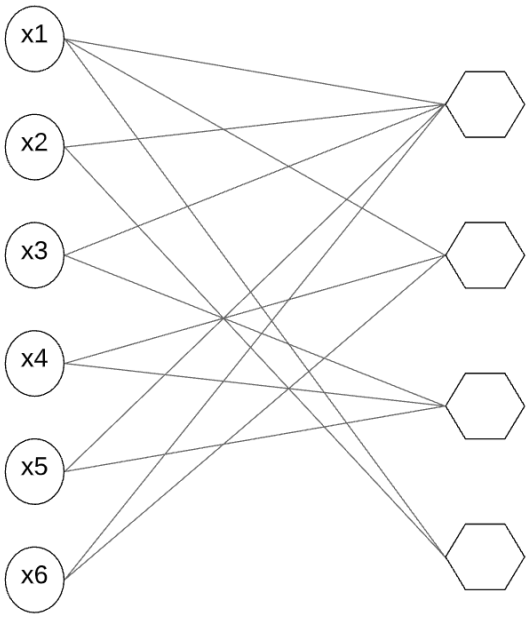

Figura 2.9: Ejemplo de gráfico bipartito para un código LDPC

Las ecuaciones de paridad correspondientes al código LDPC que se representa en la figura 2.9 serían por tanto las siguientes:

$$\begin{aligned} x_1 + x_2 + x_3 + x_5 + x_6 &= 0 \\ x_1 + x_4 + x_6 &= 0 \\ x_3 + x_4 + x_5 &= 0 \\ x_1 + x_2 &= 0. \end{aligned}$$

De forma similar a lo que ocurría en el caso de códigos bloque lineales, estas ecuaciones de paridad pueden utilizarse para la detección y corrección de errores introducidos por el canal de transmisión. En el caso de los códigos LDPC se han diseñado algoritmos eficientes que realizan esta tarea aprovechando la representación en forma de grafo bipartito.

Otra manera equivalente de definir un código LDPC es directamente a través de su matriz control de paridad, H, donde todos los valores de esta matriz dependen directamente de las aristas definidas en el grafo bipartito, ya que, todo gráfico bipartito tiene asociado un código lineal. Así, una arista A_{ij} del grafo bipartito que conecta el nodo variable i y el nodo de paridad j, se corresponderá con un valor de uno en la posición correspondiente de la matriz H, i.e., $H_{j,i} = 1$. De esta forma, la matriz control de paridad para el código LDPC representado en la

figura 2.9 sería:

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{2.12}$$

2.2 Modulación

La modulación es la técnica que se utiliza para convertir la secuencia binaria que queremos transmitir en señales adecuadas para ser enviadas través del medio físico considerado. Para ello, se utiliza una onda portadora y la información es modulada en alguna de las características de esta onda. Esto se traduce en un mejor aprovechamiento del canal por el que se realizará la comunicación, además de minimizar el número de errores que se pueden dar.

En este apartado nos centraremos específicamente en modulaciones digitales, en las que se realiza un mapeo entre la secuencia de bits y la forma de onda analógica que se adapta a las características del canal. Dentro de las modulaciones digitales, podemos distinguir entre las modulaciones en banda base y la modulaciones paso banda. Cuando las señales que se transmiten se encuentran en su frecuencia original, entonces se trata de una modulación en banda base, entre las que se puede destacar *Pulse-Amplitude Modulation* (PAM); mientras que, en el caso de las modulaciones paso banda, su representación espectral se centra en torno a la frecuencia de la onda portadora, de los cuáles se explicarán las modulaciones *Phase-Shift Keying* (PSK) y QAM.

A la hora de elegir una determina modulación es importante tener en cuenta una serie de características que definen las propiedades de la modulación: velocidad de transmisión, energía media de símbolo, ancho de banda requerido, eficiencia espectral, probabilidad de error de bit, ...

En el caso de la velocidad de transmisión, este parámetro está directamente relacionado con el período de la onda portadora utilizada para realizar la modulación, T, y con el número de niveles de la modulación, M, es decir, el número de señales disponibles para ser asignadas a los bits que queremos transmitir. De hecho, la velocidad de transmisión se calcula como:

$$v_t = \frac{\log_2(M)}{T} \text{ bits/s.} \tag{2.13}$$

La energía media de símbolo determina la cantidad de energía que es necesario utilizar para transmitir una determinada cantidad de información. Este parámetro depende específicamente de la modulación considerada, al igual que el ancho de banda necesario para transmitir la señal modulada.

Otro parámetro muy importante y utilizado para medir la calidad de una modulación digital es la eficiencia espectral. Este parámetro es una medida de la eficiencia con la que la modulación aprovecha el ancho de banda disponible en el canal. Se mide en bps/Hz y su fórmula general es:

$$\xi = \frac{v_t}{B} \text{ bits/s/Hz}, \tag{2.14}$$

donde T representa la tasa de transmisión, mientras que B es el ancho de banda del canal.

2.2.1 M-PAM

En este tipo de modulación, la información a transmitir se modula en la amplitud de la señal portadora, es decir, se utilizan diferentes amplitudes para transmitir las diferentes palabras binarias que tenemos a la entrada. Si el número de niveles o señales disponibles para la modulación PAM es M, en cada símbolo se podrán transmitir $k = \log_2(M)$ bits. En ese caso, las amplitudes asociadas a cada una de las M posibles palabras binarias de longitud k se determinan como:

$$s_i = a_i\sqrt{E_p} \qquad \forall i = 0, \dots M-1, \tag{2.15}$$

siendo $a_i = 2i - M$ la amplitud del i-ésimo símbolo y E_p la energía del pulso utilizada para modular. De acuerdo a este planteamiento, la figura 2.10 representa la constelación de los símbolos de la modulación para el ejemplo de una 4-PAM.

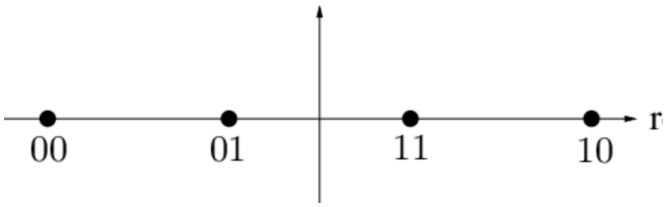

Figura 2.10: Constelación de la modulación 4-PAM

En el caso general, la energía media de símbolo para una M-PAM viene dada por la siguiente ecuación:

$$E_s = \frac{M^2 - 1}{3} E_p \text{ J}. \tag{2.16}$$

Por otro lado, sustituyendo en la fórmula general de la eficiencia espectral 2.14, es posible determinar este parámetro para modulaciones M-PAM como:

$$\xi = 2\log_2(M) \text{ bits/s/Hz}, \tag{2.17}$$

teniendo en cuenta que el ancho de banda requerido para la transmisión de señales PAM es $B = 1/2T$ Hz.

2.2.2 M-PSK

En la modulación digital de fase o PSK, la información a transmitir se modula en la fase de la señal portadora. De manera que, por ejemplo, para una 4-PSK se van a utilizar cuatro señales con diferente fase para transmitir palabras binarias de dos bits. La constelación de símbolos de esta modulación se muestra en la figura 2.11.

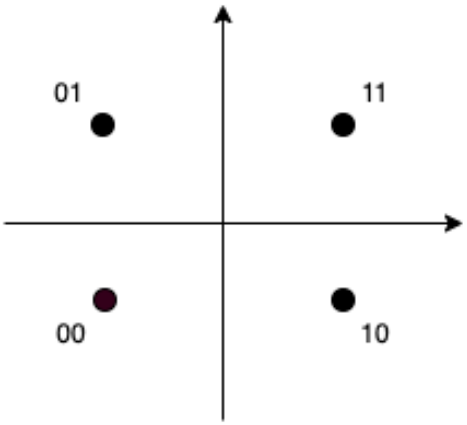

Figura 2.11: Constelación de la modulación 4-PSK

Para una M-PSK, las señales utilizadas para realizar la modulación se obtienen de la siguiente forma:

$$s_i = \left[\cos(\phi_i)\sqrt{\frac{E_p}{2}}, \sin(\phi_i)\sqrt{\frac{E_p}{2}}\right] \qquad \forall i = 0, \dots M-1, \tag{2.18}$$

donde ϕ_i representa las fase de la i-ésima señal de la modulación y E_p es, de nuevo, la energía del pulso utilizado para modular. Las fases para los diferentes símbolos se puede obtener como:

$$\phi_i = \frac{2\pi}{M} i \quad i = 0, \dots, M-1. \tag{2.19}$$

En el caso de PSK, la energía de todos los símbolos es siempre la misma y, por tanto, la energía media no depende del número de niveles. La fórmula para calcularla es:

$$E_s = \frac{E_p}{2} \text{ J}. \tag{2.20}$$

Por otra parte, la eficiencia espectral para el caso de una M-PSK viene dada por:

$$\xi = \log_2(M) \text{ bits/s/Hz}, \tag{2.21}$$

ya que se trata de una modulación paso banda y el ancho de banda requerido es $B = 1/T$ Hz.

2.2.3 M-QAM

En la modulación de amplitud en cuadratura o QAM, la información va a ser modulada en amplitud y fase. Por tanto, los símbolos de la modulación son obtenidos como:

$$s_i = [a_i \cos(\phi_i)\sqrt{\frac{E_p}{2}}, a_i \sin(\phi_i)\sqrt{\frac{E_p}{2}}] \qquad \forall i = 0, \dots M-1. \tag{2.22}$$

Es decir, esta modulación se puede interpretar como una combinación de las dos modulaciones anteriores, ya que se varía simultáneamente la amplitud y la fase de la señal portadora sinusoidal utilizada para modular. En QAM, el número de niveles, M, es habitualmente potencia de 4, y las modulaciones más habituales son 16-QAM, 64-QAM y 256-QAM.

La energía media de símbolo para una modulación M-QAM depende directamente de la disposición de los símbolos en el plano bidimensional. Por lo tanto, no existe una fórmula concreta para determinar este parámetro. Sin embargo, se puede calcular promediando la energía de cada símbolo en la constelación, es decir:

$$E_s = \frac{1}{M}\sum_{i=0}^{M-1} |s_i|^2 \quad \text{J}. \tag{2.23}$$

Mientras que su eficiencia espectral es la misma que la de la modulación M-PSK, es decir:

$$\xi = \log_2(M) \ \text{ bits/s/Hz}. \tag{2.24}$$

2.2.4 Ordenamiento de los símbolos de una modulación

Los M símbolos de una modulación pueden ordenarse sobre la constelación digital de acuerdo a dos criterios: ordenamiento natural o ordenamiento basado en el mapeado de Gray.

En el primer caso, los símbolos se ordenan consecutivamente de acuerdo a su índice natural. Sin embargo, esto puede dar lugar a que símbolos vecinos se diferencien en un número de bits cualquiera. En ese caso, si la demodulación es incorrecta y se detecta un símbolo vecino, el número de errores que se pueden cometer a nivel de bit puede ser grande.

Para solucionar este problema, se utiliza el mapeado de Gray que establece un algoritmo sencillo para ordenar los símbolos sobre la constelación, de forma que símbolos vecinos sólo se diferencian en un único bit. Esto garantiza que la detección incorrecta de un símbolo sólo afectará a un bit tras el proceso de demodulación.

La diferencia entre ambos métodos de ordenamiento para una modulación QPSK puede apreciarse en la figura 2.12.

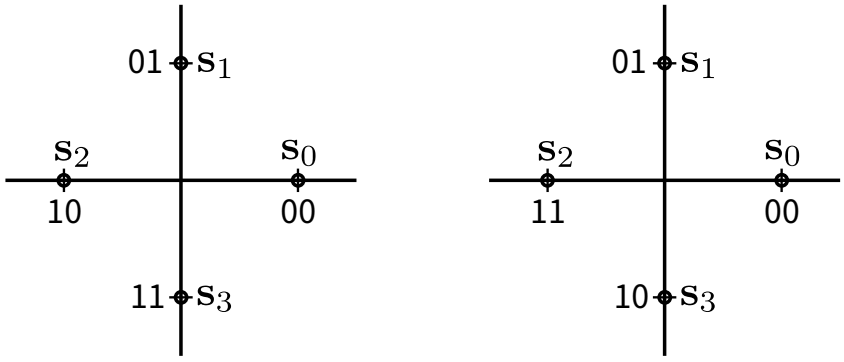

Figura 2.12: Comparación entre un ordenamiento natural y el mapeado de Gray para QPSK

2.3 Canal de comunicación

Los canales de comunicación son el medio físico por el cuál se transmiten las señales portadoras de la información desde el emisor hasta el receptor. Debido a las características e imperfecciones del canal, el mensaje original puede verse afectado por distintos efectos no deseados como ruido, interferencias, atenuaciones, etc. Las interferencias suelen aparecer cuando se trabaja con señales con bandas de frecuencia muy próximas, mientras que, el ruido son señales aleatorias que se añaden a la señal originalmente transmitida.

Los canales de comunicación presentan algunos parámetros esenciales: la capacidad de canal, la velocidad de transmisión y el ancho de banda. La capacidad de canal es la máxima cantidad de datos que pueden ser transmitidos a través de ese canal, depende tanto de su ancho de banda como de la relación señal a ruido, y se mide en bits por segundo. La velocidad de transmisión mide la cantidad de información que se transmite en una unidad de tiempo. Esta variable depende de la frecuencia de la señal y del ancho de banda del canal utilizado. El ancho de banda determina el rango de frecuencias disponibles para realizar la transmisión.

Los canales de comunicación pueden dividirse en dos grupos, aunque en ambos la transmisión se lleva a cabo mediante señales de naturaleza analógica como ondas electromagnéticas o pulsos de luz. Estos tipos de canal son:

- Canales confinados. Son aquellos en los cuáles la transmisión de la información se lleva a cabo a través de un medio físico. Al tratarse de medios cerrados, son más resistentes contra las interferencias o el ruido. En muchos casos requieren de un elevado coste de instalación, y la distancia que pueden alcanzar es limitada. El cable coaxial, el cable de par trenzado o la fibra óptica son algunos ejemplo de este tipo de canal.

- Canales no confinados. Son aquellos dónde la transmisión es inalámbrica, es decir, se realiza a través del aire. No requieren de instalación ni mantenimiento físico y pueden

ser usadas en transmisiones de largas distancias. Como contrapartida, son más susceptibles ante ruidos e interferencias. A pesar de ello, la libertad que nos confiere este tipo de canal ha hecho que su uso haya incrementado a lo largo de los años, siendo hoy en día impensable la vida cotidiana sin este medio de transmisión [11].

En este proyecto se han considerado dos tipos de canales habitualmente empleados en la literatura: canal AWGN y canal Rayleigh. El primero se utiliza para modelar el efecto del ruido térmico que puede considerarse Gaussiano, mientras que el segundo permite modelar las características de un canal inalámbrico con desvanecimiento a pequeña escala. Además, se explicará el canal de comunicaciones que resulta de utilizar múltiples antenas para la transmisión y recepción de la información. Estos sistemas suelen conocerse con el nombre de sistemas MIMO y el canal resultante en el contexto de transmisiones inalámbricas presenta una serie de propiedades que se pueden modelar matemáticamente de forma lineal.

2.3.1 Canal AWGN

El ruido aditivo blanco Gaussiano o AWGN es un tipo de ruido básico que trata de representar los efectos aleatorios que se pueden producir en una transmisión. Como su propio nombre indica, las principales características de este tipo de ruido son [12]:

- Es Gaussiano puesto que las componentes del ruido tiene una distribución normal en el dominio del tiempo con valor medio igual a 0.
- Es blanco, porque presenta una potencia uniforme en toda la banda de frecuencia del sistema de comunicaciones o, equivalentemente, no existe correlación entre las diferentes componentes del ruido,
- Es aditivo porque se suma al efecto del canal sobre la información introducida.

Por tanto, cuando hablamos de canal AWGN nos referimos al conjunto de canales que añaden ruido aditivo blanco Gaussiano a las señales transmitidas a través de ellos. En prácticamente todos los casos de transmisión de datos se da por hecho que la información va a ser alterada por algún tipo de ruido o interferencia, siendo este el ruido el más común. Por esto, el canal AWGN es utilizado para modelar muchos casos de comunicaciones reales.

En el caso de un canal AWGN, la señal recibida se puede representar directamente como:

$$r_i = s_i + n_i, \tag{2.25}$$

donde r_i y s_i representan el símbolo recibido y el transmitido, respectivamente, y n_i es la componente de ruido AWGN tal que $n_i \sim \mathcal{N}(0, \sigma_n^2)$.

En el año 1948, Shannon determinó la máxima cantidad de datos digitales que pueden transmitirse por un canal sin errores, con un ancho de banda específico y con presencia de ruido. Junto a Hartley estableció el teorema Shannon-Hartley que determina la velocidad máxima a la que se puede transmitir la información por un canal con ancho de banda finito y considerando ruido Gaussiano para que la probabilidad de error sea arbitrariamente pequeña. Observaron que, con la misma potencia de ruido, a medida que se incrementa la velocidad de transmisión los errores aumentan. Por ello, este teorema se formula como:

$$C = B \log_2 \left(1 + \frac{P_t}{\sigma_n^2} \right), \tag{2.26}$$

donde C representa la velocidad máxima de transmisión medida o capacidad del canal, en bits/s; B es el ancho de banda del canal, en Hercios; y $\eta = P_t/\sigma_n^2$ es la relación señal a ruido, también conocida como SNR [13]. Este parámetro permite definir directamente las condiciones/estado del canal y, habitualmente, se utiliza en simulación para fijar el nivel de distorsión que se introduce sobre la señal transmitida.

2.3.2 Canal Rayleigh

Cuando se transmite una señal sobre un medio inalámbrico se pueden producir dos tipos de efectos sobre la señal: el desvanecimiento a pequeña escala y los efectos derivados de la propagación de la señal por el medio más relacionados con el desvanecimiento a mayor escala.

Por un lado, *large-scale fading*, o desvanecimiento a gran escala, se produce cuando en la transmisión de una señal a grandes distancias, esta señal encuentra obstáculos entre el emisor y receptor, lo que genera una pérdida de la potencia o atenuación de la señal. Por otra parte, *small-scale fading*, o desvanecimiento a pequeña escala, se refiere a los rápidos cambios de la fase y amplitud de una señal de radio durante períodos cortos de tiempo o distancias cortas. Este tipo de desvanecimiento depende básicamente de la relación entre los parámetros de la señal, como su ancho de banda o período de símbolo, y entre los parámetros del canal [14].

Los efectos derivados del desvanecimiento a gran escala se suelen modelar mediante coeficientes de atenuación que multiplican a la señal transmitida y que dependen de las características del medio de propagación. Los efectos causados por el desvanecimiento a pequeña escala en medios inalámbricos se suelen modelar como el producto de los símbolos transmitidos por el canal por una serie de coeficientes cuyo módulo sigue una distribución de Rayleigh. Es decir, la señal recibida será:

$$r_i = h_i s_i + n_i, \tag{2.27}$$

donde r_i y s_i representan el símbolo recibido y el transmitido, respectivamente, h_i es el coeficiente asociado al desvanecimiento a pequeña escala y n_i es la componente de ruido AWGN.

En general, los coeficientes de desvanecimiento h_i serán valores complejos donde la parte real y la parte imaginaria no están correlacionadas y siguen una distribución Gaussiana de media 0 y varianza unidad. Esto garantiza que su módulo sigue una distribución de Rayleigh. En el contexto del modelado matemático de canales inalámbricos, estos valores se generan de forma aleatoria, lo que permite, además, emular las condiciones variables de este tipo de canales. Dependiendo de la velocidad con la que cambie el canal podemos hablar de varios tipos de desvanecimientos:

- Desvanecimiento rápido: el tiempo durante el cual el canal es constante es muy pequeño y, por tanto, el canal varía muy rápido.
- Desvanecimiento en bloque: el canal permanece constante durante la transmisión de un bloque de símbolos.
- Desvanecimiento lento: las condiciones del canal se mantienen constantes durante un período largo de tiempo, de forma que podemos hablar de canales cuasi-estáticos.

Para desarrollar el simulador, vamos a centrarnos en el desvanecimiento rápido debido a que es el más habitual y no supone ninguna pérdida de generalidad el hecho de considerar solo un tipo de desvanecimiento. Para este tipo de canales, podemos hablar de capacidad instantánea de canal que depende directamente del coeficiente h_i en cada instante de tiempo. En ese caso, la capacidad viene dada por:

$$C = B \log_2 \left(1 + \frac{|h_i|^2 P_t}{\sigma_n^2} \right), \tag{2.28}$$

donde $|h_i|^2$ representa el cuadrado del módulo del coeficiente h_i. Como se puede observar, esta capacidad de canal puede ser interpretada como una extensión directa de la capacidad de un canal AWGN. En este caso, las condiciones del canal vienen determinadas por el coeficiente h_i y, de nuevo, por el valor de SNR, $\eta = P_t/\sigma_n^2$, que se suele configurar en simulación.

En el caso de canales Rayleigh es necesario realizar una operación de ecualización del canal antes de proceder a la demodulación de la información recibida. Esta operación de ecualización consiste básicamente en invertir la distorsión introducida por el canal sobre la señal transmitida. Existen diferentes tipos de filtros que permiten llevar a cabo la operación de ecualización aunque, en este caso, nos vamos a centrar en el filtro *Minimum Mean-Square Error* (MMSE) por su compromiso entre sencillez y rendimiento. Este filtro se define como:

$$w = \frac{h_i^*}{|h_i|^2 + \sigma_n^2}, \tag{2.29}$$

donde el superíndice * representa el conjugado del coeficiente que acompaña y σ_n^2 es la varianza del ruido AWGN.

2.3.3 Canal MIMO

Los sistemas de comunicaciones MIMO (del inglés *multiple-input multiple-output*) suponen una gran mejora respecto de los sistemas tradicionales con una sola antena, puesto que permiten el uso de múltiples antenas transmisoras y receptoras. De esta forma, al transmitirse la señal de información se obtienen múltiples canales sobre una misma banda de frecuencia, lo que es conocido como multiplexación espacial. En la figura 2.13 se aprecia el esquema de un sistema basado en MIMO.

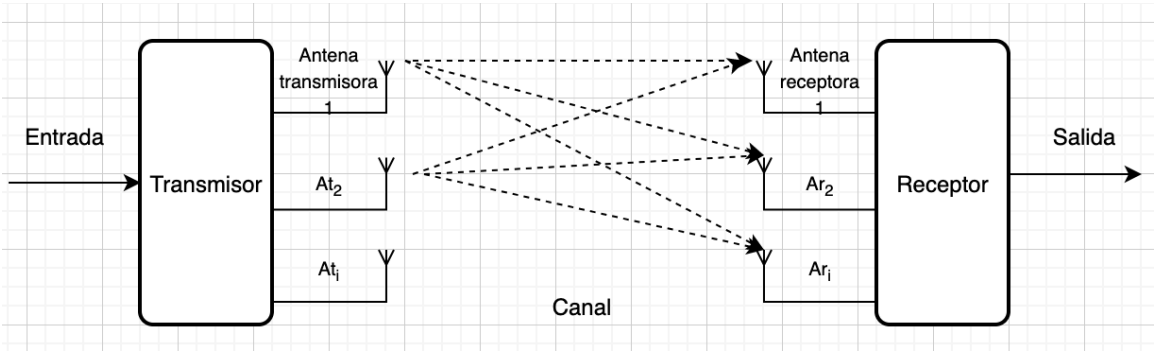

Figura 2.13: Esquema gráfico de un sistema MIMO.

Debido a su mayor número de antenas, este tipo de sistemas son capaces de incrementar el número de bits transmitidos por unidad de tiempo, es decir, permiten aumentar la velocidad de transmisión. Alternativamente, pueden ser utilizados para aumentar la fiabilidad de la comunicación aprovechando las diferentes señales recibidas en el receptor para reducir la tasa de error. Esta propiedad se conoce como diversidad espacial [15]. Por todas estas ventajas, los sistemas MIMO son muy utilizados actualmente en todo tipo de comunicaciones inalámbricas, desde comunicaciones móviles hasta redes WiFi, pasando por estándares de alcance medio como WiMax.

En general, el modelado de los canales MIMO inalámbricos resultantes del uso de múltiples antenas en el transmisor y receptor, no es sencilla. Sin embargo, siguiendo una filosofía similar a la explicada para el caso de una antena y canales Rayleigh, el modelo matemático se simplifica si nos centramos en los efectos debidos al desvanecimiento a pequeña escala. En ese caso, asumiendo un sistema MIMO con n_T antenas en el transmisor y n_R antenas en el receptor, la señal recibida en la i-esima antena se puede representar como:

$$y_i = \sum_{j=1}^{n_T} h_{i,j} s_j + n_i, \qquad \forall j = 1, \ldots, n_R, \tag{2.30}$$

donde $h_{i,j}$ representa el coeficiente asociado al canal entre la antena transmisora j y la antena receptora i, s_j es el símbolo modulado transmitido por la j-ésima antena y n_i es la componente

de ruido AWGN correspondiente. Esta relación lineal puede ser rescrita en notación matricial como:

$$y = Hs + n, \tag{2.31}$$

con $y = [y_1, \ldots, y_{n_R}]^T$, $s = [s_1, \ldots, s_{n_T}]^T, n = [n_1, \ldots, n_{n_R}]^T$ y la matriz de canal

$$H = \begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1n_T} \\ h_{21} & h_{22} & \cdots & h_{2n_T} \\ \vdots & \vdots & \ddots & \vdots \\ h_{n_R1} & h_{n_R2} & \cdots & h_{n_Rn_T} \end{bmatrix}.$$

De esta forma, el canal MIMO queda totalmente caracterizado por la matriz de respuesta del canal, H. Las entradas de esta matriz suelen ser valores complejos que siguen una distribución Rayleigh estándar, es decir, cuya parte real e imaginaria son variables aleatorias Gaussianas independientes de media 0 y varianza 1. De nuevo podemos hablar de diferentes tipos de desvanecimiento dependiendo de la velocidad con la que varíen los coeficientes del canal o, equivalentemente, del tiempo de coherencia del canal.

La capacidad instántanea de un canal MIMO viene dado por la siguiente ecuación [16, 17]:

$$C(H) = \max_{p(s)} I(s;y) = \max_{p(s)} \log\left[\det\left(I_{n_R} + \frac{P_t}{n_T\sigma_n^2} HH^H\right)\right], \tag{2.32}$$

es decir, viene determinada por la distribución de entrada al canal que maximiza la información mutua entre la entrada y la salida del canal, $I(s;y)$. El superíndice ^{H} representa la operación del conjugado transpuesto sobre la matriz que se aplica.

Para el caso de canales MIMO inalámbricos, la matriz H es aleatoria y, por lo tanto, la capacidad del canal correspondiente es también una variable aleatoria. Sin embargo, si suponemos que las realizaciones del canal se describen mediante un proceso estocástico estacionario WSS (del inglés Wide Sense Stationary), podemos calcular la capacidad ergódica del canal MIMO como la esperanza de las capacidad instantáneas sobre todas las realizaciones de la variable aleatoria H, es decir:

$$C_E = \mathbb{E}_H\left[\log\ \det\left(I_{n_R} + \frac{P_t}{n_T\sigma_n^2} HH^H\right)\right]. \tag{2.33}$$

Por último, al igual que en el caso de canales Rayleigh, es necesario realizar una operación de ecualización para deshacer los efectos del canal. De nuevo, vamos a centrarnos en el filtro MMSE que viene dado por:

$$W = (HH^H + \frac{P_t}{\sigma_n^2} I_{n_R})^{-1} H. \tag{2.34}$$

2.4 Demodulación

La operación de demodulación puede verse como el proceso inverso de la modulación. Es decir, es el proceso de recuperar la secuencia de bits transmitida a través del canal a partir de la secuencia de símbolos recibidos o ecualizados. Si analizamos el caso de canales AWGN, el símbolo recibido viene dado por:

$$r_i = s_i + n_i. \tag{2.35}$$

Por tanto, el objetivo del proceso de demodulación es determinar el símbolo que más probablemente se ha transmitido a través del canal a partir del símbolo recibido r_i (*hard decision*), o bien proporcionar información relativa a la probabilidad de haber transmitido cualquiera de los símbolos de la modulación y recibir r_i (*soft decision*).

Como la información ha sido transmitida a través de un canal ruidoso, han podido producirse errores y, por tanto, cambios en la información original que se quería transmitir. Llegados a este punto, se inicia una fase de corrección de los errores que se han podido producir. En esta fase de corrección de errores, el demodulador puede o no intervenir, y por eso, podemos diferenciar entre dos tipos de decisores:

- *Hard decision.* El demodulador no toma parte de la corrección de errores, puesto que le entrega la información al decodificador tal y como la detecta tras su paso por el canal, es decir, como un conjunto fijo de valores que en los códigos binarios son siempre de 0 o 1. Es decir, el demodulador proporciona al decodificador una decisión binaria basada en determinar cuál ha sido el símbolo que se ha transmitido con mayor probabilidad.

  En el caso de comunicaciones digitales, el demodulador óptimo es el detector máximo a posteriori (MAP) que, para el caso de símbolos equiprobables, coincide con el detector por máxima verosimilitud o ML (del inglés, *maximum likelihood*). En ese caso, el símbolo que más probablemente se ha transmitido se calcula como:

  $$\hat{s}_{\text{MAP}} = \hat{s}_{\text{ML}} = \arg\min_{s_i} \sum_{j=1}^{N} (r_{i,j} - s_{i,j})^2 = \arg\min_{s_i} |r_i - s_i|^2. \tag{2.36}$$

  Es decir, el símbolo demodulado será aquel que está más cerca en términos de distancia euclídea del símbolo recibido r.

- *Soft decision.* Aquí, por el contrario, el demodulador proporciona información para predecir el símbolo más probable a través de una estimación, que puede tomar valores intermedios dentro de un rango, y que le pasa al codificador. Estos posibles valores se denominan LLRs (*log-likelihood ratios*).

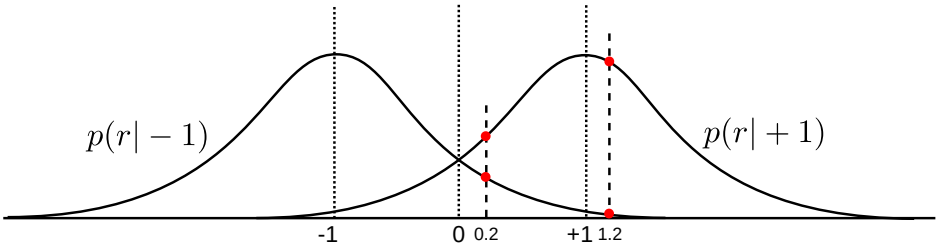

Figura 2.14: Ejemplo de información *soft*.

En la figura 2.14, se puede apreciar un ejemplo donde se compara el tipo de información que se proporciona en el demodulación *hard* y en la demodulación *soft* para una modulación *Binary Phase-Shift Keying* (BPSK). Como se puede apreciar, se reciben dos símbolos diferentes, +0.2 y +1.2. En el caso de la demodulación *hard*, ambos símbolos son demodulados como +1, ya que están más cerca de este símbolo que del otro posible símbolo, -1. Sin embargo, el nivel de confianza en la decisión es muy diferente para los dos símbolos recibidos. En el caso de recibir +1.2, la probabilidad de haber transmitido un +1 es mucho mayor que la probabilidad de transmitir un -1. Sin embargo, en el caso de recibir +0.2, la probabilidad de haber transmitido un +1 es ligeramente mayor que la de haber transmitido un -1. En una demodulación *soft*, el decisor computaría el coeficiente entre el logaritmo de ambas probabilidades –transmitir un +1 y transmitir un -1– y enviaría esa información al decodificador de canal. El resultado de ese cálculo es una estimación mucho más precisa del nivel de confianza en la decisión del demodulador que puede ser explotada por el decodificador para mejorar sus prestaciones. En general, las LLRs se computan como:

$$L(c_i) = \log_2 \left(\frac{p(c_i = 0|r_i)}{p(c_i = 1|r_i)} \right),$$

donde c_i representa el i-ésimo bit correspondiente a la palabra demodulada.

2.5 Decodificación

La decodificación es el proceso donde, a partir de la información proporcionada por el demodulador, se obtiene el mensaje con la información fuente que el emisor le quiere transmitir al receptor.

2.5.1 Códigos bloque lineal

Recordemos que todo código bloque lineal $C(n, k)$ tiene una matriz generadora G, de dimensión $k \times n$, además de la matriz control de paridad, de dimensión $(n - k) \times k$.

Después de la operación de demodulación, en el decodificador tendremos las palabras recibidas, r, que siguen la siguiente fórmula:

$$r = c + e, \tag{2.37}$$

siendo c las palabras código transmitidas antes de ser moduladas, atravesar el canal de comunicaciones y ser demoduladas, y e, el patrón error que pudo ser introducido durante este proceso. Teniendo en cuenta esto, se puede utilizar la denominada decodificación basada en síndromes. En este tipo de decodificación, primero se calcula el síndrome para la palabra recibida de la siguiente forma:

$$s = rH^t. \tag{2.38}$$

Tras llevarse a cabo este cálculo pueden darse dos escenarios diferentes:

- $s \neq 0$. Esto implica que la palabra recibida no es una palabra código y, por lo tanto, se detecta que se ha producido algún error durante la transmisión. Para un determinado código, se puede construir su tabla de síndromes donde se especifica para cada posible patrón de error e su correspondiente síndrome. De esta forma, a partir de la palabra recibida, se puede calcular su síndrome con la ecuación en (2.38), determinar el patrón de error que se ha producido y corregir los errores introducidos durante la transmisión.
- $s = 0$. La palabra recibida es una palabra código. Esto puede ser debido a que no se ha producido ningún error durante la transmisión o que el patrón de error ha cambiado la palabra código transmitida en otra palabra código diferente, haciendo imposible la detección de los errores [18].

Para facilitar el entendimiento de esta decodificación, utilizaremos el ejemplo de un código bloque lineal con $k = 3$, $n = 6$ y $l = n - k = 3$. Las matrices generadora, G, y control de paridad, H, de este código serán las siguientes:

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}, \tag{2.39}$$

$$H = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}, \qquad H^t \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.40}$$

Imaginemos que la palabra código transmitida es c = [1 0 1 0 0 1] y que el patrón de error es e = [0 0 0 0 0 0], por lo que la palabra recibida, r, sería la misma que la inicialmente transmitida. Siguiendo la fórmula para hallar el síndrome, $s = rH^t = [0\ 0\ 0]$, por lo que podríamos suponer que no ha ocurrido ningún error.

En el siguiente ejemplo, vamos a considerar los casos en los que se produce un solo error en la transmisión. En este caso, con un solo error en e, el síndrome en s se corresponderá directamente con la columna de H en la misma posición en la que se ha producido el error. Por tanto, en nuestro ejemplo podríamos construir la siguiente tabla de síndromes:

$$e_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \longrightarrow e_1 H^t = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \tag{2.41}$$

$$e_2 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \longrightarrow e_2 H^t = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \tag{2.42}$$

$$e_3 = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \longrightarrow e_3 H^t = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \tag{2.43}$$

$$e_4 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \longrightarrow e_4 H^t = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \tag{2.44}$$

$$e_5 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \longrightarrow e_5 H^t = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \tag{2.45}$$

$$e_6 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \longrightarrow e_6 H^t = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \tag{2.46}$$

Entonces ahora modificando $e = [0\ 0\ 0\ 1\ 0\ 0]$, la palabra recibida sería r = [1 0 1 1 0 1]. En este nuevo supuesto, $s = [1\ 0\ 0]$, de forma que detectamos que se ha producido un error y suponemos que este error se ha dado en la cuarta posición ya que el síndrome coincide con la cuarta columna de H. Por tanto, el error que se ha producido puede ser corregido y la palabra fuente decodificada se corresponderá con la palabra fuente transmitida.

Ahora si consideramos que se ha recibido la palabra $r = [1\ 0\ 0\ 0\ 0\ 0]$, y hallamos el síndrome, comprobamos que $s = [1\ 0\ 1]$. Como eH^t no vale [1 0 1] para ningún patrón de error con un solo bit, podemos detectar que en este caso se ha producido más de un error, pero no podremos corregirlo. Esto se debe a que el código lineal que se ha usado como ejemplo tiene una distancia mínima de 3 y sólo permite la corrección de 1 error.

En el caso particular de los códigos de Hamming, debido a que cada columna de la matriz de control de paridad se corresponde en binario con la posición que ocupa dicha columna, cuando se calcula el síndrome directamente se obtiene la posición en la que se ha producido el error. Por tanto, la corrección del error es mucho más sencilla y rápida, ya que simplemente se

tiene que pasar el síndrome a decimal, evitando una búsqueda exhaustiva sobre las columnas de la matriz H.

2.5.2 Códigos convolucionales

Para decodificar un flujo de información codificada con un código convolucional se utiliza el algoritmo de Viterbi. Esta decodificación publicada por Andrew J. Viterbi en 1967 trata de buscar, apoyándose en el diagrama de trellis, el camino con la menor distancia de Hamming respecto a la palabra recibida.

Para determinar el camino de menor coste –menor distancia de Hamming respecto de la palabra recibida– desde el estado inicial a los nodos de la última de etapa del trellis sería necesario, en principio, explorar todos los posibles caminos y quedarnos con el de menor coste. Sin embargo, el coste computacional de esta exploración crecería de forma exponencial con el número de transiciones y el número de etapas en el trellis. Es decir, su complejidad sería prohibitiva para códigos relativamente complejos y secuencias de entrada no demasiado grandes. Es importante tener en cuenta que, en la fase de decodificación, es necesario utilizar tantas etapas en el trellis como bloques de k bits tenemos a entrada del codificador.

El algoritmo de Viterbi permite reducir significativamente la complejidad de esta operación de decodificación para códigos convolucionales. La clave del algoritmo de Viterbi está en que, en el caso de que se pueda llegar a un nodo intermedio por varios caminos diferentes, sólo es necesario considerar el camino de menor coste y explorar los caminos que lo incluyan. Esta sencilla variación sobre el algoritmo de búsqueda completa permite que la complejidad de la operación de decodificación pase a ser lineal con el número de estados y etapas en la decodificación.

Para ejemplificar el funcionamiento del algoritmo de Viterbi, vamos a considerar el código convolucional correspondiente al diagrama de bloques de la figura 2.5 con diagrama de trellis representado en la figura 2.15. Pongamos como ejemplo que la secuencia recibida es $r = [11\ 01\ 10]$. Utilizando el diagrama de trellis del código y el algoritmo de Viterbi, se va determinando, etapa a etapa, el camino con menor distancia de Hamming a la palabra recibida. Partimos siempre del estado inicial del decodificador en 00. La primera palabra código recibida es 11, para el caso de que la decodificación fuera un 0, la distancia sería de 2, puesto que $d_H(11, 00) = 2$; mientras que, en el caso de que la entrada fuera un 1, la distancia sería 0 ya que $d_H(00, 00) = 0$. Este procedimiento puede observarse en la figura 2.16. El siguiente bloque de la secuencia recibida es 01, por lo que hallando las diferentes rutas con sus distancias, el diagrama quedaría como se muestra en la figura 2.17.

En la siguiente etapa del trellis ya es posible llegar a cada nodo por dos caminos diferentes. Aplicando el algoritmo de Viterbi, se elegiría siempre el camino con menor distancia acumulada y se descartaría el otro. El último bloque de la palabra recibida es 10, por lo que el

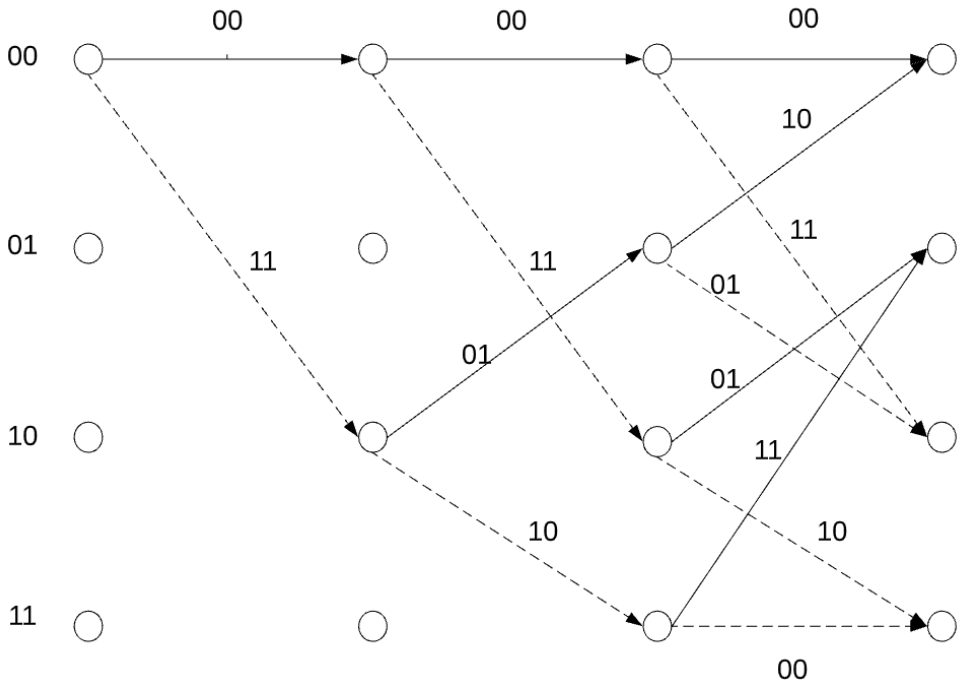

Figura 2.15: Diagrama de trellis del código convolucional de ejemplo

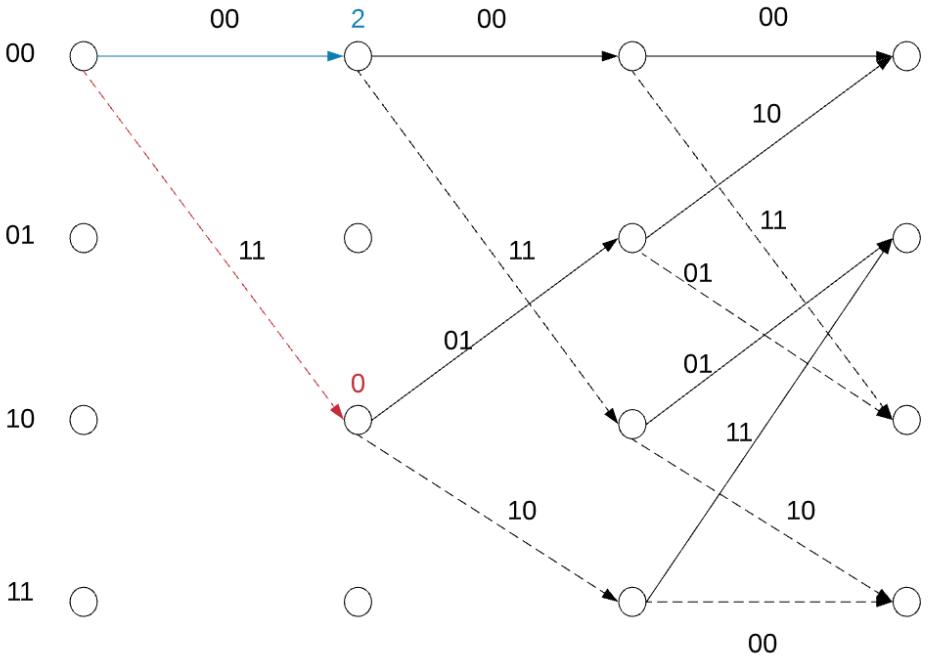

Figura 2.16: Diagrama de trellis después del primer paso

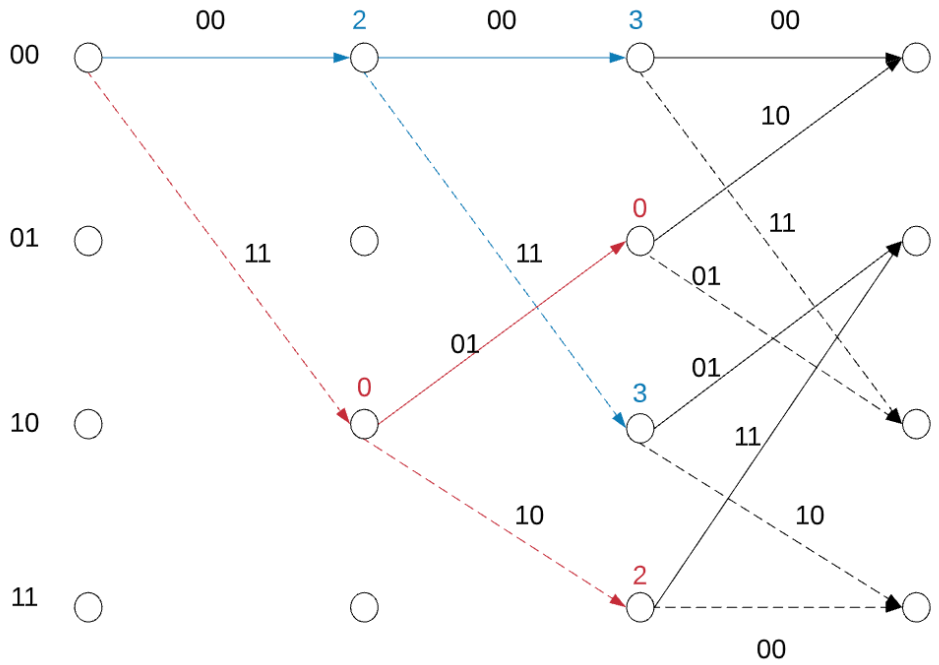

Figura 2.17: Diagrama de trellis después del segundo paso

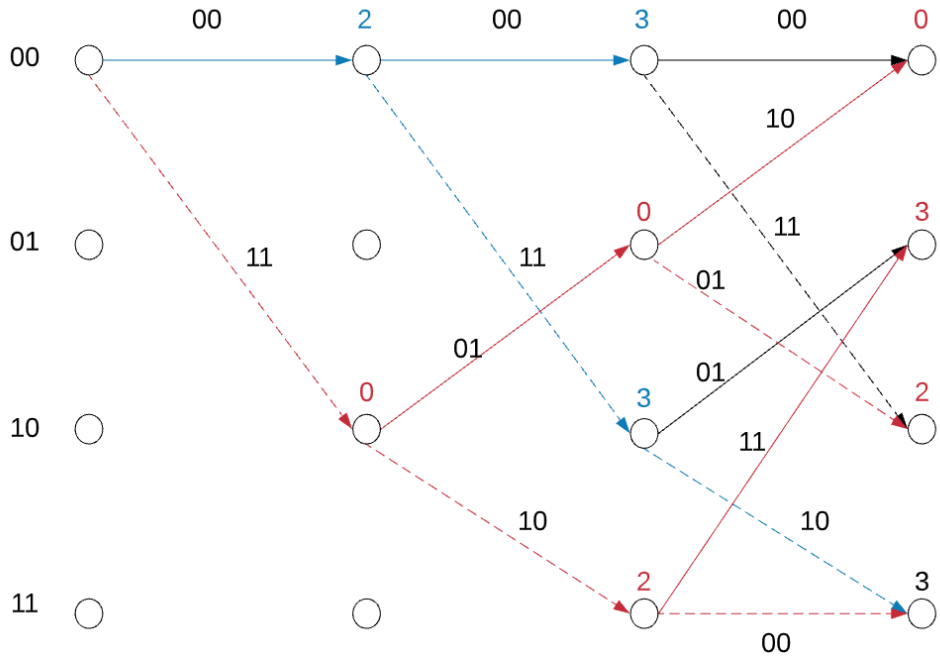

Figura 2.18: Diagrama de trellis tras el último paso

resultado final de la decodificación sobre el diagrama de trellis sería el que se muestra en la figura 2.18. En este punto, debemos elegir el camino por el que se han cometido menos errores, es decir, el camino con menor distancia de Hamming. En este caso existe un camino con $d_H = 0$, por lo que será el elegido. Siguiendo las transiciones correspondientes a este camino, se obtiene la secuencia decodificada que, en este caso, sería [1 0 0].

2.5.3 Turbo códigos

Para realizar la decodificación en los turbo códigos no existe un único algoritmo. Se pueden utilizar diversos algoritmos aunque todos ellos se basan en intercambiar, de forma iterativa, información *soft* de los bits recibidos entre los dos decodificadores convolucionales concatenados que forman el turbo código. La información *soft* de entrada son generalmente la LLRs proporcionadas por el demodulador. Esta información se va refinando en las diferentes iteraciones del algoritmo de decodificación y la decisión final de los bits transmitidos se realiza a partir de las LLRs obtenidas en la última iteración.

Uno de los algoritmos más utilizados para la decodificación de turbo códigos es el algoritmo basado en el criterio MAP. En este algoritmo, cada decodificador obtiene como salida una distribución de probabilidades a posteriori (LLRs) para cada símbolo recibido, a partir de una distribución de probabilidades a la entrada. La secuencia final de distribuciones se obtiene mediante un método iterativo donde la secuencia de LLRs a la salida de un decodificador sirve como entrada para el otro decodificador hasta que la diferencia entre sus distribuciones sea suficientemente pequeña. Es importante tener en cuenta que es necesario entrelazar de forma correcta la secuencia de LLRs a la salida del primer decodificador antes de pasarlas como entrada al segundo codificador, debido a la presencia del entrelazador que se utiliza en la fase de codificación (recordar sección 4.14).

2.5.4 LDPC

Los algoritmos de decodificación para los códigos LDPC se basan en el intercambio de información entre los dos tipos de nodos que tenemos para el código en su representación gráfica en forma de grafo bipartito. En el lado izquierdo, tenemos los nodos variable que representan los símbolos de la palabra código y, por lo tanto, hay tantos como bits tiene cada palabra código. En el lado derecho, estarían los nodos paridad que representan restricciones de paridad entre los nodos variables y cuyo número es igual al número de filas de la matriz control de paridad. Entre estos nodos se establecen conexiones por medio de aristas, donde cada una representa la participación de un nodo variables en su correspondiente función de paridad.

Esta manera de representar los códigos LDPC mediante grafos bipartitos permite factorizar una función global de varias variables como el producto de distintas funciones locales

de una sola variable. En este caso, se puede utilizar para realizar la factorización de la función conjunta de probabilidad asociada a todos los símbolos que participan en una restricción de paridad y computar de forma eficiente las distribuciones marginales de probabilidad de los bits recibidos con el algoritmo de suma-producto. En el grafo bipartito correspondiente a códigos LDPC, los nodos variable representan las variables, mientras que los nodos paridad representan las funciones locales. Por lo que es fácil intuir que, en este tipo de grafos, los bits de las palabras código se corresponden con las variables, y las restricciones de paridad con las funciones locales.

Algoritmo suma-producto

El algoritmo suma-producto es un algoritmo iterativo de paso de mensajes entre los nodos variables y los nodos de paridad. Si llamamos V al conjunto de todos los nodos variables, y P al conjunto de nodos de paridad, para todo $i \in V$ y $a \in P$, un mensaje enviado desde i a un nodo a es el producto de todos los mensajes recibidos por i de todos sus nodos vecinos a excepción de a. Esto se puede denotar como:

$$n_{i\to a}(i) = \prod_{h\in E(i)/a} m_{h\to i}(i), \tag{2.47}$$

donde $E(x_i)$ es el conjunto de nodos de paridad que están conectados a i, y $m_{h\to i}(i)$ es el mensaje enviado desde el nodo de paridad h al nodo variable i. También podemos calcular el mensaje enviado desde un nodo paridad p a un nodo variable v con la siguiente fórmula:

$$m_{p\to v}(v) = \sum_{X/v} f_p(X) \prod_{j\in N(p)/v} n_{j\to p}(j), \tag{2.48}$$

donde $N(p)$ es el conjunto de nodos variables que son vecinos del nodo paridad p, $n_{j\to v}(j)$ es el mensaje enviado desde el nodo variable j al nodo de paridad p y $f_p(X)$ representa la función local de restricción del nodo de paridad p aplicada sobre todos sus nodos variables vecinos excepto v. Es decir, el nodo de paridad actualiza su información integrando los mensajes procedentes de todos los nodos variables excepto v y cálcula el mensaje utilizando su función local de acuerdo a la restricción de paridad.

El algoritmo de suma-producto permite actualizar los mensajes en los nodos variables y de paridad de forma iterativa. En la primera iteración los nodos variables actualizan sus mensajes con las LLRs proporcionadas por el demodulador y pasan sus mensajes a los nodos de paridad a los que están conectados. El proceso iterativo va actualizando los mensajes de los dos tipos de nodos de acuerdo a las ecuaciones 2.47 y 2.48. Cuando se completan todas las iteraciones del algoritmo de suma-producto, los bits decodificados son determinados a partir de los mensajes finalmente obtenidos en los nodos variable siguiendo una filosofía similar a la explicada en la decodificación de los turbo códigos.

2.6 Métrica de rendimiento de un sistema de comunicaciones digital

Cuando se transmite información a través de un sistema de comunicaciones digital es probable que, durante esta transmisión, se produzcan errores en la información que se quiere enviar debido a interferencias o distorsiones introducidas por el canal de comunicación. La integridad del sistema puede verse comprometida por lo que es necesario evaluar el rendimiento del sistema, para lo cuál es habitual utilizar las llamadas curvas de *Bit Error Rate* (BER).

ElBER viene determinada por la probabilidad de error en la transmisión y se define, por tanto, como el número total de errores en los bits recibidos respecto a los bits enviados, i.e.:

$$\text{BER} = \frac{\text{nº de bits erróneos}}{\text{nº de total de bits fuente}}. \tag{2.49}$$

Este valor depende habitualmente de las condiciones del canal sobre el que se realiza la transmisión, es decir, del valor de SNR en recepción. Por esa razón, el rendimiento de un sistema de comunicaciones se suele medir para diferentes condiciones del canal o, lo que es lo mismo, para un rango de SNRs. De esta forma, una manera muy gráfica para evaluar este rendimiento es visualizar su curva de BER frente a los valores de SNR, donde se dibuja la BER obtenida para cada valor concreto de SNRs en el rango considerado. En la figura 2.19, se muestra un ejemplo de este tipo de gráficas.

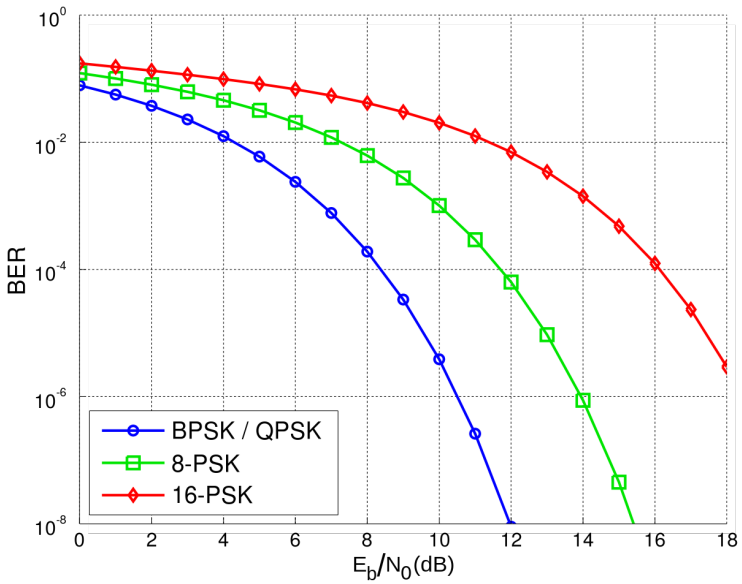

Figura 2.19: Ejemplo de curvas para distintas modulaciones

Este tipo de curvas nos permiten comparar rápidamente el rendimiento entre diferentes tipos de sistemas de comunicaciones, o entre diferentes configuraciones de un mismo modelo.

Capítulo 3

Fundamentos tecnológicos

En este apartado explicaremos todas las tecnologías utilizadas para desarrollar nuestro simulador de sistemas de comunicaciones. Entre las diferentes herramientas empleadas, podemos destacar las siguientes: Python 3.7, lenguaje de programación con el cuál fue desarrollada la aplicación; Qt Designer, herramienta con la que se ha realizado la interfaz gráfica de usuario; además de numerosas librerías de Python utilizadas para el desarrollo del *software*.

3.1 Python

Python fue creado por Guido van Rossum en la década de los 80 en el Centro para las Matemáticas y la Informática, para tomar el relevo del lenguaje ABC. Su nombre viene dado por la afición de su creador por los humoristas Monty Python.

Entre las principales características de este lenguaje podemos destacar que:

- Python es un lenguaje de programación interpretado y multiparadigma ya que permite adoptar los principales estilos de programación como son programación orientada a objetos, programación imperativa y programación funcional. Un objetivo de Python es la facilidad de extensión por lo que este lenguaje también puede soportar otros paradigmas.
- Es multiplataforma en el sentido de que el código, cuyo formato es estructural, puede ser ejecutado sobre varios sistemas operativos diferentes.
- Para la administración de la memoria, este lenguaje utiliza tipado dinámico, mediante el cuál una variable puede tomar valores de distinto tipo, y conteo de referencias, que cuenta el número de veces que un recurso es referido.
- Python se puede usar libremente, ya que se desarrolla bajo licencia de código abierto aprobada por OSI.

Entre las principales ventajas de Python se encuentra su facilidad de portabilidad comparado con otros sistemas, es sencillo de aprender y simplifica mucho la programación, además de ser un lenguaje muy legible donde es fácil empezar a trabajar sobre un código ya escrito.

Por todo esto, Python fue elegido como lenguaje para el desarrollo, por delante de otras alternativas como Matlab que, aunque es muy eficiente en cálculos matemáticos, es más limitada a la hora de programar. Además de que, en Python, la implementación de las funciones está disponible por lo que se puede revisar, comprender y modificar, mientras que, en Matlab,

Como posibles desventajas podemos destacar que las librerías propiamente incluidas en Python son poco utilizadas, optando más por librerías realizadas por terceros [19].

Python presenta, como otros lenguajes de programación, gran variedad de tipos y estructuras de datos. Podemos trabajar con los clásicos tipos simples de datos: string, int, float, boolean, ... Además se incluyen tipos de datos compuestos que resultan útiles en diferentes situaciones.

A continuación, se van a describir brevemente algunas de las características más destacadas de Python, que han motivado su elección y que se han utilizado de forma profusa durante la realización de este proyecto.

3.1.1 Tipos de datos compuestos

Listas

En primer lugar tenemos las listas, que son variables que almacenan elementos ordenados por posiciones, y donde cada elemento puede ser un tipo de dato diferente.

```
lista = ['cadena', 'c', 100]
['cadena', 'c', 100]
```

Se puede acceder a cada elemento de la lista a través de su índice, y, además, los elementos son mutables.

```
lista[0]
'cadena'
lista[0] = 35
[35, 'c', 100]
```

Python incluye una serie de métodos que pueden aplicarse a las listas, entre los más comunes están: len() que devuelve la longitud de la lista; append(), que añade un elemento a la lista; o pop(), que devuelve el último elemento y lo borra.

Diccionarios

Los diccionarios son un tipo de datos que siguen una relación clave-valor. En ellos no existe orden, y los valores son mutables. En este caso, se acceden y asignan valores en el

diccionario a través de la clave.

```
diccionario = {1:10, 2:[], 3:"cadena"}
{1: 10, 2: [], 3: 'cadena'}

diccionario[1]
10

diccionario[2] = True
True
```

Entre los métodos que más destacan para aplicar sobre diccionarios están: has_key(clave) que devuelve un booleano indicando si la clave pasada como argumento existe en el diccionario; keys() que devuelve una lista con las claves del diccionario; values() que devuelve una lista con los valores del diccionario; y len() que devuelve la longitud del diccionario.

3.1.2 Funciones

Python sigue una programación estructurada. Esto es gracias al uso de funciones, que son bloques de código que llevan a cabo acciones concretas. Para lograr este propósito, las funciones pueden recibir o no parámetros que serán utilizados en las operaciones de la función y, normalmente, las funciones retornan uno o más valores.

```
def nombreFuncion(self, parametro1, parametro2, ...):
    sentencias
    RETURN valor
```

Gracias a utilizar este tipo de programación estructurada existen una serie de ventajas. Por un lado, la modularización, es decir, fragmentar el código en distintos bloques para así facilitar su modificación y que el código sea más legible y fácil de entender. Por otro lado, la reutilización, evitando así repetir sentencias del código cuando es necesario utilizarlas en distintas situaciones. Por último, la detección y corrección de fallos es más sencilla.

3.1.3 Módulos y paquetes

Para continuar con una estructuración de un código más legible y fácil de modificar, tenemos los módulos que son archivos .py en los que se pueden definir clases, funciones y variables. Además, también se pueden usar los paquetes que son directorios donde se almacenan módulos y otros paquetes, y su único requisito es contener un archivo __init__.py. Los módulos pueden ser importados a través de la palabra reservada import.

```
paquete1/
    __init__.py
    modulo1.py
```

```
    paquete2/
        __init__.py
        modulo2.py
        modulo3.py
```

Con el siguiente esquema, si quisieramos importar todos los módulos del "paquete2" podríamos indicarlo de cualquiera de las siguientes formas:

```
import paquete1.paquete2
from paquete1 import paquete2
from paquete1.paquete2 import *
```

Si sólo quisiéramos importar algún módulo específico entonces se utilizaría la siguiente sintaxis:

```
from paquete1.paquete2 import modulo2
```

3.1.4 Programación orientada a objetos en Python

La programación orientada a objetos es un paradigma de la progrmación que se basa en la organización del código en distintas clases, de las que se crearán objetos que se relacionarán entre sí para lograr los objetivos pretendidos por la aplicación.

Las clases en Python se definen de la siguiente manera:

```
class Vehículo:
    pass
```

Los objetos representan abstracciones y están formados por una identidad, un tipo y un valor. La identidad de un objeto es inmutable y es representada por un número entero que se puede consultar a través de la función $id()$. Su tipo es también inmutable y define el tipo de operaciones que se pueden hacer sobre ese objeto, la función $type()$ devuelve el tipo del objeto. El valor representa el contenido del objeto, es decir, un valor real que toma ese objeto.

El estado de un objeto es la representación lógica que toma en un momento dado y que puede variar a lo largo del tiempo. Un objeto posee una serie de atributos que son las características individuales que definen su estado, y que los diferencian de otro de la misma clase, ya que, cada uno, puede tomar valores diferentes para estas variables. Por ejemplo:

```
class Vehículo:
    matrícula = "1312ABC"
    marca = "Mercedes"
    modelo = "Clase C"
    años = 5
    encendido = False
```

Existen algunos atributos especiales, en los cuáles, su nombre va entre dos pares de guiones bajos; por ejemplo, __name__, que se corresponde con el nombre del atributo. Además, Python no distingue entre atributos públicos y privados. Por eso, por convención, si se quiere señalar que un atributo tiene que ser interpretado como privado se utiliza doble guión bajo al principio del nombre.

Los métodos de una clase son una serie de sentencias que definen las operaciones que se van a llevar a cabo sobre los objetos de esa clase. Se dice que son funciones ligadas a objetos, porque se definen igual que las funciones (la única diferencia es que son declaradas dentro de una clase y que su primer parámetro es siempre referente a la instancia que la llama), y el estado de los objetos puede ser modificado tras su ejecución. Siguiendo el ejemplo anterior, un método para la clase *Vehículo* podría ser:

```
def arrancar(self):
    self.encendido = True
    return "El vehículo ha arrancado"
```

Igual que los atributos, también existen una serie de métodos especiales. El más destacado es __init__() que funciona como los constructores de otros lenguajes de programación. Sus argumentos son los parámetros a incorporar cuando se quiere instanciar un objeto.

```
def __init__(self, matricula, marca, modelo, años):
        self.matricula = matricula
        self.marca = marca
        self.modelo = modelo
        self.años = años
```

Esto se hace ya que normalmente no se busca que todas las instancias se creen con los mismos valores en sus atributos. Así gracias a los constructores, se podrán crear instancias diferentes de una misma clase.

Existen diferentes técnicas en las que se basa la programación orientada a objetos.

Herencia

La herencia es una técnica que consiste en la creación de una primera clase general, que llamaremos superclase, a partir de la cuál otras clases más específicas, llamadas subclases, podrán reutilizar su código, es decir, pueden heredar de la clase principal sus atributos y métodos.

A diferencia de otros lenguajes de programación, Python permite herencia múltiple, por lo que una misma subclase puede heredar de más de una superclase.

Encapsulamiento

Para que los datos de los objetos no puedan ser modificados directamente por alguien que no deba acceder a ellos, existe el encapsulamiento. Trata de ocultar el estado de los objetos para que solo pueda ser modificado a través de las operaciones definidas para ese objeto.

Por tanto, existirán métodos propios del objeto que serán los encargados de modificar el estado de los objetos. De esta manera, no se necesita conocer ni la manera de trabajar interna del objeto ni las variables que utiliza.

Polimorfismo

El polimorfismo es la técnica por la cuál es posible al llamar a un determinado método de un objeto, obteniendo diferentes resultados en función de la clase del objeto. Esto se explica ya que un mismo método puede ser definido en distintas clases de manera que hagan operaciones diferentes.

```
class Moto(Vehiculo):
    def girar(self):
        print("Girar manillar")

class Coche(Vehiculo):
    def girar(self):
        print("Girar volante")
```

```
>>> moto1 = Moto()
>>> moto1.girar()
"Girar manillar"

>>> coche1 = Coche()
>>> coche1.girar()
"Girar volante"
```

También existe el concepto de sobreescritura de métodos, dónde un método de la superclase es definido de la misma forma por una subclase, pero presentando un comportamiento diferente.

```
class Vehiculo(Object):
    def arrancar(self):
        print("El vehiculo empieza a funcionar")

class Coche(Vehiculo):
    def arrancar(self):
        print("Encender el motor")
```

3.2 Qt Designer

Desarrollado por Haavard Nord y Eik Chanble-Eng, que fundarían en 1994 la compañía Quasar Technologies y que acabaría convirtiéndose en Trolltech, Qt nace en 1992 como un código abierto aunque no totalmente libre. Para utilizarlo era necesaria la compra de su licencia comercial, aunque como código abierto podía usarse a través de la licencia Free Qt, que no permitía la distribución de modificaciones.

Gracias al éxito del escritorio KDE, donde Qt fue utilizada para su desarrollo, empezó a darse a conocer y GNU trató de establecer la biblioteca "Harmony", la cuál sería compatible con Qt pero de forma libre.

Ya en el año 2000, y bajo la licencia GPL, que protege la distribución, modificación y libre uso de Software, empezó a ofrecerse la biblioteca Qt 2.1 solamente para Linux. Se tendría que esperar hasta 2003 para que saliera a la luz la versión bajo GPL para Mac OS X, y no sería hasta 2005 cuando se publicó para Windows. En 2008, aparecería la versión bajo licencia GPL versión 3, y bajo licencia LGPL 2.1 un año después [20].

Qt es un *framework* para el desarrollo de interfaces gráficas multiplataforma de código abierto, con mucha información a disposición de los desarrolladores y que incluye librerías específicas para Python, por lo que fue elegida para la implementación de las interfaces.

Qt Designer es la aplicación dedicada al desarrollo de interfaces de usuario usando este *framework*. Esta aplicación resulta muy intuitiva a la hora de trabajar y permite crear interfaces gráficas elegantes y refinadas con un esfuerzo razonable.

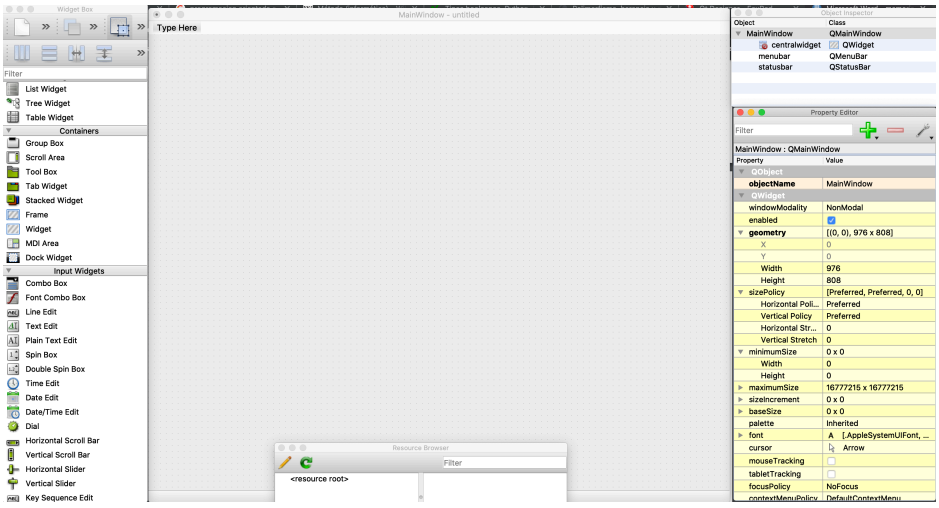

Figura 3.1: Pantalla principal de Qt Designer

Como se puede apreciar en la figura 3.1, inicialmente se presenta una ventana vacía que representa tu interfaz gráfica y, a su alrededor, los diferentes menús que se utilizarán.

El primero menú, llamado "*Widget Box*", presenta los diferentes botones, espaciadores, contenedores, ... que pueden ser arrastrados a la ventana principal de manera que se puede trabajar de una manera muy ágil en la disposición de los diferentes elementos de la interfaz. Uno de estos *widgets* que tiene gran importancia y son más complicados de utilizar, son los *layouts*, con los cuáles se pueden cuadrar los diferentes *widgets* de manera que queden perfectamente alineados en la aplicación. También son de gran importancia los espaciadores, con los que se puede personalizar la distancia que hay entre grupos de *widgets*.

El siguiente menú importante es el de "*Property Editor*". En él se despliegan toda una serie de propiedades y valores correspondientes a un *widget* específico. Es un menú muy interesante para especificar de manera individual campos como el nombre del *widget*, las dimensiones, el texto que aparecerá en la interfaz, si es un botón disponible o no para pulsar, etc.

El "*Object Inspector*" es el menú en el que se muestran todos los objetos organizados mediante una lista jerárquica. Por último, el "*Resource Browser*" es un menú dedicado a los *widgets* que puedan usar algún tipo de recurso, como por ejemplo, un icono.

Qt almacena las interfaces en archivos *ui*. Estos archivos utilizan el formato XML para la representación de los diferentes objetos de la interfaz y sus características. A partir de este archivo *ui* se puede generar una clase Python con la herramienta pyuic5:

```
1 pyuic5 -x ventana.ui -o ventana_ui.py
```

El anterior comando tendrá que ejecutarse cada vez que se hagan modificaciones en la interfaz para que estas modificaciones se trasladen también a la clase correspondiente de Python. El archivo generado "ventana_ui.py" podrá luego ser importado en nuestra clase de Python con:

```
1 from ventana_ui import *
```

Pudiendo así trabajar desde el IDE de desarrollo con el código generado por la aplicación que genera las interfaces gráficas.

3.3 Git

Como herramienta de control de versiones, para la realización de este proyecto, se ha utilizado Git. El uso de un sistema para el control de versiones durante el desarrollo de cualquier producto *software* es esencial, especialmente en aquellos casos donde sea necesario trabajar en equipo. Además, permite mantener un histórico del proyecto desarrollado y recuperar versiones anteriores.

Git ha sido elegido antes que otras herramientas como subversion, por la facilidad que proporciona a la hora de manejar las diferentes ramas del proyecto, especialmente para la realización de pruebas, y por la flexibilidad para poder trabajar con un repositorio local.

3.4 PyCharm

Para facilitar el desarrollo de la aplicación se ha elegido como IDE, PyCharm. Esta elección se debe a todas las utilidades que proporciona, la facilidad de edición de código, la posibilidad de depurar de forma efectiva el código y, además, porque es específico para el lenguaje de programación del proyecto, Python.

3.5 Latex

Para la realización de la memoria se ha utilizado un editor de texto avanzado, orientado a crear documentos de alta calidad que pueden incluir figuras, tablas o expresiones matemáticas.

En concreto, se ha empleado un editor colaborativo para Latex llamado Overleaf.

3.6 Draw.io

Es un editor de diagramas online y gratuito, utilizado para la elaboración de la gran parte de las figuras incluidas en esta memoria.

3.7 Librerías externas

Python es uno de los lenguajes más utilizados hoy en día gracias, sobre todo, a su facilidad de aprendizaje y a su gran flexibilidad ya que permite realizar tanto aplicaciones de escritorio, aplicaciones web, además de poder hacerlo en distintos sistemas operativos. Por ello, a parte de las librerías propias de Python, surgen librerías externas, que ofrecen implementaciones específicas a través de interfaces.

Las principales librerías externas utilizadas para el desarrollo de aplicación son las que se detallan a continuación.

3.7.1 PyQt5

Es el conjunto completo de *bindings* de Qt versión 5 para Python. Se trata de una librería multiplataforma escrita en C++. Incluye *pyuic*, que es el encargado de generar el código en Python a partir del *eXtensible Markup Language* (XML).

3.7.2 NumPy

Es la extensión más utilizada de Python a la hora de trabajar con vectores y matrices ya que incluye funciones matemáticas de muy alto nivel para operar sobre ellos.

Presenta una mejora considerable en el tiempo de ejecución para cálculos numéricos debido a que la manera en que accede a datos esta librería es más eficiente de como lo hace Python con su estructura básica de datos. Las funciones de NumPy realiza menos cálculos intermedios e internamente están programadas en C, que es mucho más rápido que Python [21].

3.7.3 CommPy

CommPy es una librería que proporciona implementaciones de algunos de los algoritmos utilizados para realizar determinadas operaciones en los sistemas de comunicaciones digitales. Para llevar a cabo la implementación de las funciones que proporciona utiliza NumPy y SciPy.

En este proyecto se ha utilizado el módulo convcode.py para las funciones relativas a los códigos convolucionales, ldpc.py para códigos LDPC y turbo.py para los turbo códigos [22].

3.7.4 Matplotlib

Matplotlib es una librería de Python que permite generar gráficas de calidad a partir de datos guardados generalmente en listas o arrays. Es utilizada, sobre todo, en *scripts* de Python, servidores de aplicaciones web e interfaces gráficas de usuario [23].

Es capaz de generar de forma cómoda diagramas, histogramas, curvas de error, gráficos de barras y otros tipos de gráficos con una apariencia visual sencilla y agradable. Además, ofrece una gran cantidad de opciones y utilidades para personalizar el aspecto visual de estos gráficos.

Capítulo 4

Metodología, recursos y planificación

4.1 Metodología

Tanto para el desarrollo de la herramienta *software* como para el desarrollo de la interfaz gráfica se ha llevado a cabo una metodología iterativa e incremental [24], la cuál consiste en dividir el ciclo de vida del proyecto en distintos bloques temporales denominados iteraciones.

El objetivo de esta metodología es que, al finalizar cada iteración, se obtenga un versión funcional del producto que incluya parte de los requisitos establecidos en la fase de análisis. El resultado de una iteración se convierte en el punto de partida de la siguiente, y así se van incrementando las funcionalidades de la aplicación obteniendo un crecimiento progresivo del producto. En la iteración final, se obtiene la versión definitiva del *software* de acuerdo a la especificación de requisitos formulada por el cliente.

Entre las principales ventajas de este tipo de desarrollo encontramos que:

- Gracias a la división en iteraciones, la complejidad de la aplicación se reparte.
- A partir de la primera iteración, siempre se va a tener disponible una versión funcional de la aplicación que se puede mostrar al cliente y evitar así una larga espera que influya negativamente en sus expectativas.
- Es posible detectar errores en el desarrollo del producto y evitar que estos errores se propaguen hasta la versión final del software, haciendo muy costoso solucionarlos.
- A partir de los primeros prototipos se puede obtener experiencia para el desarrollo de los posteriores y recibir información del cliente para mejorar el desarrollo o refinar los requisitos.

4.2 Recursos

Los recursos son una parte fundamental en la elaboración de un proyecto. Comprenden los diferentes medios materiales (*hardware*, *software*, etc) y humanos para la realización del mismo, así como sus costes asociados. La viabilidad económica del proyecto y las desviaciones en sus costes dependen, en gran medida, de una correcta estimación de los recursos necesarios para su desarrollo.

Para la elaboración del proyecto se han considerado tres tipos diferentes de recursos:

4.2.1 Recursos humanos

El grupo de trabajo para la elaboración de este proyecto está formado por tres personas y los roles de cada uno son los siguientes:

- **Director**. Es el encargado de supervisar y ayudar al autor en el desarrollo del proyecto. Este rol es asumido en este proyecto por los directores del mismo, Óscar Fresnedo Arias y José Pablo González Coma.
- **Analista**. Define los requisitos funcionales y no funcionales que debe tener el sistema. Este rol es asumido por el autor.
- **Desarrollador**. Será el encargado de llevar a cabo el diseño y la implementación del *software* del proyecto así como las pruebas del mismo. Este rol es asumido también por el autor.

4.2.2 Recursos software

Los recursos software son los comentados en la sección 3. Es importante destacar que todos las herramientas empleadas son de uso libre y no suponen ningún coste adicional para el proyecto.

4.2.3 Recursos materiales

El único recurso material utilizado en este proyecto es el ordenador personal del autor. De esta forma, los recursos materiales no han supuesto ningún coste adicional en el proyecto debido a que ya se disponía de ellos.

4.3 Planificación

De acuerdo a la filosofía de la metodología incremental considerada para este proyecto, su desarrollo se ha dividido en las siguientes iteraciones:

- **Iteración 1**. Estudio de los componentes de un sistema de comunicaciones digital y elección de las tecnologías necesarias para el desarrollo del simulador.
- **Iteración 2**. Implementación de los esquemas de modulación y demodulación considerando un canal AWGN. Además, se llevará a cabo la implementación de una interfaz gráfica básica para permitir seleccionar determinados parámetros del sistema y mostrar los resultados que se van obteniendo.
- **Iteración 3**. Implementación de esquemas básicos de codificación: códigos bloque lineales y códigos convolucionales.
- **Iteración 4**. Implementación de esquemas modernos de codificación: códigos LDPC y turbo códigos.
- **Iteración 5**. Implementación de los elementos necesarios para simular la transmisión a través de canales Rayleigh y para el uso de sistemas MIMO.
- **Iteración 6**. Refinar y completar la interfaz gráfica final con nuevas opciones para seleccionar todos los parámetros configurables y nuevas utilidades para mostrar resultados e información más detallada.

Además, se ha desglosado cada iteración en las tareas necesarias que se han desarrollado para la realización del proyecto. En la figura 4.1 se puede observar, para cada iteración, las tareas que las conforman, su duración, fechas de inicio y finalización, horas de trabajo, recursos humanos que realizan la tarea y costes asociados a cada una de ellas. En la figura 4.2, se muestra además el diagrama de Gantt correspondiente.

4.4 Análisis económico

Las horas de trabajo dedicadas por los recursos de la sección 4.2.1 para llevar a cabo las diferentes tareas del proyecto tienen un impacto directo en los costes asociados al desarrollo del proyecto. Los costes relativos a estos recursos humanos por cada hora de trabajo se han estimado considerando el informe *Guia Salarial Sector TI Galicia 2015-2016* [25], y son los mostrados en la tabla 4.1.

Recursos humanos	Desarrollador	Analista	Director
Javier Mejuto Vázquez	20,80€	26,00€	-
Óscar Fresnedo Arias	-	-	45,50€
José Pablo Gonzalez Coma	-	-	45,50€

Cuadro 4.1: Costes de los recursos humanos

Análisis general	**30d**	**80 horas**	**16/10/19**	**26/11/19**		**2.197,00 €**
Estudio componentes sistema de comuncicaciones	12d	22 horas	16/10/19	31/10/19	Analista	572,00 €
Elección de las tecnologías	6d	15 horas	01/11/19	08/11/19	Analista	390,00 €
Especificar objetivos de proyecto y requisitos	12d	34 horas	09/11/19	25/11/19	Analista	884,00 €
Revisión del análisis	1d	9 horas	26/11/19	26/11/19	Analista;Directo1;Direc	351,00 €
Versión básica	**34d**	**50 horas**	**27/11/19**	**13/01/20**		**1.094,60 €**
Implementación modulación y demodulación básica	23d	22 horas	27/11/19	27/12/19	Desarrollador	457,60 €
Implementación canal AWGN	16d	15 horas	13/12/19	03/01/20	Desarrollador	312,00 €
Desarrollo de primera versión de interfaz	10d	10 horas	23/12/19	03/01/20	Desarrollador	208,00 €
Revisión versión básica	1d	3 horas	13/01/20	13/01/20	Analista;Directo1;Direc	117,00 €
Implementación esquemas básicos codificación	**34d**	**81 horas**	**14/01/20**	**28/02/20**		**1.794,00 €**
Implementación código Hamming	19d	32 horas	14/01/20	07/02/20	Desarrollador	665,60 €
Implementación códigos convolucionales	21d	35 horas	30/01/20	27/02/20	Desarrollador	728,00 €
Añadir codificaciones a interfaz gráfica	6d	8 horas	20/02/20	27/02/20	Desarrollador	166,40 €
Revisión versión codificaciones basicas	1d	6 horas	28/02/20	28/02/20	Analista;Directo1;Direc	234,00 €
Implementación esquemas modernos codificación	**38d**	**105 horas**	**02/03/20**	**22/04/20**		**2.293,20 €**
Implementación turbo códigos	33d	50 horas	02/03/20	15/04/20	Desarrollador	1.040,00 €
Implementación ldpc	18d	43 horas	25/03/20	17/04/20	Desarrollador	894,40 €
Añadir codificaciones a interfaz gráfica	5d	6 horas	15/04/20	21/04/20	Desarrollador	124,80 €
Revisión versión codificadores modernos	1d	6 horas	22/04/20	22/04/20	Analista;Directo1;Direc	234,00 €
Implementación nuevos canales	**10d**	**60 horas**	**23/04/20**	**06/05/20**		**1.357,20 €**
Canal rayleigh	7d	20 horas	23/04/20	01/05/20	Desarrollador	416,00 €
Sistemas MIMO	5d	34 horas	29/04/20	05/05/20	Desarrollador	707,20 €
Revisión versión nuevos canales	1d	6 horas	06/05/20	06/05/20	Analista;Directo1;Direc	234,00 €
Versión final interfaz gráfica y pruebas	**23d**	**57 horas**	**07/05/20**	**08/06/20**		**1.349,40 €**
Desarrollo de interfaz final	11d	18 horas	07/05/20	21/05/20	Desarrollador	374,40 €
Pruebas y control de errores en la interfaz	16d	30 horas	17/05/20	05/06/20	Desarrollador	624,00 €
Revisión versión final	1d	9 horas	08/06/20	08/06/20	Analista;Directo1;Direc	351,00 €

Figura 4.1: Planificación de las iteraciones

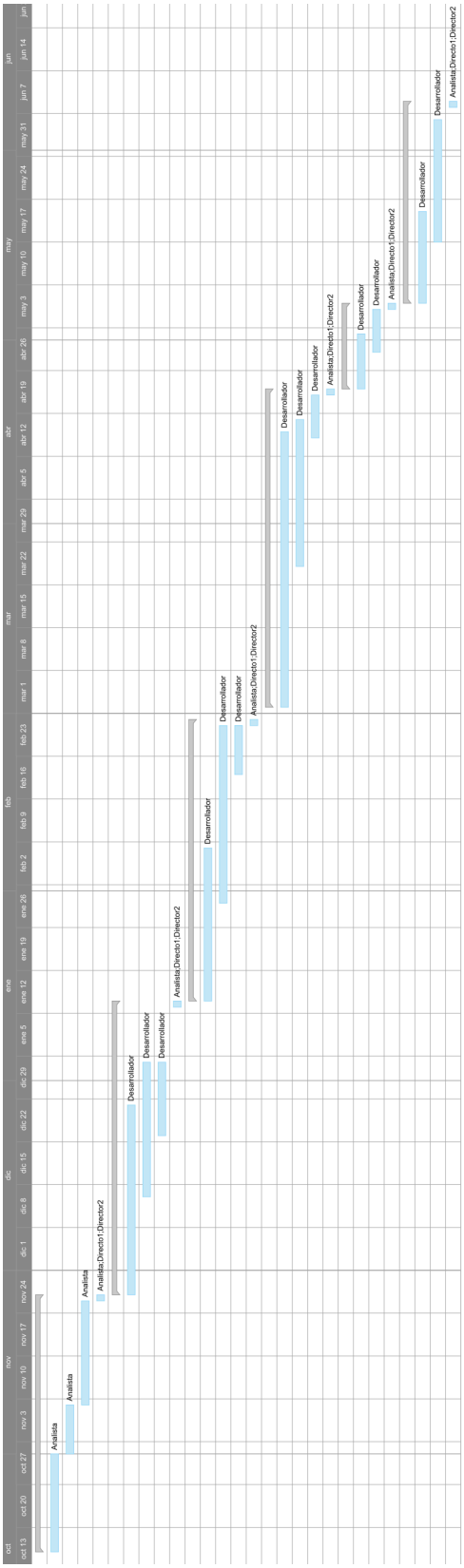

Figura 4.2: Diagrama de Gantt

Como se puede apreciar en las figuras anteriores, la duración de cada iteración ha dependido de la cantidad de trabajo a realizar y de la dificultad del mismo. El proyecto comenzó el 16 de Octubre de 2019 y finalizó el 8 de Junio. Durante este período, el desarrollo del proyecto tuvo que ser detenido durante algunos días, entre los que se incluyen días festivos o días previos a la realización de exámenes. El resumen de toda la información referente al desarrollo del proyecto se muestra en la tabla 4.2

Fecha inicio	16/10/2019
Fecha finalización	08/06/2020
Duración	169 días
Trabajo	433 horas
Coste	10.085,40€

Cuadro 4.2: Resumen general del proyecto

Capítulo 5

Análisis

En un proyecto de *software* el primer proceso de gran importancia es el análisis de requisitos, ya que se establece los servicios que debe proporcionar el producto, así como sus restricciones. El éxito final del proyecto dependerá en gran medida de que este análisis esté realizado de manera cuidadosa. Gracias a la metodología adoptada, los requisitos no identificados en la fase de análisis van apareciendo de forma natural al mostrar los productos intermedios al cliente.

Este proceso de análisis e identificación de requisitos puede llevarse a cabo a través de distintas técnicas, como pueden ser las entrevistas, en las que tanto el cliente como el desarrollador juegan un papel muy importante.

Se han identificado dos clases de requisitos: los requisitos funcionales y los requisitos no funcionales.

5.1 Requisitos funcionales

Estos requisitos se refieren a los servicios que el usuario espera que ofrezca el sistema. También, en algunos casos, puede especificarse lo que no debe hacer el sistema.

Gran parte de los problemas en el desarrollo de *software* es la imprecisión a la hora de concretar estos requisitos entre desarrollador y cliente que, a la larga, puede traducirse en retrasos o costes adicionales.

Para la realización de la aplicación gráfica que permita realizar la simulación de diferentes esquemas de comunicación inalámbrica se han establecido los siguientes requisitos funcionales:

- **Configuración de escenarios**. Uno de los aspectos fundamentales de la aplicación es la flexibilidad a la hora de configurar los distintos componentes del sistema de comunicaciones y la posibilidad de seleccionar los distintos parámetros relevantes para cada uno de estos componentes.

En particular, la aplicación debe permitir de forma sencilla elegir distintos esquemas de codificación: codificación de Hamming, códigos convolucionales, turbo códigos o códigos LDPC y, para cada uno de ellos, configurar los parámetros necesarios que definan el comportamiento de estos esquemas. De forma similar, debe permite seleccionar diferentes tipos de modulación y canales sobre los que transmitir la información. Debido al amplio abanico de posibilidades, nos centraremos en los esquemas de modulación habitualmente utilizados en los estándar de comunicación inalámbrica, es decir, PAM, PSK y QAM, y en canales de tipo Rayleigh con la posibilidad de simular el uso de múltiples antenas en transmisión y/o recepción (canales MIMO).

- **Simulación de la transmisión con el esquema seleccionado**. La aplicación debe implementar todas y cada una de las operaciones necesarias para enviar un determinado mensaje a su destino: generación del mensaje fuente, codificación de la información, modulación de la secuencia codificada, transmisión por el canal y ecualización, demodulación y, finalmente, recuperar el mensaje tras la decodificación. Además, debe ser capaz de estimar las curvas de BER para distintos valores de SNR en el canal.

- **Visualización de resultados.** La aplicación debe permitir obtener, mediante las gráficas presentadas, conclusiones respecto al rendimiento de la configuración seleccionada. Entre los resultados que se deben mostrar destacan las curvas de BER para comparar diferentes esquemas, constelación de símbolos transmitidos y recibidos, estadísticas referentes a la configuración del sistema e información teórica sobre las prestaciones del sistema elegido.

- **Interfaz gráfica**. El sistema debe incluir una interfaz gráfica de uso fácil e intuitivo en la que los resultados más relevantes de la simulación de la comunicación se muestren de manera visual y puedan ser guardados.

- **Manejo de errores**. El sistema tiene que generar respuesta de algún tipo en el caso de que los parámetros elegidos sean incorrectos o determinen una configuración errónea o sin sentido.

5.2 Requisitos no funcionales

Estos requisitos no se refieren a los servicios específicos que entregará el sistema, sino que, son más referidos a las restricciones u obligaciones que incluirá el sistema para garantizar la calidad del mismo. Entre este tipo de requisitos, podemos destacar:

- **Usabilidad**. La aplicación debe ser fácil e intuitiva para los usuarios que la utilicen.

- **Fiabilidad**. La aplicación debe estar disponible en cualquier momento para su uso. Además debe proporcionar resultados correctos para cualquier configuración de entrada.
- **Mantenibilidad**. Capacidad del *software* para que pueda ser fácilmente modificable en caso de correcciones o de añadir nuevas funcionalidades.

5.3 Arquitectura

La arquitectura del proyecto sigue el patrón MVC, mostrado en la figura 5.1, para separar la lógica de la aplicación de la representación de la interfaz gráfica y de la gestión de eventos y comunicaciones. Los componentes en los que se divide la arquitectura son:

- **Modelo**. Contiene la representación de los datos del sistema y gestiona el acceso a ellos, además de definir la lógica de negocio. Envía a la vista los datos solicitados para ser mostrados.
- **Vista**. Es la encargada de presentar los datos y las operaciones que se pueden realizar sobre ellos al usuario, usualmente a través de una interfaz gráfica.
- **Controlador**. Es el encargado de hacer peticiones al modelo cuando se solicita información en la vista. Por eso, actúa como intermediario entre modelo y vista, transformando la información para adaptarlas a sus necesidades.

En el caso concreto de este proyecto, las tareas de cada uno de estos componentes son las siguientes: el modelo se encarga de la lógica de negocios, en la que se implementan las operaciones necesarias para la simulación del sistema de comunicaciones y obtención de información relevante, y del modelo de datos, que guarda los datos más importantes asociados con cada elemento que conforma dicho sistema; la vista, encargada de mostrar las posibles opciones de configuración que tiene el usuario para realizar la simulación, y toda la información relativa a la simulación seleccionada a través de varias gráficas y una tabla con datos relevantes; y el controlador, que trabaja como conector entre la vista y el modelo, el cuál guarda el estado de los parámetros elegidos por el usuario y, en función de estos, el modelo deberá realizar unas operaciones u otras.

Por último, en la figura 5.2, se muestra el diagrama de clases de la aplicación. Como se puede apreciar, hay un controlador específico para cada etapa del sistema de comunicaciones que se encarga de, a partir de las opciones elegidas en la vista, llevar a cabo las operaciones necesarias haciendo uso del *interface* apropiado. Para cada tipo de código, modulador o canal, se implementan de forma específica las operaciones correspondientes, definidas en el *interface* genérico asociado a cada etapa. Toda la información necesaria para llevar a cabo la simulación, y para mostrar o dibujar los datos relevantes se almacena en las clases del modelo de datos. Estos datos serán utilizados, posteriormente, por la vista para su visualización en la aplicación.

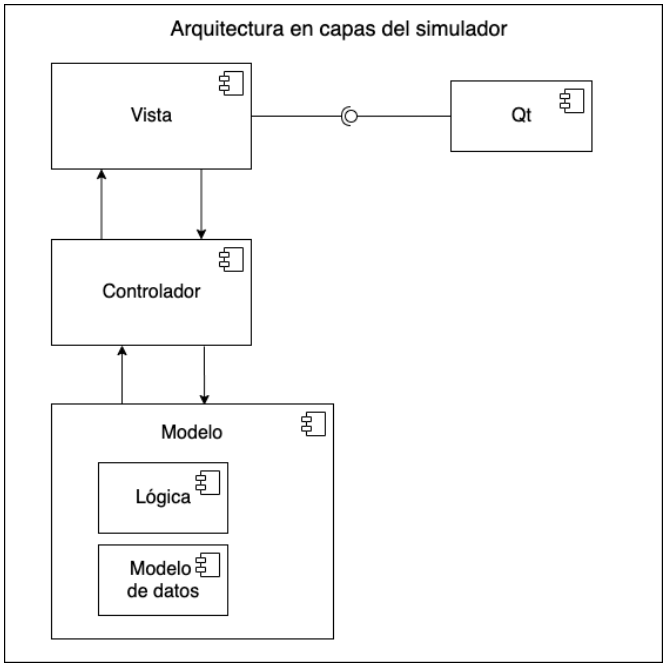

Figura 5.1: Arquitectura MVC de la aplicación

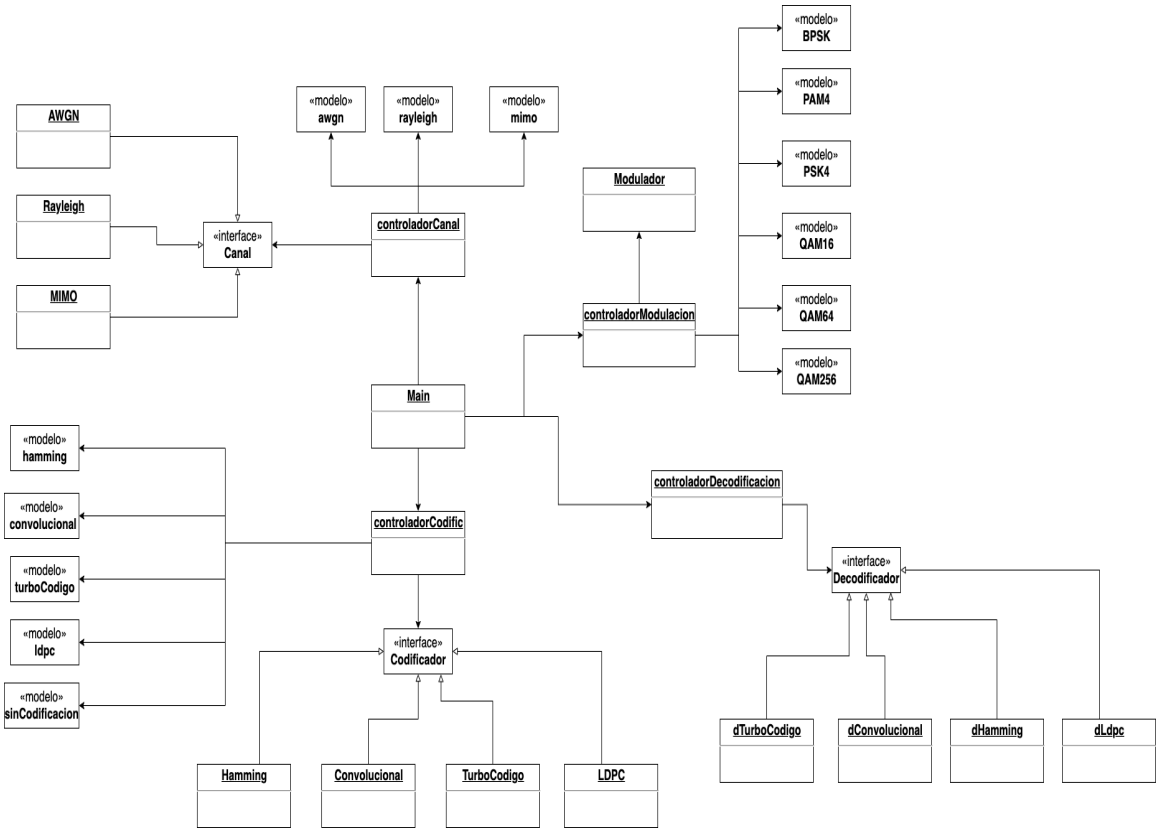

Figura 5.2: Diagrama de clases para el desarrollo de la aplicación

Capítulo 6

Desarrollo

En este capítulo se explicará con detalle las diferentes fases en el desarrollo tanto del *software* de la aplicación como de su interfaz gráfica. El planteamiento que se ha seguido para realizar esta descripción está basado en las diferentes iteraciones que se han establecido para el desarrollo del proyecto de acuerdo a la metodología iterativa incremental usada.

6.1 Iteración 1

Esta primera iteración se basa en analizar y comprender los conceptos que serán desarrollados posteriormente en la aplicación. Se lleva a cabo un estudio de las diferentes fases que se requieren en un sistema de comunicaciones, así como de las diferentes variantes en cada una de ellas explicadas en el capítulo 2.

Además, en esta iteración, se lleva a cabo la elección de las tecnologías con las que se trabajará en el proyecto comentadas en el capítulo 3 y se especifican los objetivos a alcanzar con el proyecto, así como los requisitos necesarios para perfilar las funcionalidades que deberá tener la aplicación una vez esté terminada.

6.2 Iteración 2

En esta iteración se desarrolla una primera versión básica del *software*. Las principales tareas realizadas se pueden resumir en los siguientes puntos:

- Inicialmente, se implementa una clase Modulación que incluye la constelación correspondiente a una modulación BPSK. La función que contiene la clase recibe como parámetro de entrada un número de bits, que representan la información que se quiere transmitir en forma binaria, y devuelve un conjunto de símbolos modulados que se corresponde con la señal portadora de la información en el modelo discreto equivalente.

- A continuación, se implementa la clase Canal, que toma como entrada el conjunto de símbolos obtenidos en el paso previo. Su función será añadir ruido AWGN a los símbolos de entrada simulando lo que sucede en la realidad cuándo la información es transportada a través de un canal AWGN. En estas primeras fases de la implementación, la relación señal a ruido o SNR con la que se simula es muy alta, lo que significa que el número de errores va a ser mínimo, para poder comprobar que los procesos de modulación y demodulación se realizan de manera correcta.

- Por último, se desarrolla la clase Demodulación. En esta primera fase del desarrollo, esta clase contiene los atributos y métodos necesarios para realizar una demodulación *hard* a partir de la información recibida. El parámetro de entrada es el conjunto de símbolos procedentes del canal y, tras el proceso de demodulación, se obtienen una estimación de los bits originales que se han transmitido.

- Una vez verificado que los procesos de modulación y demodulación funcionan de acuerdo a lo esperado, se implementan los cambios necesarios para permitir realizar la simulación para un rango de valores de SNR. Finalmente, se implementan los diferentes esquemas de modulación elegidos para este proyecto así como los dos tipos de ordenamiento de símbolos considerados: ordenamiento natural y mapeado de Gray. Esto nos permite completar una primera versión básica funcional de la aplicación.

- En esta iteración, se implementan además los métodos necesarios para calcular información básica referente a los esquemas de modulación considerados como, por ejemplo, la energía media de símbolo, la eficiencia espectral, la velocidad de transmisión.

En esta iteración, además, se ha desarrollado una primera versión muy sencilla de la interfaz gráfica para poder ver los resultados de las primeras simulaciones. Su diseño en Qt Designer es el mostrado en la figura 6.1. La interfaz incluye una línea de entrada donde se introduce la cantidad de bits con los que se quiere realizar la simulación, *radio buttons* para elegir el tipo de modulación junto con un *check box* en el que se indica si se quiere utilizar o no el ordenamiento basado en el mapeado de Gray.

Además, se incluye una gráfica donde se dibujará la curva de BER para cada uno de los valores de SNR en el rango seleccionado, de acuerdo a las opciones indicadas, y pudiéndose comparar entre las distintas modulaciones.

6.3 Iteración 3

En esta iteración se lleva a cabo la implementación de los dos esquemas básicos de codificación: los códigos de Hamming (7, 4) y los códigos convolucionales. Los principales puntos

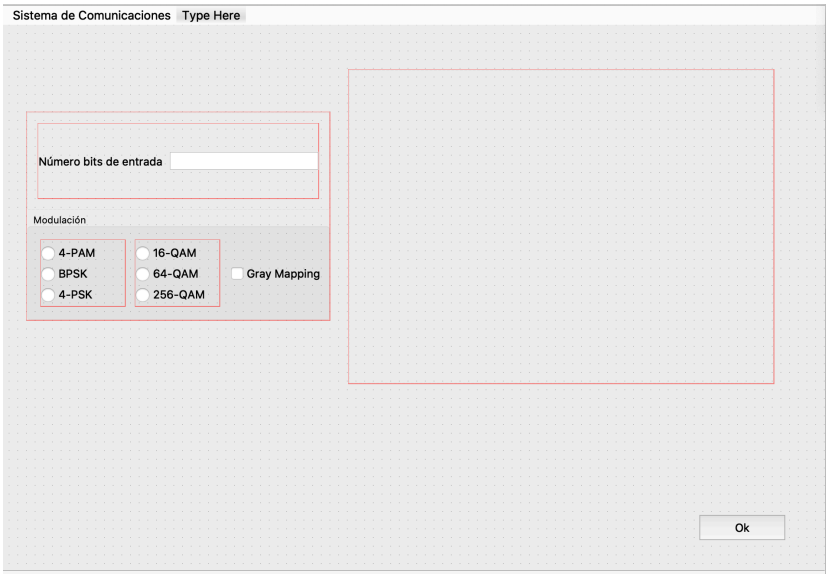

Figura 6.1: Diseño en Qt Designer de la interfaz gráfica

abordados en esta fase del desarrollo son:

- Tratándose ya de una fase avanzada en el desarrollo de la aplicación, se lleva a cabo la implementación siguiendo las directrices de la arquitectura elegida. En particular, se crean controladores para las distintas etapas de un sistema de comunicación. Estos controladores sirven para desacoplar la vista del modelo. Así, en función de los parámetros elegidos en la interfaz gráfica, cada controlador decide que operaciones se tienen que llevar a cabo, y devuelve a la vista la información proporcionada por el modelo que se mostrará en la interfaz.

- Para la implementación de los códigos de Hamming y convolucionales, se crea una nueva clase, Codificador, en la que se realizan las operaciones necesarias para realizar cada una de las codificaciones. También se crea la clase Decodificador, cuyos métodos implementan de forma particular los algoritmos de decodificación necesarios para cada tipo de código.

- Por último, se implementan los métodos necesarios para visualizar el diagrama de bloques y el *trellis* de un código convolucional, junto con la posibilidad de calcular la distancia libre a partir del diagrama de estados o del propio diagrama de *trellis*. En el caso de los códigos de Hamming, la distancia mínima del código es conocida a priori y no es necesario realizar ninguna implementación adicional.

Para adaptar esta etapa del sistema en la interfaz, se añade un menú con un *check box* que permite elegir entre distintos códigos, como puede verse en la figura 6.2. Además, en el caso

de los códigos convolucionales es necesario indicar explícitamente la matriz generadora del código.

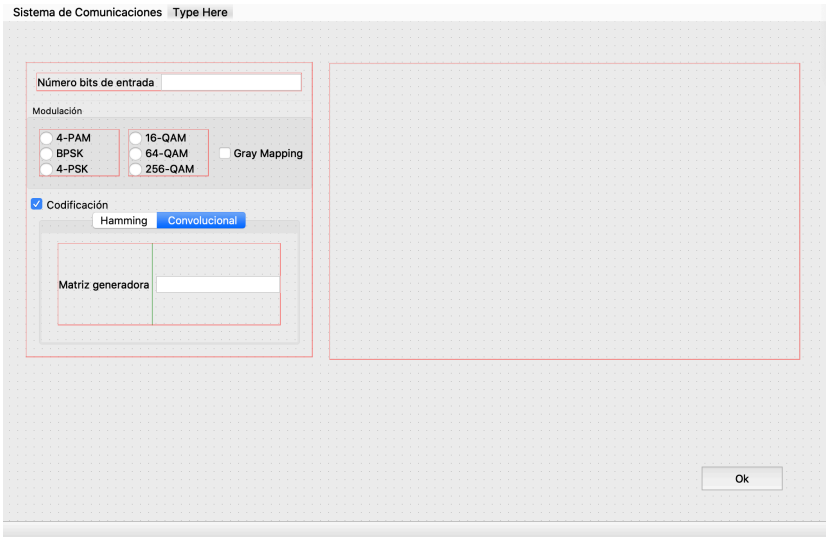

Figura 6.2: Interfaz gráfica al finalizar la iteración 3

6.4 Iteración 4

Esta iteración incluye la implementación de dos esquemas de codificación avanzados: turbo códigos y LDPC.

Se trata de la iteración de mayor duración debido a la complejidad de este tipo códigos. En este caso, se ha hecho uso de varias funciones de la librería Commpy para implementar las operaciones de codificación y decodificación en ambos esquemas de codificación. Sin embargo, ha sido necesario modificar algunos aspectos de la propia implementación de estas funciones para que el funcionamiento se ajustase a lo que se buscaba en el proyecto y a lo esperado a priori. Entre otras cosas, se han tenido que cambiar los siguientes puntos:

- La forma de procesar las *Log-Likelihood Ratio*s (LLRs) en el algoritmo Máximo A Posteriori (MAP) usado en la decodificación de los turbo códigos.
- La forma de recuperar los bits de paridad generados por el segundo codificador convolucional de un turbo código.
- Ha sido necesario implementar a mano la operación de entrelazado y desentrelazado requerida en la codificación y decodificación de estos códigos.
- En el caso de los códigos LDPC, el funcionamiento de las funciones se ha ajustado a lo

esperado, aunque han sido necesario implementar algunos cambios para soportar los dos tipos de ordenamiento considerados en el proyecto, natural y mapeado de Gray.

Para el correcto funcionamiento de los algoritmos de decodificación, se han implementado además las operaciones necesarias para soportar la demodulación *sotf* en el receptor.

Con respecto a la parte visual, se han añadido nuevas pestañas en el menú de la interfaz gráfica, como se muestra en la figura 6.3, para poder simular cada uno de estos códigos. En el caso de los turbo códigos se debe introducir la matriz generadora de los códigos convolucionales que lo forman y, en los códigos LDPC, el fichero donde se especifica el diseño de la matriz control de paridad dispersa. En este proyecto, se puede seleccionar entre dos tipos de códigos LDPC, con *rates* 1/2 o 3/4, que han sido específicamente diseñados para el estándar inalámbrico WiMax.

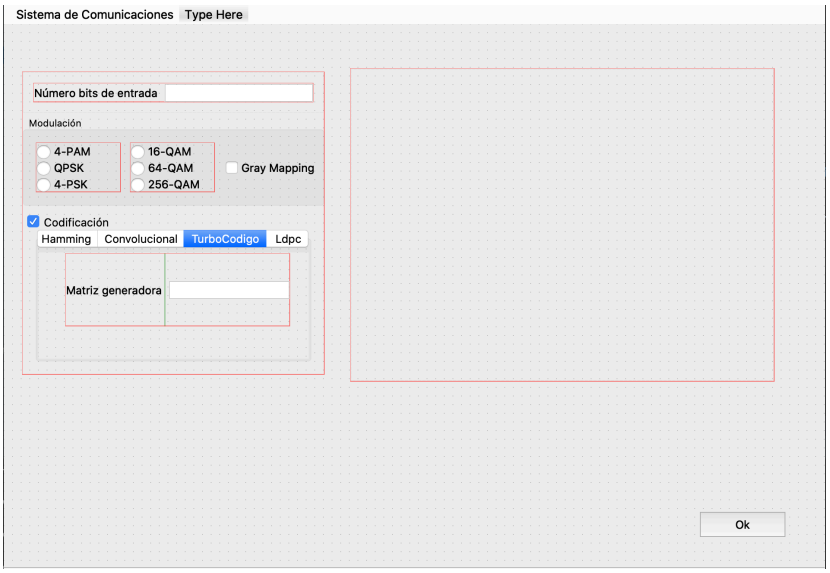

Figura 6.3: Interfaz gráfica al finalizar la iteración 4

6.5 Iteración 5

Hasta esta iteración todas las simulaciones utilizaban como medio por el que se enviaba la información un canal que añadía ruido AWGN. En esta iteración se implementan los otros dos tipos de canales con los que se podrá simular: canales Rayleigh y canales correspondientes a un sistema MIMO. Los principales puntos que se abordaron en esta iteración fueron:

- Se añaden a la clase Canal las funciones con las operaciones necesarios para la implementación de estos dos tipos de canales, destacando que los sistemas MIMO añaden

cierta complejidad ya que trabajan sobre matrices y vectores en lugar de usar escalares debido a las múltiple antenas transmisoras y receptoras.

- Para estes dos tipos de canales, es necesario realizar una operación de ecualización como paso previo a la demodulación de la información recibida. Como se ha comentado en el capítulo 2, nos centraremos en el uso del filtro lineal MMSE.

- Se incorporan los métodos necesarios para establecer la capacidad de este tipo de canales.

Además, como se ve en la figura 6.4, en la interfaz gráfica se añade un menú con la opción de elegir estos tipos de canales junto con el canal AWGN, que ya había sido implementado anteriormente.

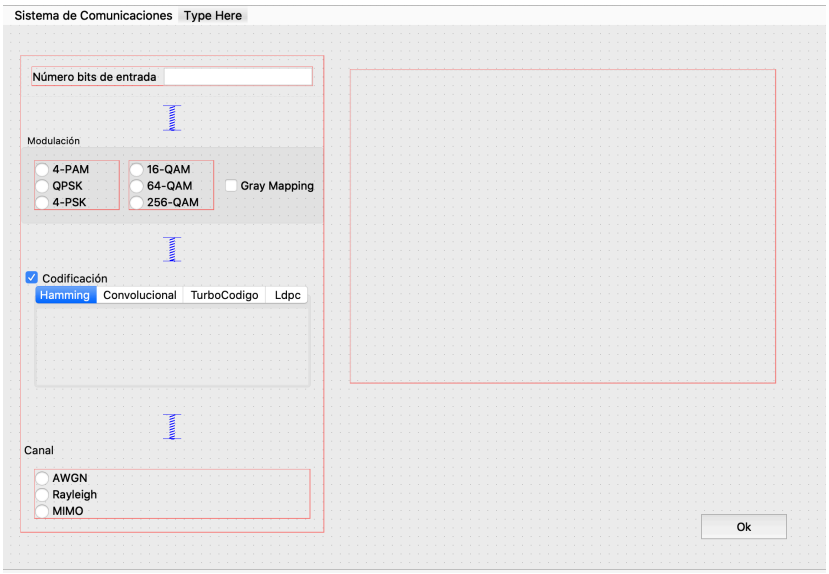

Figura 6.4: Interfaz gráfica al finalizar la iteración 5

6.6 Iteración 6

Esta última iteración se centra en aumentar la información que se muestra en la interfaz gráfica y dar mayor flexibilidad a la configuración de los parámetros de simulación. Las principales funcionalidades que se han añadido son:

- Hasta ahora, se mostraba únicamente la gráfica de BER para un rango de SNRs predefinido. Esta gráfica indica el ratio de bits erróneos que se da en la transmisión de la información con los distintos tipos de codificaciones, modulaciones y canales. En la

versión final de la interfaz, puede ser elegido también el rango de SNR en la que se lleva a cabo la simulación.

- Se añade también una nueva gráfica en la que se dibuja la constelación de la modulación elegida, mostrando además los símbolos que se reciben tras su paso por el canal, permitiendo comparar de forma visual el efecto de emplear una u otra modulación. Ambas gráficas disponen de un menú con diferentes opciones como la de guardar la imagen de la gráfica.

- Se añade una pantalla donde se puede consultar el diagrama de bloques o el diagrama de *trellis* para el código convolucional seleccionado.

- Además se incluye una tabla resumen con los datos más interesantes derivados de la simulación, entre los que están: el *rate* y la distancia mínima del código elegido, la energía medio de símbolo, la tasa efectiva (velocidad) de la transmisión, la capacidad de canal o la eficiencia espectral.

- Para finalizar, en esta iteración se realiza una comprobación exhaustiva de errores verificando que todos los parámetros necesarios sean introducidos y de una forma correcta.

Capítulo 7

Funcionamiento de la herramienta

En este capítulo se muestra el resultado final de la interfaz gráfica desarrollada, se explican algunas de sus funcionalidades más destacadas y la información que proporciona.

La aplicación presenta la interfaz gráfica mostrada en la figura 7.1.

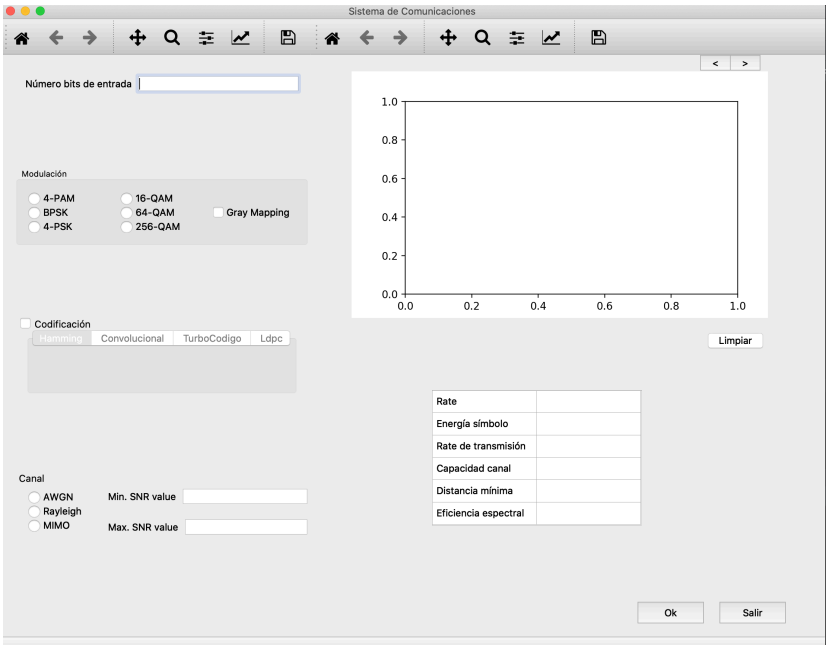

Figura 7.1: Pantalla de la aplicación

En la parte izquierda de la aplicación se encuentran los menús con las posibles opciones a elegir para realizar la simulación, mientras que la parte derecha está destinada a mostrar los resultados obtenidos en los experimentos por medio de gráficas y una tabla informativa. En la parte inferior derecha de la interfaz están los botones para la ejecución de la simulación y para cerrar la aplicación.

Por lo tanto, en primer lugar es necesario elegir los diferentes parámetros con los que se

quiere realizar la simulación, como se muestra en la figura 7.2.

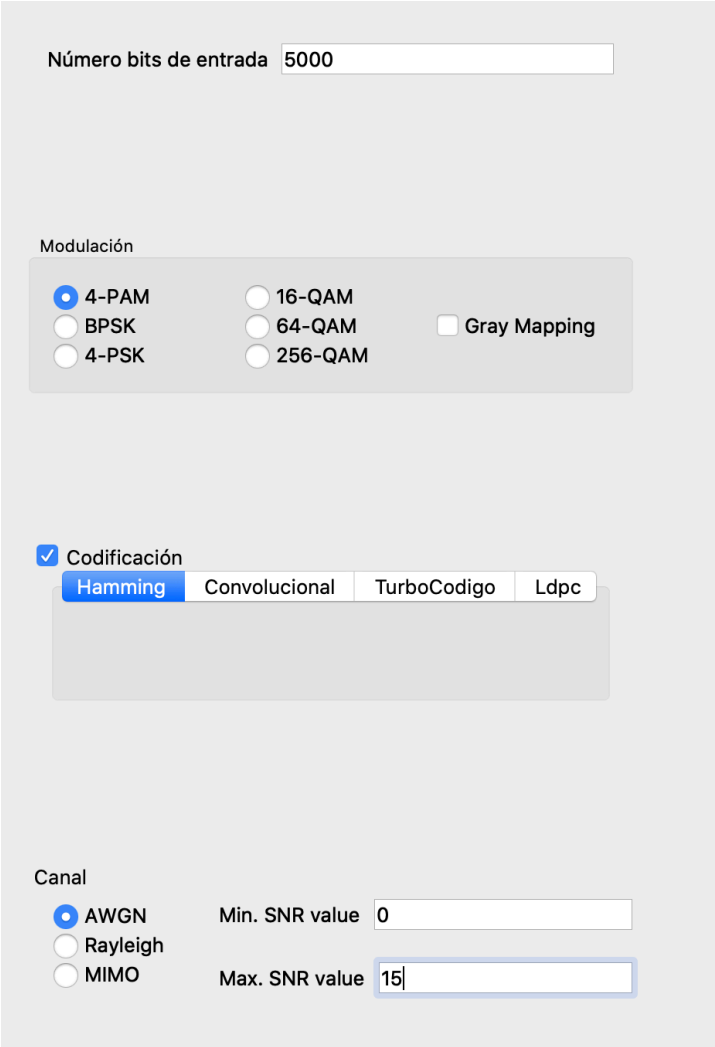

Figura 7.2: Menús para elegir los parámetros del simulador

Se puede observar como es necesario ingresar las siguientes opciones:

- Número de bits fuente que se quieren enviar en la simulación.
- Modulación: esquema de modulación con sus correspondientes niveles y tipo de ordenamiento: natural o mapeado de Gray.
- Codificación: se indica a través de un ”check box” si se quiere utilizar algún tipo de código para la corrección de errores. Una vez marcada esta opción, hay cuatro posibles códigos a utilizar. En el caso de los códigos convolucionales y los turbo códigos es necesario introducir la matriz generadora del código y, para los códigos LDPC, el fichero usado para la generación de la matriz control de paridad.

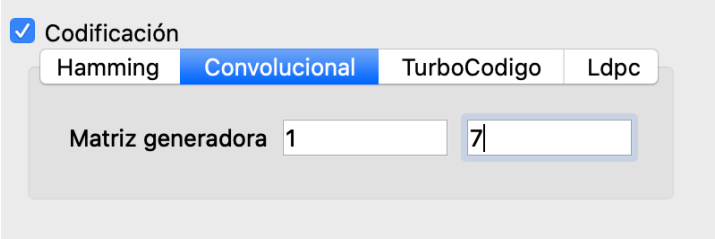

Figura 7.3: Elección de código convolucional para la simulación

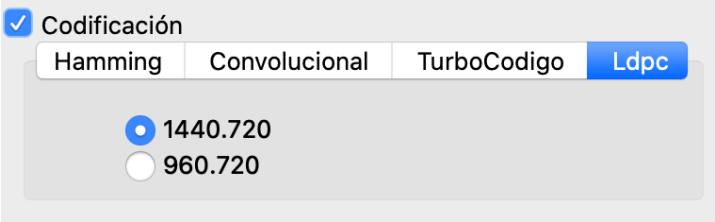

Figura 7.4: Elección del código LDPC para la simulación

- Canal: elección del tipo de canal y selección del rango de valores de SNR para realizar la transmisión sobre el canal.

Una vez elegidos correctamente los parámetro deseados, se pulsa el botón "*Ok*" y se procede a simular la transmisión de la información con la configuración seleccionada. Cuando finaliza la ejecución, observamos dos gráficas y una tabla con información acerca de la simulación elegida.

La primera gráfica muestra la curva de BER que representa el efecto que tiene la relación entre la potencia de la señal y del ruido sobre el número de bits erróneamente detectados en el receptor. De esta forma, se puede apreciar como, a medida que se incrementa la SNR, el número de errores en la información recibida será cada vez más pequeño. Además, la pendiente de la curva indica la magnitud del impacto de la SNR respecto al número de errores. Es decir, si la curva se aplana a partir de cierto valor de SNR, incrementar la potencia con la que se transmite no supondrá beneficios a la hora de detectar los datos.

Mediante esta gráfica es posible comparar el rendimiento para diferentes configuraciones del sistema simulando, como se puede observar en la figura 7.5. Comparando las diferentes curvas podemos determinar que configuración del sistema proporciona mejores prestaciones para un determinado nivel de SNR y comprobar el diferente comportamiento de estas configuraciones. En la leyenda de la figura se indica la combinación de parámetros elegidos y, en el caso de seleccionar la opción de mapeado de Gray para la modulación, se pinta la curva con una línea discontinua.

Mediante las flechas situadas en la parte superior de las gráficas, se puede acceder a la representación de la figura 7.6, que muestra la constelación de los símbolos recibidos corres-

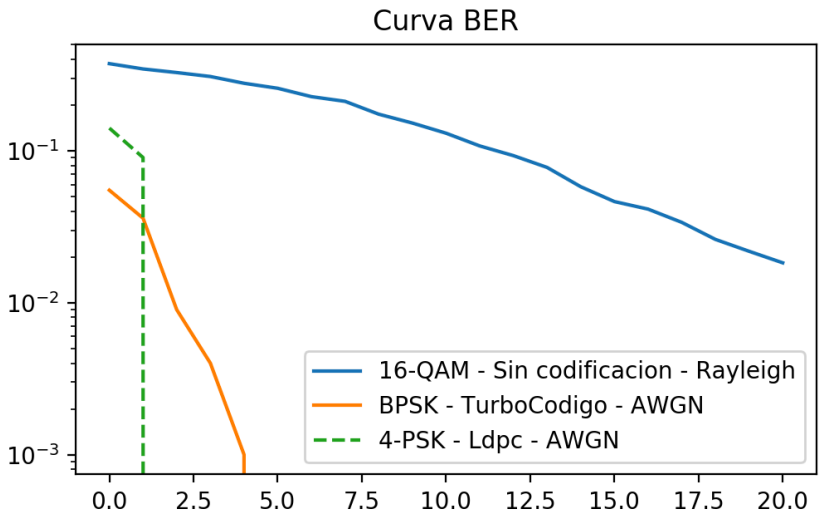

Figura 7.5: Gráfica que muestra las curvas BER para diferentes configuraciones del sistema de comunicaciones

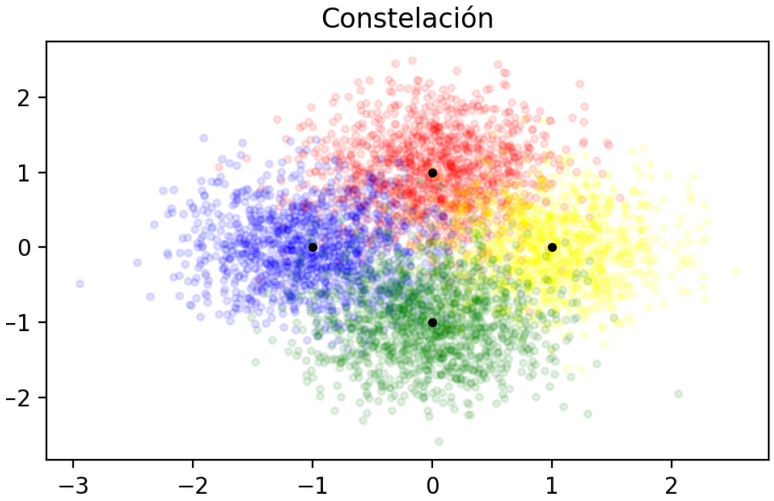

Figura 7.6: Constelación de símbolos recibidos para una modulación QPSK

pondiente a la última simulación. En ella se puede observar los símbolos de la modulación seleccionada, pintada como puntos negros, y, también, los símbolos afectados por el ruido tras su paso por el canal y posterior ecualización. Estos símbolos ruidosos se presentan en diferentes colores, que indican el símbolo realmente transmitido para cada uno de ellos. Se puede apreciar como las nubes de puntos de un determinado color tienen su centro en cada uno de los símbolos de la constelación. Esta gráfica permite visualizar gráficamente como los símbolos ruidosos observados en el receptor pueden llevar a una detección errónea de los símbolos demodulados. Es decir, pueden ser confundidos con símbolos vecinos en el proceso de demodulación, produciéndose así errores en la información final que llega al receptor.

Como funcionalidad adicional para estas gráficas, se incorpora en su parte inferior derecha un botón que permite borrar las curvas actualmente dibujadas sobre la gráfica. Además, cada una dispone de un menú en la parte superior con el que se puede mover, hacer zoom, guardar la gráfica, etc.

Por último, se muestra una tabla con las variables más reseñables en cuánto a la simulación entre los que se encuentran: el *rate* y distancia mínima del código; la energía media de símbolo y eficiencia espectral de la modulación; la capacidad del canal y el *rate* efectivo de la transmisión (bits por uso de canal). Un ejemplo del tipo de información que se muestra se puede apreciar en la figura 7.7.

Rate	1.75
Energía símbolo	1.0000
Rate de transmisión	3.5 b/s
Capacidad canal	3.4594 b/s
Distancia mínima	3
Eficiencia espectral	2

Figura 7.7: Tabla con información de la simulación

7.1 Errores de configuración

La aplicación realiza un control de errores de configuración para evitar la asignación de parámetros incompatibles. A este efecto, si algún campo necesario está vacío o existe alguna combinación de parámetros que no tenga aplicación práctica, se presenta una pantalla que especifica el error en la introducción de los parámetros.

Por ejemplo, en caso de que no se introduzca alguno de los parámetros necesarios para la simulación, se muestra el error de la figura 7.8.

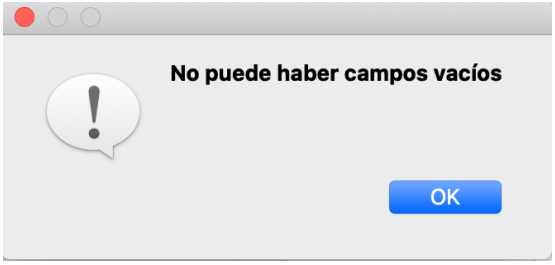

Figura 7.8: Pantalla de error debido a la existencia de un campo vacío

Otros casos como que se introduzca una SNR mínima con un valor mayor que la máxima, o que el número de bits introducidos para la codificación LDPC no sea correcto, se muestran en las figuras 7.9 y 7.10, respectivamente.

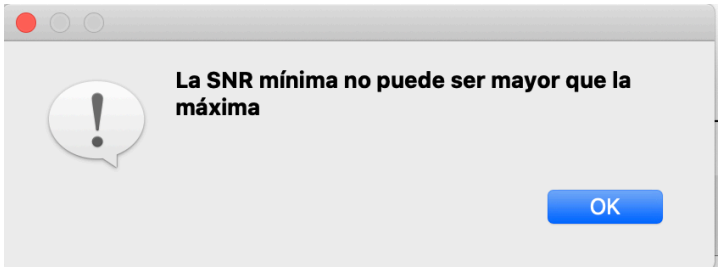

Figura 7.9: Pantalla de error debido a error en la introducción de las SNR

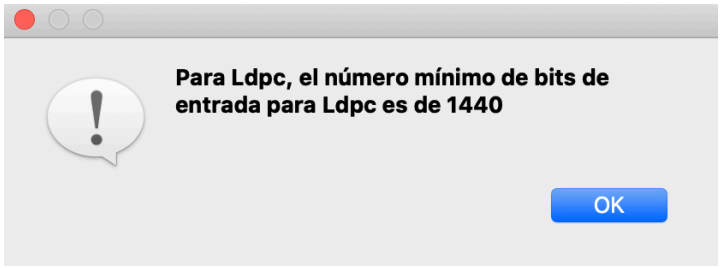

Figura 7.10: Pantalla de error debido a la incorrecta introducción de bits para LDPC

Capítulo 8

Conclusiones y trabajo futuro

En este capítulo se expondrán las conclusiones finales a las que se ha llegado con la realización del proyecto, además de posibles opciones que se podrían considerar en el futuro para la continuación del mismo.

8.1 Conclusiones

El proyecto realizado partía con el objetivo prioritario de comprender en profundidad los principales elementos de un sistema de comunicaciones inalámbricas. Con este objetivo, se llevó a cabo el desarrollo de una herramienta *software* capaz de simular las etapas por las que pasa un mensaje que quiere ser transmitido a través de un canal de comunicaciones y la implementación de una interfaz gráfica que mostrara los resultados. El resultado final es el aprendizaje de conceptos relativos a: codificaciones de canal básicas y modernas, utilizadas en la actualidad; el funcionamiento de los esquemas de modulación necesarias para el envío de información; y los canales de comunicaciones más habituales. Además de esto, ha sido posible llevar a cabo exitosamente el desarrollo de una herramienta gráfica con una interfaz intuitiva. Esta herramienta es capaz de comparar simulaciones con diferentes opciones y mostrar sus resultados de una manera visual y fácil de entender para el usuario.

Gracias a la realización de este trabajo, se han recordado y ampliado la capacidad de manejar distintas tecnologías aprendidas durante el grado, y se han aprendido otras nuevas nunca utilizadas anteriormente. Además, los conocimientos adquiridos durante los últimos años en cuánto a análisis, gestión, diseño y desarrollo de una aplicación, se han puesto en práctica en un proyecto real.

8.2 Trabajo futuro

La implementación del sistema ha sido realizada de manera que sea sencillo incorporar posibles nuevas funcionalidades. Gracias a esto, las posibles líneas de trabajo futuro serían las siguientes:

- Implementación de una librería propia con las funciones necesarias para el correcto funcionamiento de turbo códigos y códigos LDPC con todo tipo de modulaciones.
- Implementación de otros tipos de canales de comunicación como el canal Rician, así como la incorporación de señales de banda ancha o escenarios multiusuario.

Apéndices

Lista de acrónimos

TIC Tecnologías de la Información y la Comunicación.

LDPC *Low-Density Parity-Check code.*

PAM *Pulse-Amplitude Modulation.*

PSK *Phase-Shift Keying*.

QPSK *Quadrature Phase-Shift Keying*.

BPSK *Binary Phase-Shift Keying*.

QAM *Quadrature Amplitude Modulation.*

AWGN *Additive White Gaussian Noise.*

MIMO *Multiple-Input Multiple-Output*.

SNR *Signal-to-Noise Ratio.*

MMSE *Minimum Mean-Square Error*.

MVC Modelo Vista Controlador.

BER *Bit Error Rate.*

MAP Máximo A Posteriori.

LLR *Log-Likelihood Ratio.*

OSI *Open System Interconnection.*

OFDM *Orthogonal Frequency Division Multiplexing*.

XML *eXtensible Markup Language.*

Bibliografía

[1] C. E. S. y Warren Weaver, *A Mathematical Theory of Communication*, 1949.

[2] F. Gutiérrez Bernal and J. Espinoza, "Simulación de sistema de comunicaciones con OFDM basado en GUI de Matlab," Oct, 2013. [En línea]. Disponible en: https://www.researchgate.net/publication/320357064_Simulacion_de_sistema_de_comunicaciones_con_OFDM_basado_en_GUI_de_MatLab

[3] T. Reyes, "Elementos de un sistema de telecomunicaciones. funciones y características," 2017. [En línea]. Disponible en: https://www.monografias.com/docs114/elementos-sistema-telecomunicaciones/elementos-sistema-telecomunicaciones.shtml

[4] "Teoría de la información." [En línea]. Disponible en: https://es.wikipedia.org/wiki/Teor%C3%ADa_de_la_informaci%C3%B3n

[5] "Codificación de canal I: introducción y códigos de bloque." [En línea]. Disponible en: http://openaccess.uoc.edu/webapps/o2/bitstream/10609/63345/6/Teor%C3%ADa%20de%20la%20codificaci%C3%B3n%20y%20modulaciones%20avanzadas_M%C3%B3dulo%202_%20Codificaci%C3%B3n%20de%20canal%20I%3B%20introducci%C3%B3n%20y%20c%C3%B3digos%20de%20bloque.pdf

[6] "Código Hamming." [En línea]. Disponible en: https://es.wikipedia.org/wiki/C%C3%B3digo_Hamming

[7] "Código convolucional." [En línea]. Disponible en: https://es.qwe.wiki/wiki/Convolutional_code

[8] "Turbo código." [En línea]. Disponible en: https://es.wikipedia.org/wiki/Turbo_c%C3%B3digo

[9] R. G. Gallager, *Low Density Parity Check Codes*, 1960.

[10] D. J. M. y Radford M. Neal, *Near Shannon Limit Performance of Low Density Parity Check Codes*, 1995.

[11] P. Turmero, "Canales de transmisión de datos." [En línea]. Disponible en: https://www.monografias.com/trabajos105/canales-transmision-datos-medio/canales-transmision-datos-medio.shtml

[12] "Additive white Gaussian noise." [En línea]. Disponible en: https://en.wikipedia.org/wiki/Additive_white_Gaussian_noise

[13] "Shannon-Hartley teorema - Shannon–Hartley theorem." [En línea]. Disponible en: https://es.qwe.wiki/wiki/Shannon%E2%80%93Hartley_theorem

[14] "About fading." [En línea]. Disponible en: http://rfmw.em.keysight.com/wireless/helpfiles/n5106a/about_fading.htm

[15] "Capacidad de canal en sistemas MIMO." [En línea]. Disponible en: http://www.tsc.uc3m.es/docencia/SyCT/docencia/SyCT/2007/CAPACIDAD%20DE%20CANAL%20EN%20SISTEMAS%20MIMO.pdf

[16] I. E. Telatar, "Capacity of multi-antenna Gaussian channels," *Bell Labs Technical Memorandum*, Jun 1995.

[17] ——, "Capacity of multi-antenna Gaussian channels," *European Trans. on Telecommunications*, vol. 10, no. 6, pp. 585–595, Nov 1999.

[18] "Codigo de bloques lineales." [En línea]. Disponible en: https://es.slideshare.net/Comunicaciones2/codigo-de-bloques-lineales

[19] "Python." [En línea]. Disponible en: https://es.wikipedia.org/wiki/Python

[20] "Tutorial de Qt4 Designer y QDevelop." [En línea]. Disponible en: https://upcommons.upc.edu/bitstream/handle/2099.1/7656/memoriafinalPFC.pdf?sequence=1&isAllowed=y

[21] "Numpy." [En línea]. Disponible en: https://es.wikipedia.org/wiki/NumPy

[22] "Commpy." [En línea]. Disponible en: https://pypi.org/project/scikit-commpy/

[23] "Matplotlib." [En línea]. Disponible en: https://pypi.org/project/matplotlib/

[24] Tutorialspoint, "SDLC - iterative incremental model," accedido por última vez el 16/06/2020. [En línea]. Disponible en: https://www.tutorialspoint.com/adaptive_software_development/sdlc_iterative_incremental_model.htm

[25] “GuiaSalarial.” [En línea]. Disponible en: https://es.scribd.com/document/288511179/Guia-Salarial-Sector-TI-Galicia-2015-2016