3

Descripción de la simulación

3.1. Introducción

A lo largo del siguiente capítulo vamos a describir diferentes aspectos de las simulaciones, como el programa empleado, la estructura de la simulación y las distintas funciones empleadas.

El objetivo de este capítulo es mostrar al lector la estructura llevada a cabo y describirle los elementos de la simulación en vistas a su posible utilización en otros proyectos.

3.2. Matlab

Para realizar la comparación de los distintos códigos y modulaciones, se ha empleado la versión 7.10.0.499 (R2010a) de Matlab.

MATLAB [1] es un software matemático que ofrece un entorno de desarrollo integrado con un lenguaje de programación propio. Además incluye numerosos *toolbox* con un gran número de funciones, además de ejemplos y demos que sirven de gran ayuda junto al manual que incorpora.

Para la realización del proyecto, hemos empleado el *toolbox* de comunicaciones, dado que encontramos en el funciones que nos son de gran utilidad.

3.3. Estructura de la Solución

La estructura seguida para llevar a cabo las simulaciones es parecida a la vista en la figura 2.1 en la sección 2.1. Sin embargo, aquí hemos omitido el codificador de fuente por simplicidad.

[1] *MATrix LABoratory*

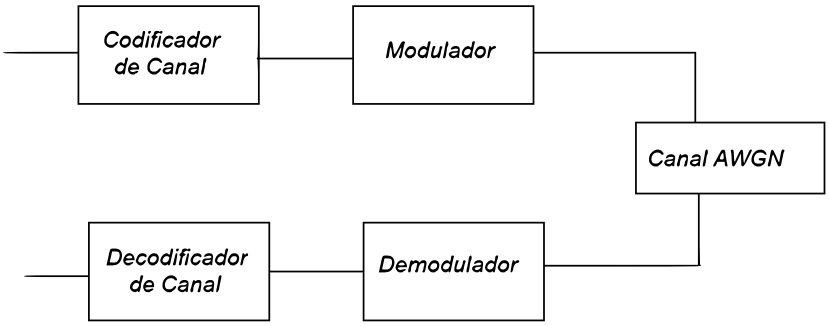

Figura 3.1: Esquema de la Simulación

Para simular el sistema de comunicaciones, como podemos ver en el grafo de la figura 3.2, primero generamos una secuencia de bits aleatorios. Una vez generados los bits aleatorios, definimos el codificador a emplear, Hamming o convolucional, a través de sus parámetros, y codificamos los datos. Una vez tenemos los datos codificados, el siguiente paso es modularlos. Tras este paso estaría definido el Transmisor.

El siguiente paso es definir el canal y el receptor. Debido a que queremos medir la *BER* en diferentes escenarios de ruido, es necesario hacer un bucle para repetir el proceso para los distintos valores de E_b/N_0. De acuerdo a estos valores, el canal introducirá más o menos ruido, lo que influirá directamente en la tasa de error, ya que a mayor ruido, mayor probabilidad de error.

A continuación, se encuentra el receptor, formado por el demodulador y el decodificador. Del decodificador sale una secuencia de bits que será bastante similar a la entrada de la entrada del sistema.

Como estas dos secuencias no son exactamente iguales, se habrán producido errores. Tras decodificar los datos, tenemos un módulo en el que se comparan la secuencia original y la recibida para calcular la tasa de error o *BER*.

A continuación vamos a describir el método empleado para calcular la *BER*, llamado **Método de Montecarlo**, así como las distintas partes del sistema, codificador, modulador, detallando como se han implementado en Matlab.

3.3.1. Método Montecarlo

El método de Montecarlo es un método no determinista o estadístico numérico, usado para aproximar expresiones matemáticas complejas y costosas de evaluar con exactitud. El método se llamó así en referencia al Casino de Montecarlo (Principado de Mónaco), al ser la ruleta un generador simple de números aleatorios. El nombre y el desarrollo sistemático de los métodos de Montecarlo datan aproximadamente de 1944.

En estos métodos el error es $1/\sqrt{N}$, donde N es el número de pruebas y, por tanto, ganar una cifra decimal en la precisión implica aumentar N en 100 veces.

Para estimar la *BER*, vamos a realizar una serie de experimentos, transmitir una serie de datos como se expone en [8]. Una vez transmitidos y recibidos, vamos a contar el número de sucesos, el número de errores, y dividir por el número de ensayos o bits

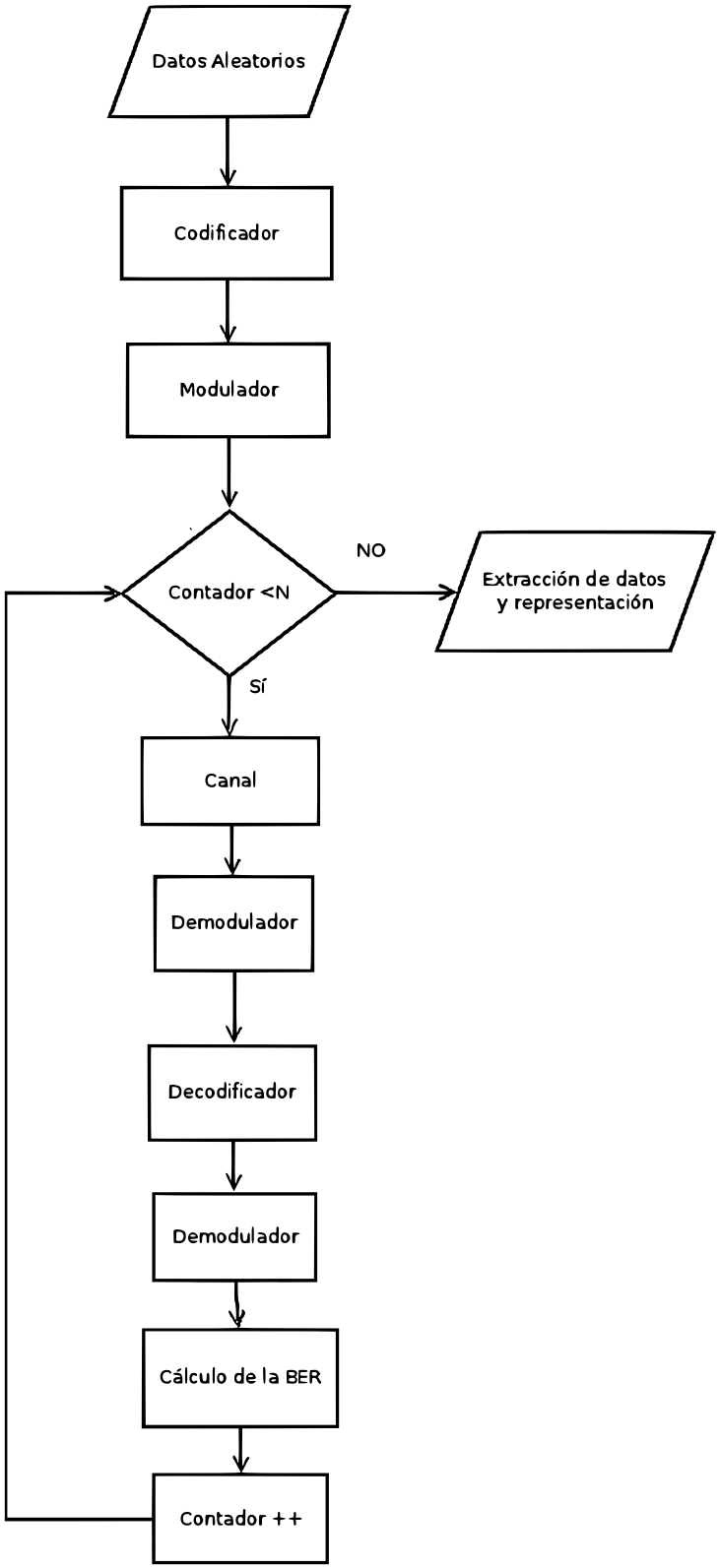

Figura 3.2: Diagrama de Flujo de la Simulación

transmitidos.

3.3.2. Transmisor

Codificador

Como hemos dicho, los codificadores que hemos empleado son codificadores de Hamming y codificadores convolucionales. Concretamente, hemos empleado los códigos Hamming (7,4), (15,11) y (31,26)[2] y los códigos convolucionales definidos por los parámetros *coderate* o tasa del código r, *Codegenerator* o código generador del código, son los coeficientes en octal del polinomio generador,*constraintlength* o K, que indica el retraso en el codificador y *tblen*, parámetro empleado a la hora de decodificar, que suele tener como valor 5 veces K y que representa la profundidad en la decodificación, también es conocido como *traceback depth*.

Nombre	Coderate	Codegenerator	Constraintlength	tblen
$Conv3$	1/2	5	[23 35]	25
$Conv1$	1/2	7	[133 171]	35
$Conv2$	1/2	9	[561 753]	45
$Conv2_1$	1/3	5	[25 33 37]	25
$Conv2_2$	1/3	7	[133 145 175]	35
$Conv2_3$	1/3	9	[557 663 711]	45

Cuadro 3.1: Parámetros Códigos Convolucionales

Los códigos Convolucionales de 3.1 han sido extraídos de [4].

Para los codificadores de Hamming empleamos la función **encode** de Matlab, que se trata de un codificador de bloque, pudiendo elegir el tipo de codificación de bloque (lineal, Hamming o cíclica).

Para los codificadores convolucionales, usamos la función **convenc** que codifica el mensaje a través del código definido mediante su Trellis. Para definirlo con el Trellis usamos la función **poly2trellis** que a partir de la Constraintlength y del Codegenerator lo genera.

Modulador

El modulador consta de dos partes, una que se encarga de modular, y otra que forma el pulso conformador empleado para la transmisión.

Para la parte de la modulación, tanto para las modulaciones *QAM* como para las *PSK* tenemos que crear el modulador a través del objeto **modem.qammod** y **modem.pskmod** respectivamente, indicando los parámetros necesarios,M, el *offset* de la fase, el tipo de salida de los datos ... Para que se haga efectiva la modulación usamos **modulate**.

Para el pulso conformador, podemos elegir entre un pulso rectangular, función *rectpulse* o un pulso coseno alzado, mediante **fdesign.pulseshaping**.

[2]Recordemos que los códigos de Hamming se definen por los parámetros (n,k) como dijimos en 2.3.2

3.3.3. Canal

Como dijimos anteriormente, el canal empleado es un canal AWGN al cuál vamos a variar su relación E_b/N_0. Matlab tiene una función que implementa este canal es la función **awgn**. Esta función trabaja con la *SNR*, por lo que debemos relacionar E_b/N_0 con ella. Debido al muestreo, la codificación y la modulación, debemos realizar unos ajustes. La *SNR* vendrá dada por:

$$SNR = \frac{E_b}{N_0} - 10\log_{10}(N_{Sam}) + 10\log_{10}(k) + 10\log_{10}(codeRate) \tag{3.1}$$

Donde N_{Sam} es la tasa de muestreo y $k = \log_2(M)$.

3.3.4. Receptor

Demodulador

De forma parecida al modulador, existen demoduladores en Matlab para dichas modulaciones **modem.pskdemod** y **modem.qamdemod** junto a **demodulate** para que se lleve a cabo la demodulación.

Decodificador

Dependiendo de la codificación, realizamos un tipo de decodificación. Para los códigos Hamming, usamos la función **decode** que decodifica códigos de bloque. Los códigos convolucionales, usan el algoritmo de Viterbi como dijimos en 2.3.3, para emplear este algoritmo usamos la función **vitdec**.

Cálculo de la BER

Para hacer la comparación entre la secuencia original y la recibida, empleamos una función que las compara y nos devuelve el número de errores, **biterr**. Tenemos que tener en cuenta a la hora de comparar los posibles retardos introducidos por el decodificador.