ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

ESTUDIO Y SIMULACIÓN DE TURBO CÓDIGOS UTILIZANDO EL ALGORITMO MAP Y SOVA

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

DIEGO FERNANDO VALLEJO HUANGA

diegovallejo1985@gmail.com

DIRECTOR: Ing. Robin Álvarez, MSc, PhD

arobin7es@yahoo.es

Quito, Septiembre 2011

DECLARACIÓN

Yo, Diego Fernando Vallejo Huanga, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional y que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____

Diego Fernando Vallejo Huanga

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Diego Fernando Vallejo Huanga, bajo mi supervisión.

_____

Ing. Robin Álvarez, MSc, PhD

DIRECTOR DE PROYECTO

AGRADECIMIENTOS

Deseo expresar mis más sincera muestra de agradecimiento, a DIOS sobre todas las cosas, por mantenerme en su camino y permitirme seguirle.

De gran importancia es para mi mencionar la inmensa gratitud que tengo a mis padres María y Hugo, por su fortaleza en la vida, su gran sabiduría, su amor incondicional, por enseñarme lo que este proyecto no puede mostrar pero de alguna manera ha plasmado; por indicarme que la meta más difícil se alcanza si se la realiza con amor, convicción y trabajo.

A mi hermano Oscar por su invaluable apoyo y su gran amistad.

Condición sine qua non para que hubiese sido posible realizar este trabajo es el apoyo de mis amigos Patricio, Marco, Fabricio, Diego, Edwin, Fernando y a los que en este momento no recuerdo pero que han sido fundamentales en mi desarrollo personal y profesional.

Especial sentido tiene la vida cuando es compartida con amor y alegría junto a la persona que se ama, a esa persona especial Pauly, le agradezco su amor y paciencia.

Piedra angular en mi formación profesional han sido mis profesores, por lo que agradezco a cada uno de mis maestros, por su ética e incansable labor; en especial al profesor MAJARI por su inmensa sabiduría.

A mi director de tesis el Doctor Robín Álvarez, mentalizador de este proyecto, le hago extensivo mi agradecimiento.

DEDICATORIA

Dedico este proyecto de titulación a mis padres, amigos, hermano, novia y a cada persona que desconoce o subestima su ayuda prestada en algún momento de mi vida.

ÍNDICE DE CONTENIDOS

CAPITULO I INTRODUCCIÓN A LOS SISTEMAS DE COMUNICACIÓN

CAPITULO II ESQUEMAS DE CODIFICACIÓN Y TURBO CÓDIGOS

ÍNDICE DE FIGURAS

CAPITULO I INTRODUCCIÓN A LOS SISTEMAS DE COMUNICACIÓN

CAPITULO II ESQUEMAS DE CODIFICACIÓN Y TURBO CÓDIGOS

CAPITULO III DISEÑO, SIMULACIÓN, PRUEBAS Y ANÁLISIS DE RESULTADOS DEL SOFTWARE

ÍNDICE DE TABLAS

CAPITULO I INTRODUCCIÓN A LOS SISTEMAS DE COMUNICACIÓN

CAPITULO II ESQUEMAS DE CODIFICACIÓN Y TURBO CÓDIGOS

CAPITULO III DISEÑO, SIMULACIÓN, PRUEBAS Y ANÁLISIS DE RESULTADOS DEL SOFTWARE

ÍNDICE DE ESPACIOS DE CÓDIGO

CAPITULO I INTRODUCCIÓN A LOS SISTEMAS DE COMUNICACIÓN

CAPITULO II ESQUEMAS DE CODIFICACIÓN Y TURBO CÓDIGOS

CAPITULO III DISEÑO, SIMULACIÓN, PRUEBAS Y ANÁLISIS DE RESULTADOS DEL SOFTWARE

RESUMEN

El presente proyecto de titulación describe el proceso necesario para el diseño de un software didáctico que permita la Turbo Codificación y Decodificación basada en la idea del algoritmo MAP y SOVA, mediante el programa computacional Matlab. Para este fin, el contenido del proyecto se ha divido en cuatro capítulos que tratan de lo siguiente:

El primer capítulo, realiza un análisis introductorio a los sistemas de comunicación digital existentes bajo los cuales se va a ejecutar el proyecto con un enfoque en el canal de comunicaciones y además se estudia la teoría de codificación.

En el segundo capítulo se estudian de manera directa los esquemas de codificación FEC existentes bajo los cuales se desarrolla el estudio de los turbo códigos. Además se examinan las aplicaciones de los Turbo Códigos en esquemas de comunicaciones digitales terrestres e inalámbricos.

El tercer capítulo efectúa el planteamiento del sistema de turbo codificación y decodificación a simular, el cual es acompañado con los conceptos y principios matemáticos que lo rigen. Se desarrolla el algoritmo del programa y las sub-funciones que se van emplear con el respectivo interfaz gráfico de usuario. Se utilizan diagramas de flujo para explicar de manera clara las funciones utilizadas y la estructura. Se registran las pruebas realizadas y se hace un análisis, al que se agregan observaciones generales.

El cuarto capítulo presenta las conclusiones y recomendaciones que han surgido después de la realización del proyecto; y finalmente se presentan los anexos, los cuales permitirán una mejor comprensión del trabajo propuesto.

PRESENTACIÓN

Un Turbo Código es un tipo de código de corrección de errores en comunicaciones digitales, son considerados el método más eficaz de generación de códigos en la actualidad. Se consideran un elemento imprescindible en las comunicaciones inalámbricas y vía satélite modernas, no solo por la corrección de errores sino también por que necesita menos potencia de transmisión. Por lo que se han convertido en una de las invenciones más importantes de la teoría de comunicaciones de los últimos años. En estos momentos los Turbo Códigos están considerados entre los mejores esquemas de corrección de errores conocidos para comunicaciones digitales, debido a la posibilidad que ofrecen de obtener los limites teóricos de capacidad de canal planteados por Shannon.

El consumo de baja potencia de transmisión que esto implica, ha generado el desarrollo de grandes investigaciones en el tema en universidades y compañías de telecomunicaciones, como Sony, Sanyo, Lucent, entre otras. El presente proyecto forma parte de una visión a largo plazo, que permitirá la implementación de Turbo Códigos en sistemas empotrados aplicados a las comunicaciones, y como tal, plantea el desarrollo de un software didáctico de simulación de Turbo Códigos, mediante una plataforma interactiva de cómputo, basada en Matlab.

La plataforma computacional se basa en los fundamentos teóricos de Turbo Códigos, así como los fundamentos matemáticos que sustentan esta técnica de corrección de errores. Se busca incorporar al usuario de manera sencilla al proceso de turbo codificación y turbo decodificación por medio de diagramas esquemáticos que constituyen un turbo codificador y decodificador. Por tal razón, se hará una explicación con el objeto de dar un panorama general con énfasis en procesos de codificación que permitan modelar la generación y recuperación de un mensaje.

CAPÍTULO 1

INTRODUCCIÓN A LOS SISTEMAS DE COMUNICACIÓN

1.1 INTRODUCCIÓN

[1]En la actualidad uno de los problemas existentes en las comunicaciones es enviar de manera confiable información libre de errores a través de un canal de comunicación, aprovechando la capacidad máxima del canal de acuerdo al teorema de Claude Shannon, en donde se demostró teóricamente que para cualquier tasa de transmisión menor o igual a la capacidad de canal, existe un esquema de codificación que alcanza una probabilidad de error arbitrariamente pequeña, y por lo tanto se puede hacer la transmisión sobre el canal muy confiable y bajo una potencia de transmisión estrictamente necesaria.

Es así que mediante este capítulo se pretende realizar un análisis introductorio a la estructura del sistema de comunicación digital bajo el cual se va a ejecutar el proyecto, con un enfoque en el canal de comunicaciones, en base a los principios de Shannon. Se incluyen simulaciones mediante el programa computacional Matlab de los principales tópicos tratados, con el fin de orientar y plasmar de manera gráfica el estudio.

1.2 SISTEMAS DE COMUNICACIÓN DIGITAL

[2]La Figura 1.1 muestra la configuración básica de un enlace de comunicación digital punto a punto. Las operaciones básicas de procesamiento de señales en un sistema de comunicación digital son codificación de la fuente, codificación de canal y modulación; así como los procesos inversos en el lado receptor. Los códigos para detección y corrección de errores corresponden a la codificación de canal.

---

[1] FUENTE: SACANAMBOY, Maribell; Tesis de Maestría: Diseño e Implementación de los Turbo Codificadores definidos en los estándares de Telecomunicaciones cdma2000 (TIA/EIA 2002.2D) y WCDMA (3GPP TS 25.212 v7.2.0) usando Hardware Reconfigurable; Pág: 2.

[2] FUENTE: SCHLEGEL, Christian B.; PÉREZ Lance C.; Trellis and Turbo Coding; Pág: 3.

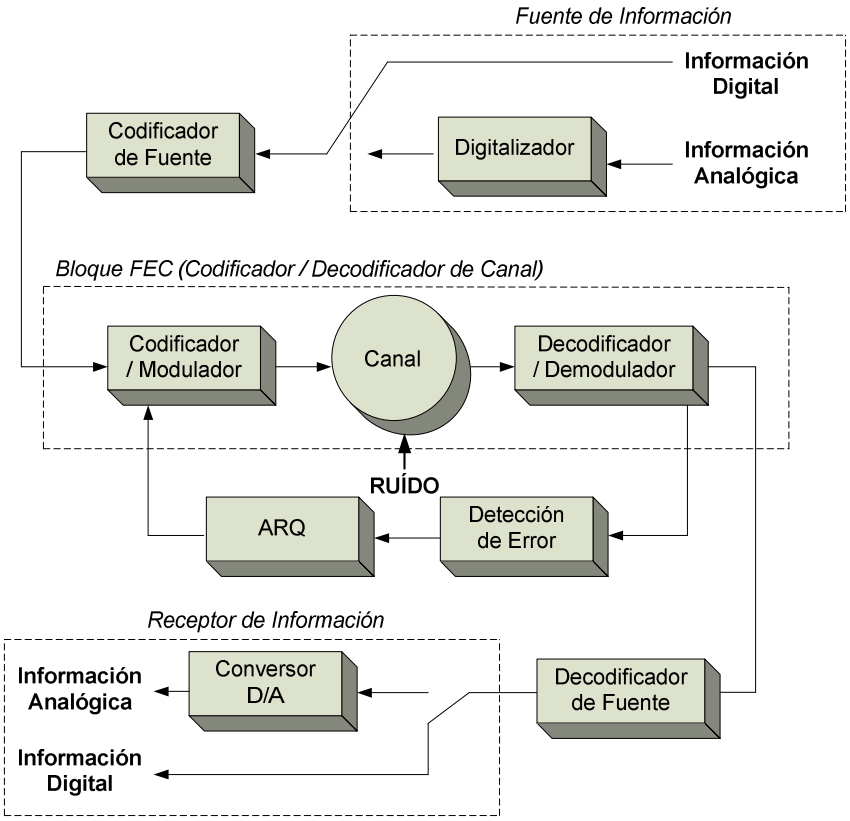


Figura 1.1. Diagrama de Bloques de un Sistema de Comunicación Digital.[3]

[4]Uno de los parámetros que miden el desempeño de un sistema de comunicación digital es la tasa de bits errados (BER) para un cierto valor de la relación de energía de bit a densidad espectral de ruido (E_b/N_o). Generalmente, debido al ruido del canal de transmisión, la única manera de proporcionar un valor de BER aceptable es agregando códigos para detección y corrección de los errores.

3.3.1 FUENTE DE INFORMACIÓN

[5]Son los datos que van a ser comunicados, tales como un archivo de computadora, una secuencia de vídeo o una conversación telefónica. Para los propósitos del proyecto, es representada en forma digital (fuente discreta), tal vez como resultado de la conversión de analógico a digital. Teóricamente en la información, las fuentes son vistas como secuencias de números aleatorios que se rigen por alguna distribución de probabilidad.

[3] FUENTE: SCHLEGEL, Christian B.; PÉREZ Lance C.; Trellis and Turbo Coding; Pág: 3.
[4] FUENTE: ALVARADO, Raúl; Códigos para Detección y Corrección de Errores en Comunicaciones Digitales; Págs: 51.
[5] FUENTE: MOON, Todd K.; Error Correction Coding: Mathematical Methods and Algorithms; Págs: 5-6.

Cada fuente de datos tiene una medida de la información que representa; la misma que (en principio) puede ser exactamente cuantificada en términos de la entropía[6]. Si se tiene una fuente de información analógica (formas de onda continuas en el tiempo), es necesario realizar una conversión A/D[7] a través de un digitalizador.

3.3.2 CODIFICADOR DE FUENTE

[8]Por lo general, los datos digitales son codificados en la fuente para eliminar la redundancia innecesaria de la misma, es decir, los datos de la fuente son comprimidos. Esta codificación de la fuente produce un efecto en el cual los datos digitales que entran en el codificador tiene estadísticas que se asemejan a la de una fuente de símbolos aleatoria con la máxima entropía, es decir, todos los símbolos digitales ocurren con igual probabilidad y son estadísticamente independientes. Si la fuente es digital no es necesario un codificador de fuente.

[9]El codificador de fuente emplea tipos de código especiales para hacer la compresión de datos, los que colectivamente se los llama como códigos fuente o códigos de compresión de datos. Entre estas técnicas de codificación están el código de Huffman, la codificación *Run-Length*, codificación aritmética, codificación Lempel-Ziv, y combinaciones de estos; todos los cuales quedan fuera del alcance de este proyecto de titulación. En este bloque también se emplea la encriptación de la información para la seguridad de la misma, utilizándose algoritmos de cifrado como el AES y el DES.

3.3.3 CODIFICADOR DE CANAL

[6] ENTROPÍA: cantidad de información por símbolo que genera en promedio la fuente.
[7] A/D: Conversión analógica-digital, consiste en la transcripción de señales analógicas en señales digitales.
[8] FUENTE: SCHLEGEL, Christian B.; PÉREZ Lance C.; Trellis and Turbo Coding; Pág: 3.
[9] FUENTE: MOON, Todd K.; Error Correction Coding: Mathematical Methods and Algorithms; Pág: 7.

[10]El codificador de canal tiene como tarea operar en estos datos comprimidos, adicionando redundancia al mensaje de tal manera que ante la presencia de ruido en el canal se pueda detectar y corregir los errores generados por este; además con la codificación de canal se puede operar con transmisión de baja potencia, realizar transmisiones a largas distancias, mayor tolerancia a la interferencia, transmitir a altas tasas de datos y hacer posible usar antenas pequeñas. Un sistema de codificación de canal además de corregir errores debe ser capaz de corregir otras fallas que se pueden presentar en la degradación del canal tales como: atenuación de la señal debido al medio, el ruido térmico, interferencia inter-símbolo, interferencia del múltiple usuario, la propagación multidireccional, y limitaciones de potencia.

1.2.3.1 Codificador para Control de Errores

[11]El codificador para control de errores se utiliza para añadir redundancia a la información mediante la transformación de la secuencia binaria en una secuencia discreta llamada secuencia codificada, con el fin de que pueda ser transmitida a través de un canal ruidoso y posteriormente, comprobar y corregir los errores que hayan podido ocurrir en la transmisión. Generalmente la secuencia codificada es una secuencia binaria, aunque existen aplicaciones en que la codificación es *M-aria*[12], donde *M* es una potencia positiva de 2.

[13]Actualmente se emplean dos formas de controlar los errores. Una de las técnicas es conocida como solicitud de confirmación ARQ, ya que el receptor comprueba los datos recibidos y si hubo error solicita una retransmisión, de lo contrario retorna una confirmación de recepción correcta. La otra técnica se conoce como corrección de error hacia delante FEC, y es utilizada en los sistemas de comunicación que operan en tiempo real (es decir, el transmisor no almacena los datos que envía como en el caso de voz y vídeo digitales).

---

[10] FUENTE: SACANAMBOY, Maribell; Tesis de Maestría: Diseño e Implementación de los Turbo Codificadores definidos en los estándares de Telecomunicaciones cdma2000 (TIA/EIA 2002.2D) y WCDMA (3GPP TS 25.212 v7.2.0) usando Hardware Reconfigurable; Pág: 3.

[11] FUENTE: LARA, Belén; Codificación de Datos: Nuevas Tecnologías en Comunicaciones Móviles; Pág: 7.

[12] M-ARIA: o M-ario, es una expresión que denota el número de bits utilizados por el alfabeto de señalización (Ejemplo: binaria, terciaria, cuaternaria, etc).

[13] FUENTE: ALVARADO, Raúl; Códigos para Detección y Corrección de Errores en Comunicaciones Digitales; Págs: 51-52.

Por cuestiones de extensión, en este proyecto de titulación solo se abordarán los códigos FEC y se describirá brevemente el funcionamiento de los códigos ARQ. En general, el proceso de codificación en los códigos FEC es una operación en la cual a un grupo de bits de datos correspondientes al mensaje que se desea transmitir, se le agrega un grupo de bits conocidos como de paridad para fines de detección y corrección de errores. Dentro de estos esquemas de codificación se encuentran los ***códigos de bloque***, los ***códigos convolucionales*** y los ***turbo códigos***. Los temas concernientes a códigos de bloque y convolucionales se abordan en el segundo capítulo del presente proyecto de titulación, mientras que los turbo códigos serán estudiados e implementados en el tercer y cuarto capítulo.

1.2.3.2 Modulador

[14]El modulador convierte los símbolos del canal discreto en formas de onda que se transmiten a través del canal o medio de transmisión físico, el cual es análogo. Para señalización binaria las técnicas de modulación más usadas son BPSK, DPSK y FSK; mientras que para señalización *M-aria*, los esquemas de modulación de mayor uso son los *M*PSK, *M*DPSK y *M*FSK.

El Espacio de Código 1.1 muestra la generación de una señal binaria en Matlab, que luego va a ser comprimida (agrupada) y modulada, con el fin de comprender de manera visual como se realizan estos procesos[15]. La señal binaria (valores aleatorios de 0s y 1s) se la crea mediante la función ***randint***, y estos valores van a ser guardados en un archivo de texto ***x.txt***, para poder manipular en posteriores simulaciones la misma secuencia de información evitando así que los datos cambien y surjan problemas de seguimiento en la evolución de dicha información de entrada. Para este caso se ha modulado con 16-QAM con el objeto de visualizar de mejor manera las constelaciones, pero esta se puede modificar a modulación PSK (cambiando el objeto ***modem.qammod*** por ***modem.pskmod***), siendo esta última la modulación que será usada en el caso del Turbo

[14] FUENTE: SCHLEGEL, Christian B.; PÉREZ Lance C.; Trellis and Turbo Coding; Pág: 4.
[15] FUENTE: THE MATHWORKS; Communications Toolbox™ 4: User's Guide; Sección: 1, Págs: 1-14.

Codificador. [16]Con la transmisión por desplazamiento de fase binaria (BPSK - *Binary Phase Shift Keying*), son posibles dos fases de salida para una sola frecuencia de portadora. Una fase de salida representa un 1 lógico y la otra un 0 lógico. Conforme la señal digital de entrada cambia de estado, la fase de la portadora de salida se desplaza entre dos ángulos que están 180° fuera de fase.
La Figura 1.2 muestra la tabla de verdad, diagrama fasorial, y diagrama de constelación para un modulador de BPSK. Un diagrama de constelación que, a veces, se denomina diagrama de espacio de estado de señal, es similar a un diagrama fasorial, excepto que el fasor completo no está dibujado. En un diagrama de constelación, sólo se muestran las posiciones relativas de los picos de los fasores.

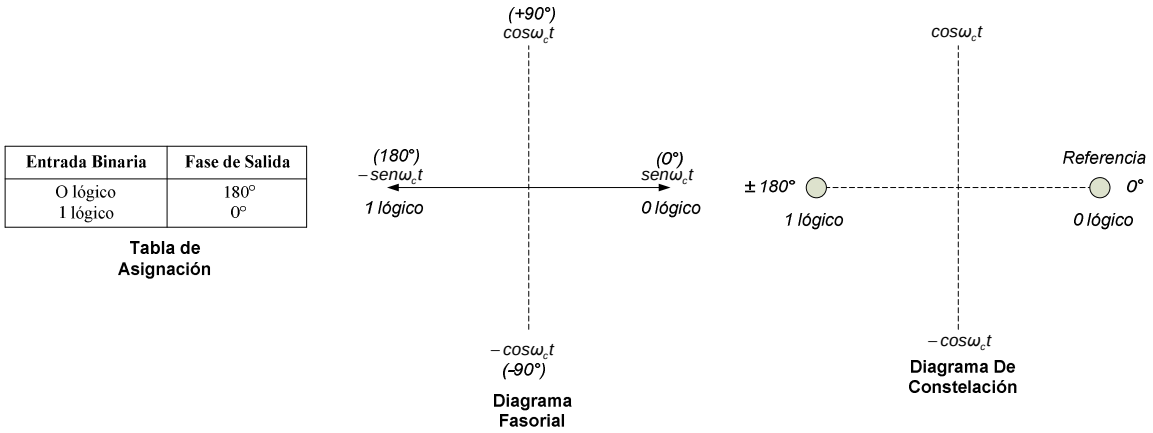


Figura 1.2. Modulador BPSK. [17]

La sintaxis detallada de las funciones de Matlab utilizadas, en esta y en las simulaciones posteriores se puntualiza en el Anexo A.

[16] FUENTE: TOMASI, Wayne; Sistemas De Comunicaciones Electrónicas; Pág: 464.
[17] FUENTE: TOMASI, Wayne; Sistemas De Comunicaciones Electrónicas; Pág: 465.

```
clear all, clc, close all
M = 16; % Número de niveles o estados de la constelación:(M=2: BPSK ó 2-PSK),(M=4: Q-PSK
ó 4-PSK),(M=16: 16-QAM),(M=64: 64-QAM),etc
n_bits_modulacion = log2(M); % Número de bits por símbolo
Long_Trama = 40000; % Longitud de la trama de bits a transmitir
x = randint(1,Long_Trama); % Generación de los bits de información con valores (0/1)
save x.txt x -ascii % Guarda la señal binaria (0/1) en un archivo de texto para poder
manipular en posteriores simulaciones la misma secuencia de información
subplot(3,1,1); stem(x(1:40),'filled');
title('BITS ALEATORIOS');
xlabel('Índice de los Bits'); ylabel('Valor Binario');

x_deci = bi2de(reshape(x,n_bits_modulacion,length(x)/n_bits_modulacion).','left-msb'); %
Cambio de binario a decimal (modem recibe por defecto valores decimales) y Agrupamiento
de bits en grupos según el tipo de modulación.
subplot(3,1,2); stem(x_deci(1:10))
title('SÍMBOLOS ALEATORIOS');
xlabel('Índice de los Bits'); ylabel('Valor Decimal');

y = modulate(modem.qammod(M),x_deci); % Modulación
subplot(3,1,3); stem(y(1:10))
title('SÍMBOLOS MODULADOS');
xlabel('Índice de los Bits'); ylabel('Valor Decimal');
```

Espacio de Código 1.1. Ejemplo de Compresión de Datos y Modulación.

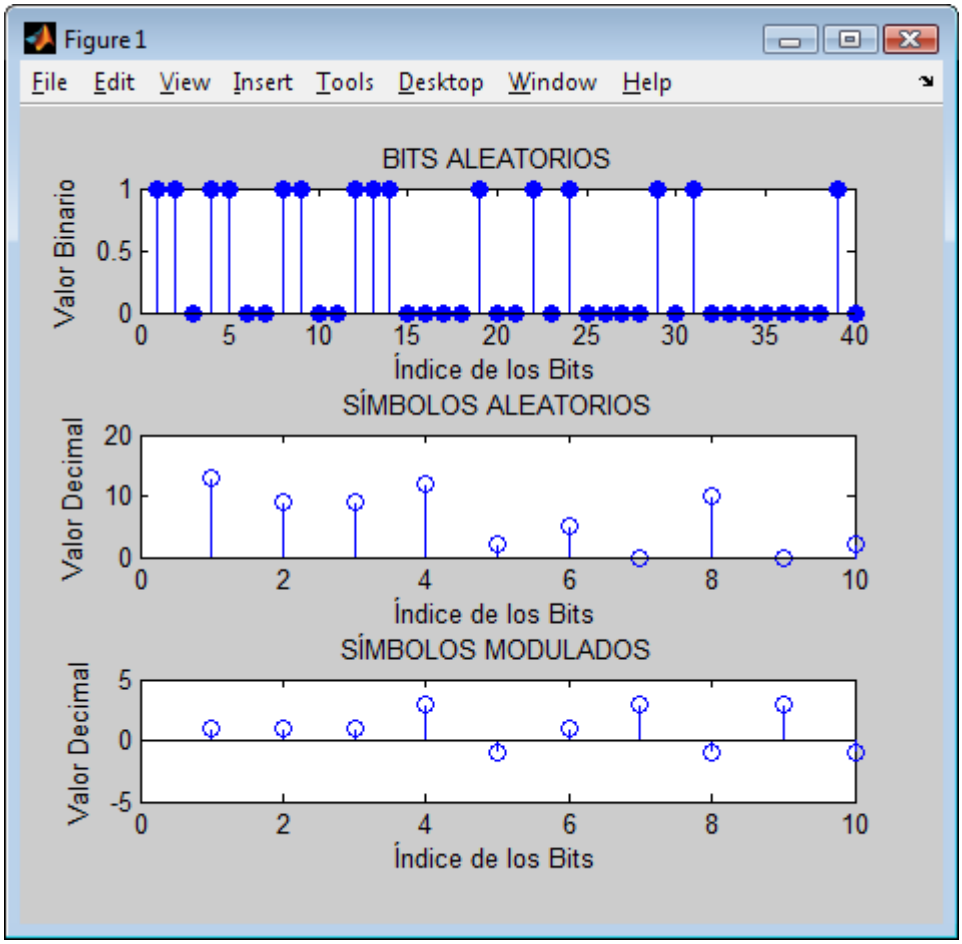


Figura 1.3. Ejemplo de Compresión de Datos y Modulación.

3.3.4 CANAL DE COMUNICACIONES

Para enviar la información desde un punto a otro, la señal transmitida debe viajar a través de un medio para alcanzar al receptor. Este camino desde el transmisor al receptor se denomina canal. Ejemplos de canales incluyen un alambre de cobre, cable de fibra óptica, o el espacio libre. Las características relevantes de un canal incluyen su retardo de propagación y cualquier distorsión que puede agregarse a la señal transmitida.

[18]En el diseño de un sistema de comunicaciones para la transmisión de información a través de canales físicos, es conveniente construir modelos matemáticos que reflejen las características más importantes del medio de transmisión. Por esta razón se hace énfasis en el estudio y descripción de los modelos de canales que son frecuentemente usados para caracterizar muchos de los canales físicos que se encuentran en la práctica. Particularmente, si el interés es la simulación el programa debe incluir un modelo predefinido, el cual es común en los escenarios de comunicaciones fijas y móviles. Se destacan en este aspecto, el modelo de fading (desvanecimientos), modelo con ruido Gaussiano (AWGN) y canal BSC. En cada uno de ellos, se pueden modificar los parámetros para satisfacer necesidades específicas.

En el canal inalámbrico se tienen dos tipos de ruidos: Aditivo y Multiplicativo. Los mismos que modifican la respuesta del canal.

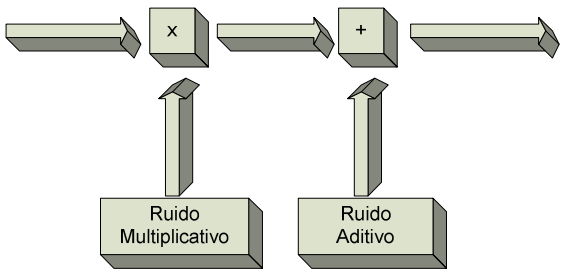


Figura 1.4. Tipos de Ruido en un Canal Inalámbrico.

1.2.4.1 Canales con Ruido Multiplicativo

Dentro del ruido multiplicativo, se tienen a los siguientes factores que alteran a la señal:

[18] FUENTE: PROAKIS, John G.; Digital Communications; Pág: 11.

- Pérdidas por Trayectoria: Path Loss
- Shadowing
- Multipath

[19]Las FDPs[20] que se obtengan, dependerán básicamente si hay línea de vista (LOS - Rice/Rician) o si no hay línea de vista (NLOS - Rayleigh).

Uno de los modelos frecuentemente usados para un canal es aquel que simula el desvanecimiento de una señal. Los canales *fading* se producen típicamente en un proceso de multitrayectoria o de dispersión (*scattering*), y se caracterizan por una amplitud y fase de la señal variando en forma aleatoria. El proceso de *fading* es considerado no selectivo en frecuencia cuando el ancho de banda señalado es mucho más pequeño que el ancho de banda de coherencia del canal, es decir cuando las componentes espectrales en el ancho de banda de la señal son igualmente afectadas por el canal. Cuando la señal recibida comprende solamente componentes dispersadas de-correlatadas, el proceso de *fading* se designa como tipo *Rayleigh*. En la Figura 1.5 se ilustra el efecto de un desvanecimiento Rayleigh sobre la potencia de la señal recibida utilizando la sub-función *rayleighchan* que posee el Matlab para este tipo de canal.

```
clear all
x = ones(2000,1); % Creación de un vector de señal
y = rayleighchan(1/10000,100); % Creación del canal Rayleigh con una señal muestreada a
10000Hz y un Efecto Doppler Máximo de 100Hz
z = filter(y,x); % Pasa la señal compleja a través del canal Rayleigh
plot(20*log10(abs(z))),grid,title('CANAL TIPO RAYLEIGH'),xlabel('Número de Muestras
(Seg)'),ylabel('Potencia de la Señal con Desvanecimiento(dBW)') % Gráfico de la Potencia
de la Señal con Desvanecimeinto Vs. Número de muestras.
```

Espacio de Código 1.2. Ejemplo de un Desvanecimiento Rayleigh

[19] FUENTE: ARÁUZ, Julio; Discrete Rayleigh Fading Channel Modeling. Universidad de Pittsburgh; Págs: 3-4.
[20] FDP: La función de densidad de probabilidad (FDP) se utiliza en estadística con el propósito de conocer cómo se distribuyen las probabilidades de un suceso o evento, en relación al resultado del suceso.

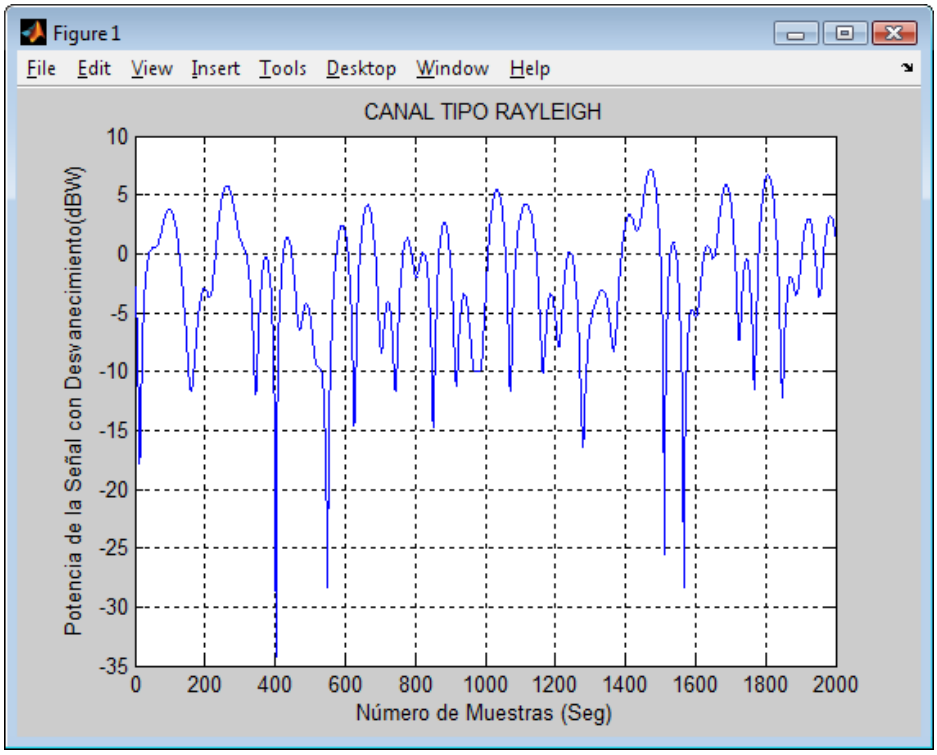


Figura 1.5. Ejemplo de un Desvanecimiento Rayleigh

Cuando una componente muy fuerte está presente en la forma de onda de la señal recibida, tal como la de un camino directo o causado por un reflector fijo en el medio, el proceso de *fading* se designa como *Rician*. Para modelar un canal con desvanecimiento, la señal de entrada se multiplica típicamente por una variable aleatoria compleja con una distribución de amplitud del tipo *Rayleigh* o *Rician*, y con una fase uniforme. La Figura 1.6 muestra un canal tipo Rician simulado en Matlab por medio de la sub-función *ricianchan*.

```
clear all
x = ones(2000,1); % Creación de un vector de señal
y = ricianchan(1/10000,100,10); % Creación del canal Rician con una señal muestreada a
10000Hz, un Efecto Doppler Máximo de 100Hz y factor de Rician de 10
z = filter(y,x); % Pasa la señal compleja a través del canal Rician
plot(20*log10(abs(z))),grid,title('CANAL TIPO RICIAN'),xlabel('Número de Muestras
(Seg)'),ylabel('Potencia de la Señal con Desvanecimiento(dBW)') % Gráfico de la Potencia
de la Señal con Desvanecimeinto Vs. Número de muestras.
```

Espacio de Código 1.3. Ejemplo de un Desvanecimiento Rician

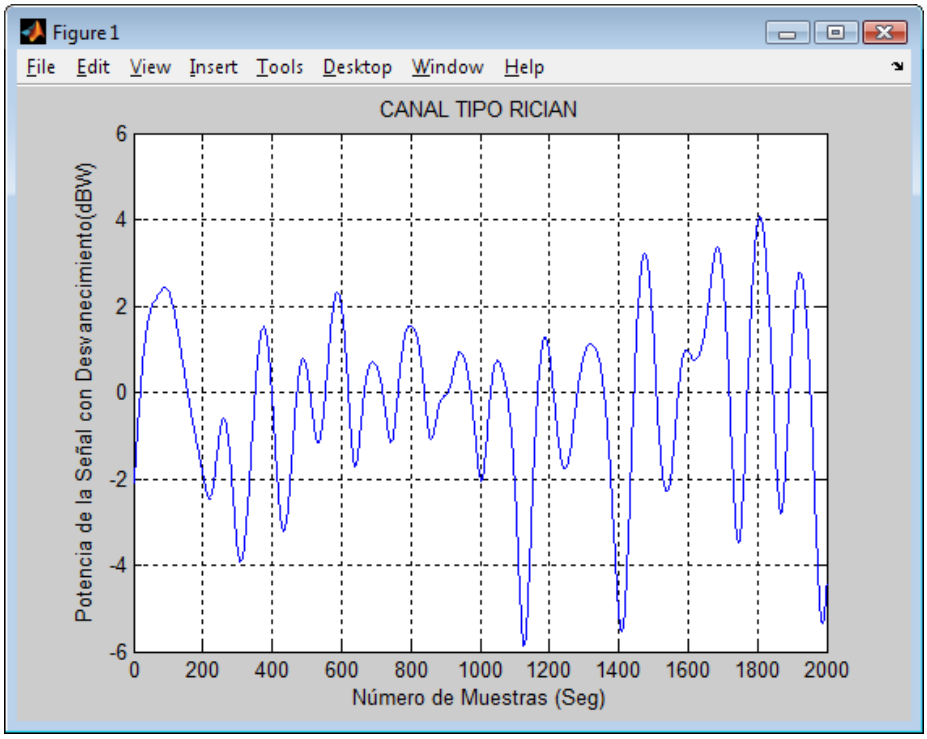


Figura 1.6. Ejemplo de un Desvanecimiento Rician

Dentro de los alcances planteados en este proyecto de titulación, no se considera la implementación de un canal multiplicativo en el esquema de turbo codificación, pero se lo ha mencionado y simulado con el objeto de dar una idea didáctica de cómo se implementa este tipo de canal de comunicaciones.

1.2.4.2 Canales con Ruido Aditivo

Se considera ruido aditivo al cual la potencia del ruido se suma a la potencia de la señal de información. Algunos ejemplos de ruido aditivo son el:

- Térmico
- De disparo
- Propio del receptor
- Interferencia de otros transmisores

[21]Es común representar al ruido Gaussiano de banda ancha con el modelo de *Ruido Blanco Gaussiano*, el cual consiste en un proceso aleatorio en el que cada muestra es una variable aleatoria Gaussiana de media cero y varianza σ^2. De esta forma, la Ecuación 1.1 describe la salida del canal AWGN *y(t)*:

$$y(t) = x(t) + n \qquad \textbf{(Ecuación 1.1)}$$

Este modelo de canal es muy usado en análisis de sistemas de comunicaciones y la potencia del ruido posee una densidad espectral uniforme (ruido blanco, *n*), el cual se agrega a la señal original *x(t)*. Desde el punto de vista estadístico, se considera al ruido como una variable aleatoria y por tanto se describe a partir de su [22]función de densidad de probabilidad condicional de la salida *y,* dada la entrada *x*, como se muestra en la Ecuación 1.2.

$$p\langle y | x \rangle = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-(y-x)^2/2\sigma^2} \qquad \textbf{(Ecuación 1.2)}$$

Esto puede no ser siempre muy realista, pero simplifica bastante la matemática asociada con la estimación del rendimiento de un sistema de comunicación. De hecho, la mayoría de las curvas de tasa de error (BER) se generan con la asunción de que el canal es con ruido gaussiano. La Figura 1.7 muestra el uso de dicho canal con una señal sinusoidal, tomando en cuenta que el programa computacional Matlab posee una sub-función ***awgn***, que simula este tipo de canal, tomando como entrada la *S/N* (o *SNR* en inglés) en dB, donde:

- [23]*S* representa la potencia media de transmisión y está dada por la Ecuación 1.3.

$$S = E_b R \qquad \textbf{(Ecuación 1.3)}$$

[21] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Pág: 16.
[22] FUENTE: MOON, Todd K.; Error Correction Coding: Mathematical Methods and Algorithms; Pág: 16.
[23] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Pág: 17.

Donde, E_b representa la energía de bit, y R la tasa de transferencia de información del sistema de comunicación.

- N representa la potencia media del ruido presente en el canal dentro del ancho de banda considerado, como se muestra en la Ecuación 1.4:

$$N = N_o W \qquad \textbf{(Ecuación 1.4)}$$

Donde, N_o representa la densidad espectral de potencia unilateral del ruido Blanco Gaussiano, equivalente a la densidad espectral de potencia bilateral de nivel $N_o/2$ en todo el rango de frecuencias $-\infty \leq f \leq \infty$ y W es el ancho de banda.

Se debe tener en cuenta que el número de bits presentes para representar una señal modulada con N estados está dado por $n = log_2 N$; por lo tanto:

$$\frac{S}{N} = \left(\frac{E_b}{N_o}\right)\left(\frac{R}{W}\right) = \left(\frac{E_b}{N_o}\right)\left(\frac{2nW}{W}\right) = \left(\frac{E_b}{N_o}\right)(2n) \qquad \textbf{(Ecuación 1.5)}$$

$$\left.\frac{S}{N}\right|_{dB} = \left.\frac{E_b}{N_o}\right|_{dB} + 10\,log(2) + 10\,log(n) \qquad \textbf{(Ecuación 1.6)}$$

$$\left.\frac{S}{N}\right|_{dB} = \left.\frac{E_b}{N_o}\right|_{dB} + 3 + 10\,log(n) \qquad \textbf{(Ecuación 1.7)}$$

Otra de las expresiones que frecuentemente es usada para relacionar la S/N y la E_b/N_o es la que denota la Ecuación 1.8:

$$\left.\frac{S}{N}\right|_{dB} = \left.\frac{E_b}{N_o}\right|_{dB} + 10\,log(n) - 10\,log(N_{muestras}) \qquad \textbf{(Ecuación 1.8)}$$

```
clear all
x = linspace(0,001*(2*pi),200); % Generación de un vector entre 0,001*(2*pi) y 200
y = 2*sin(x).*cos(x); % Generación de la Señal Original (Analógica)
z = awgn(y,15); % Generación de la Señal AWGN con SNR de 15dB
plot(x,y,x,z),grid,axis([0 6 -2 2])  % Grafica ambas señales
title('CANAL AWGN') % Título del Gráfico
legend('Señal Original','Señal con AWGN'); % Leyenda para mostrar los detalles de las
dos señales
```

Espacio de Código 1.4. Ejemplo de Ruido Gaussiano Aditivo agregado a una Señal Analógica.

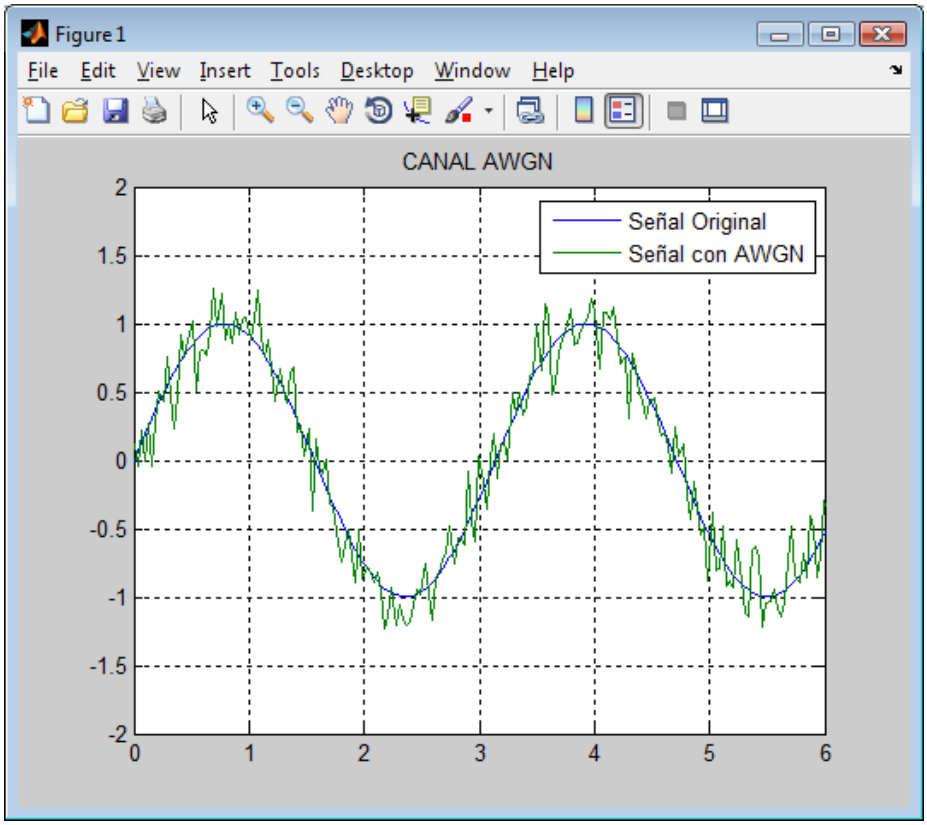


Figura 1.7. Ejemplo de Ruido Gaussiano Aditivo agregado a una Señal Analógica.

En el Espacio de Código 1.4, la función con sintaxis *awgn(y,15);* hace referencia a una señal analógica *y*, a la que se le agrega ruido WGN con *S/N* de *15* dB.

Para el caso de las simulaciones que se llevaran a cabo en capítulos posteriores del presente proyecto de titulación, para la simulación del Turbo Código no se usará la función directa del *toolbox* de comunicaciones de Matlab sino que más bien se ha tratado de diseñar un canal AWGN con características y parámetros de entrada similares a los que usa este programa, basándose en los principios que rigen a este tipo de ruido. Por esto, a continuación se presenta un detalle de este tipo de canal y los principios que lo gobiernan:

[24]Tomando en cuenta la *SNR* en dB, como parámetro de entrada esta puede ser transformada a un valor decimal y ponerla en términos de la E_b/N_o, como muestra la Ecuación 1.9:

$$\frac{E_b}{N_o} = 10^{\left(SNRdB/10\right)} \qquad \textbf{(Ecuación 1.9)}$$

La energía por bit E_b, es la energía total de la señal, dividida para el número de bits contenidos en la señal. Por lo que se puede expresar al E_b como la potencia media de la señal multiplicada por la duración de un bit, a través de la Ecuación 1.10:

$$E_b = \frac{1}{N_{\text{muestras}} . f_{bit}} \sum_{n=1}^{N} x^2(n) \qquad \textbf{(Ecuación 1.10)}$$

Donde, $N_{muestras}$ es el número total de muestras en la señal, y f_{bit} es la frecuencia de bit en [Hz]. Con *S/N* y E_b conocidas, se puede calcular la densidad espectral de potencia del ruido N_o , a partir de la Ecuación 1.11:

$$N_o = \frac{E_b}{E_b/N_o} \qquad \textbf{(Ecuación 1.11)}$$

La densidad de potencia espectral del ruido unilateral, N_o, muestra la cantidad de potencia de ruido presente en un 1 [Hz] del ancho de banda de la señal. Para encontrar la varianza, o potencia media del ruido, se debe conocer el ancho de banda del ruido. Para una señal real *x*, muestreada a f_s [Hz], el ancho de banda del ruido será la mitad de la tasa de muestreo. Por lo tanto, la varianza se encuentra multiplicando la densidad de potencia espectral del ruido por el ancho de banda del ruido, como en la Ecuación 1.12:

$$\sigma^2 = \frac{N_o f_s}{2} \quad \rightarrow \quad \sigma = \sqrt{\frac{N_o f_s}{2}} \qquad \textbf{(Ecuación 1.12)}$$

[24] FUENTE: GILLEY, James E.; Bit-Error-Rate Simulation Using Matlab; Págs: 1-6.

Donde σ^2 es la varianza de ruido en [W], y N_o es la densidad espectral de potencia unilateral del ruido en [W/Hz]. Dado que el ruido tiene una media de cero, su potencia y su varianza son idénticos. La función de Matlab *randn* genera números aleatorios distribuidos normalmente con una media de cero y una varianza de uno. Por ello, se debe escalar la salida *randn* a σ (desviación típica), esto se lo obtiene multiplicando la salida de *randn* por σ , para poder generar *n,* es decir el ruido, expresado en la Ecuación 1.13:

$$n = (\sigma)(randn) \qquad \textbf{(Ecuación 1.13)}$$

Para obtener la salida del canal AWGN se agrega a la señal original el ruido *n*. De esta manera si se le agrega el ruido WGN a la señal sinusoidal anterior, se puede observar que se obtiene un resultado similar al que propone Matlab con su función *awgn*. El Espacio de Código 1.5 muestra la implementación del Canal AWGN para la misma señal analógica.

```
x = linspace(0,001*(2*pi),200); % Generación de un vector entre 0,001*(2*pi) y 200
y = 2*sin(x).*cos(x); % Generación de la Señal Original (Analógica)
fs = 1/pi; % Frecuancia de Muestreo (señal con período pi)
SNRdB = 15; % Relación Señal a Ruido en dB
EbNo = 10^(SNRdB/10); % EbNo en decimal
n_muestras = length(y); % Número de muestras por simbolo
fb = 0.1; % Frecuencia de bit (Velocidad de transmisión)
Eb = sum(y.^2)/(n_muestras*fb); % Cálculo de la Energía de Bit
No = Eb/EbNo; % Cálculo de la densidad de potencia de ruido
sigma = sqrt(No*fs/2); % Cálculo de la varianza
n = sigma*randn(1,length(y)); % Ruído WGN, se usa randn para generar bits aleatorios
normalizados
z = y + n; % Generación de la Señal AWGN
plot(x,y,x,z),grid,axis([0 6 -2 2])  % Grafica ambas señales
title('CANAL AWGN') % Título del Gráfico
legend('Señal Original','Señal con AWGN'); % Leyenda para mostrar los detalles de las
dos señales
```

Espacio de Código 1.5. Ejemplo de Ruido Gaussiano Aditivo agregado a una Señal

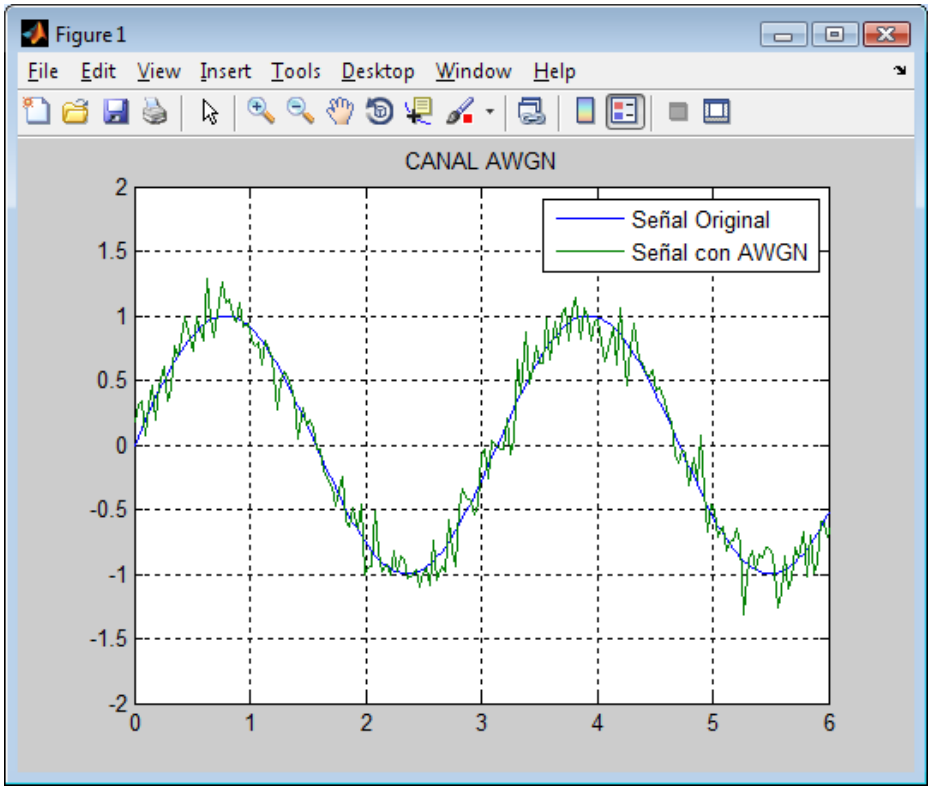


Figura 1.8. Ejemplo de Ruido Gaussiano Aditivo agregado a una Señal.

Comparando la dispersión del canal AWGN del Espacio de Código 1.4 y del Espacio de Código 1.5 mediante la desviación típica que se desprende de la varianza, se puede concluir que ambas implementaciones denotan similares resultados, ya que la función *awgn* tiene una desviación típica de 0.17783 y en el canal AWGN del Espacio de Código 1.5 la desviación típica es de 0.15824.

[25]Para el caso de los Turbo Códigos es necesario que la varianza del canal AWGN, este en términos de los parámetros que va a utilizar el sistema, por lo que se ha particularizado el mismo para una señal digital, tomando en cuenta a la tasa de código R y la relación E_b/N_o, como se muestra en la Ecuación 1.14.

$$\sigma^2 = \frac{N_o}{2} = \frac{1}{2(R)\left(E_b/N_o\right)} \quad \rightarrow \quad \sigma = \sqrt{\frac{1}{2(R)\left(E_b/N_o\right)}} \qquad \textbf{(Ecuación 1.14)}$$

[25] FUENTE: LANGTON, Charan; Intuitive Guide to Principles of Communications: Turbo Coding and MAP Decoding, Parte 1; Pág: 19.

Es decir, que el ruido para este caso estaría dado por la Ecuación 1.13 y para obtener la salida del canal AWGN, se agrega el ruido a la señal modulada *y*:

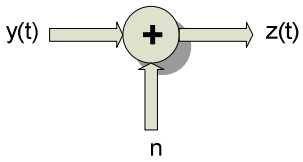
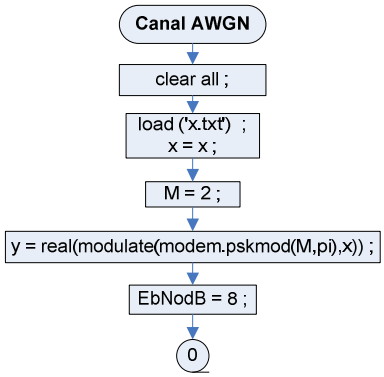


Figura 1.9. Esquema de una Señal con Ruido Aditivo.

El Espacio de Código 1.6 muestra la implementación del canal AWGN para el caso de una señal digital, cabe mencionar que a partir de este espacio de código y en las sub-siguientes simulaciones que requieran alguna secuencia de información binaria de entrada, esta no va a ser creada por medio de una función *randint* sino que se la va a cargar mediante la función *load* a través del archivo de texto *x.txt* generado en el Espacio de Código 1.1.

```
clear all
load ('x.txt'); % Carga el archivo de texto con los bits generados
x = x;   % Conserva el mismo nombre de la variable
% Si se desea generar una nueva secuencia de bits de información se debe:
% comentar las dos lineas anteriores de programa e introducir: x = randint(1,40);
M = 2; % Número de niveles o estados de la señal modulada
y = real(modulate(modem.pskmod(M,pi),x)); % Bits Modulados (+1/-1)
EbNodB = 8; % Variación de EbNo (en decibelios)
tasa = 1/2; % Tasa de Código
for nEN = 1:length(EbNodB)
    EbNo(nEN) = 10^(EbNodB(nEN)/10); % Convierte Eb/No de db a números decimales
end
sigma = 1/sqrt(2*tasa*EbNo); % Cálculo de la Varianza
n = sigma*randn(1,length(y)); % Ruído WGN, se usa randn para generar bits normailzados
z = y + n; % Generación de la Señal AWGN
subplot(2,1,1); stem(y(1:10),'filled');
title('SEÑAL ORIGINAL') % Título del Gráfico
xlabel('Índice de los Bits'); ylabel('Valor Modulado');
subplot(2,1,2); stem(z(1:10),'filled');
title('CANAL AWGN') % Título del Gráfico
xlabel('Índice de los Bits'); ylabel('Valor Modulado con Ruido');
```

Espacio de Código 1.6. Ruido Gaussiano Aditivo agregado a una Señal Digital.



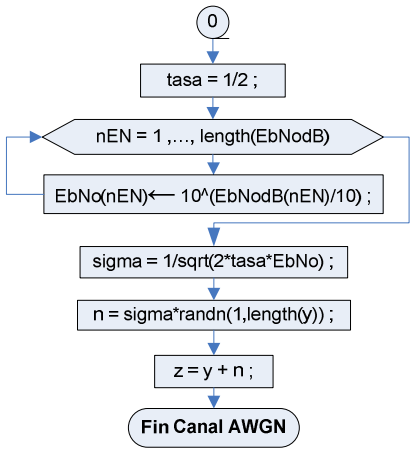


Figura 1.10. Diagrama de Flujo del Ruido Gaussiano Aditivo agregado a una Señal Digital.

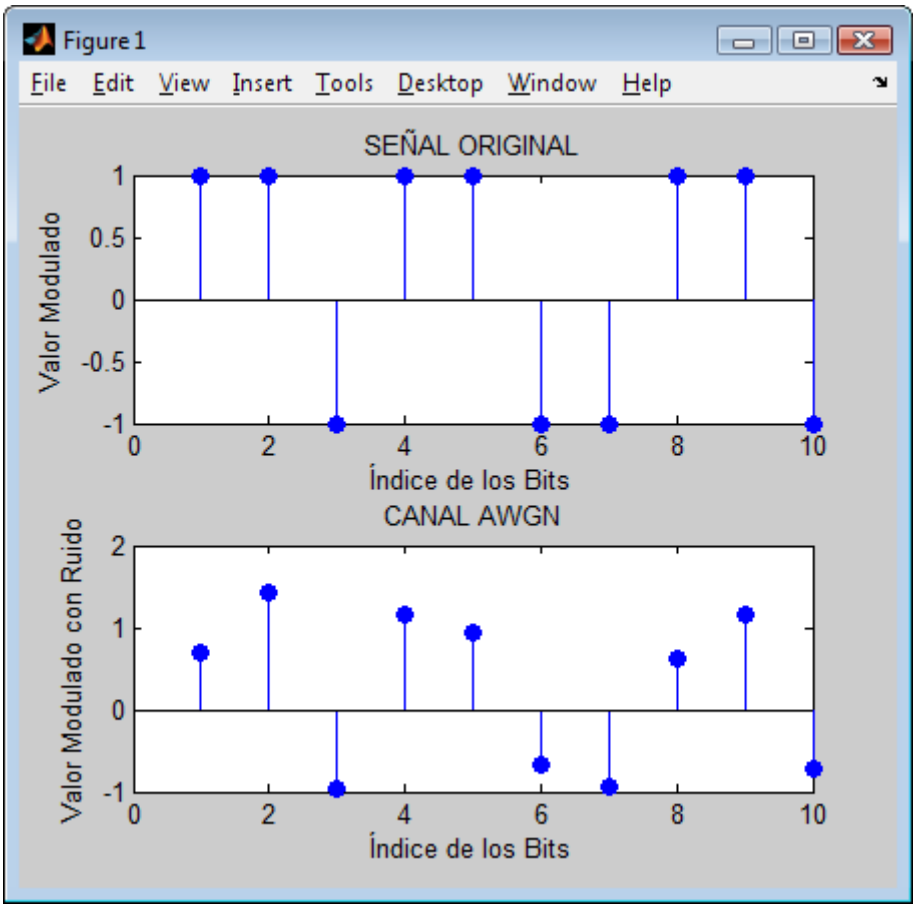


Figura 1.11. Ruido Gaussiano Aditivo agregado a una Señal Digital.

1.2.4.3 Canal Binario Simétrico (BSC)

Este tipo de canal aún cuando no pertenece al grupo de los aditivos ni multiplicativos, es generalmente usado cuando se simula la transmisión de datos digitales. Este modelo matemático de canal asume que todos los errores en los

bits son igualmente probables y que los errores ocurren con una probabilidad fija que se especifica por el usuario. Como se muestra en la Figura 1.12, si se transmite un “1” se tiene una probabilidad p de recibirse incorrectamente como un “0”, y una probabilidad [1-p] de ser recibido correctamente, donde p es típicamente un número pequeño que fluctúa su valor entre 0 y 1. Este modelo de canal es útil cuando no se está interesado en el rendimiento abajo del nivel de bit; como por ejemplo evaluar una codificación para una tasa de error de canal pre-especificada.

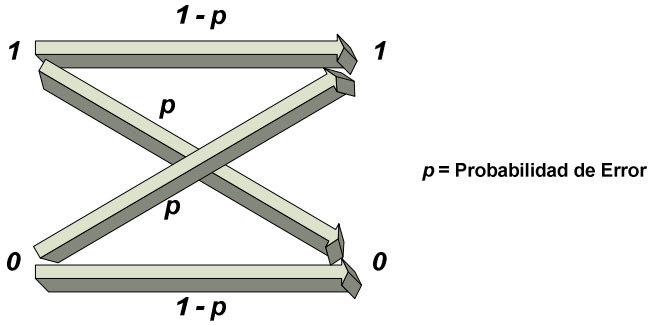


Figura 1.12. El Canal Binario Simétrico (BSC). [26]

La Figura 1.13 muestra una señal simulada en Matlab y a la misma señal pasando a través de un canal BSC con el 100% de probabilidad de que se cometa un error:

```
clear all
load ('x.txt'); % Carga el archivo de texto con los bits generados
x = x;   % Conserva el mismo nombre de la variable
% Si se desea generar una nueva secuencia de bits de información se debe:
% comentar las dos lineas anteriores de programa e introducir: x = randint(1,10);
p  = 1; % Inserción de errores de bit, en los bits aleatorios, con probabilidad de error
del 100%
y = bsc(x(1:10),p); % Pasa la Señal con valores de 0 y 1 a traves del canal BSC y con
probabilidad p
plot(x,' - b'),grid,axis([1 10 -0.5 1.5]); hold on;  % Grafica Señal Original
plot(y,' - g'),axis([1 10 -0.5 1.5]); hold off  % Grafica Señal a Través del Canal BSC
title('CANAL BSC') % Título del Gráfico
legend('Señal Original','Señal a Través del Canal BSC'); % Leyenda para mostrar los
detalles de las dos señales
```

Espacio de Código 1.7. Ejemplo de un Canal BSC con Probabilidad de Error del 100%

[26] FUENTE: LIN, Shu; COSTELLO, Daniel; Error Control Coding: Fundamentals and Applications; Pág: 7.

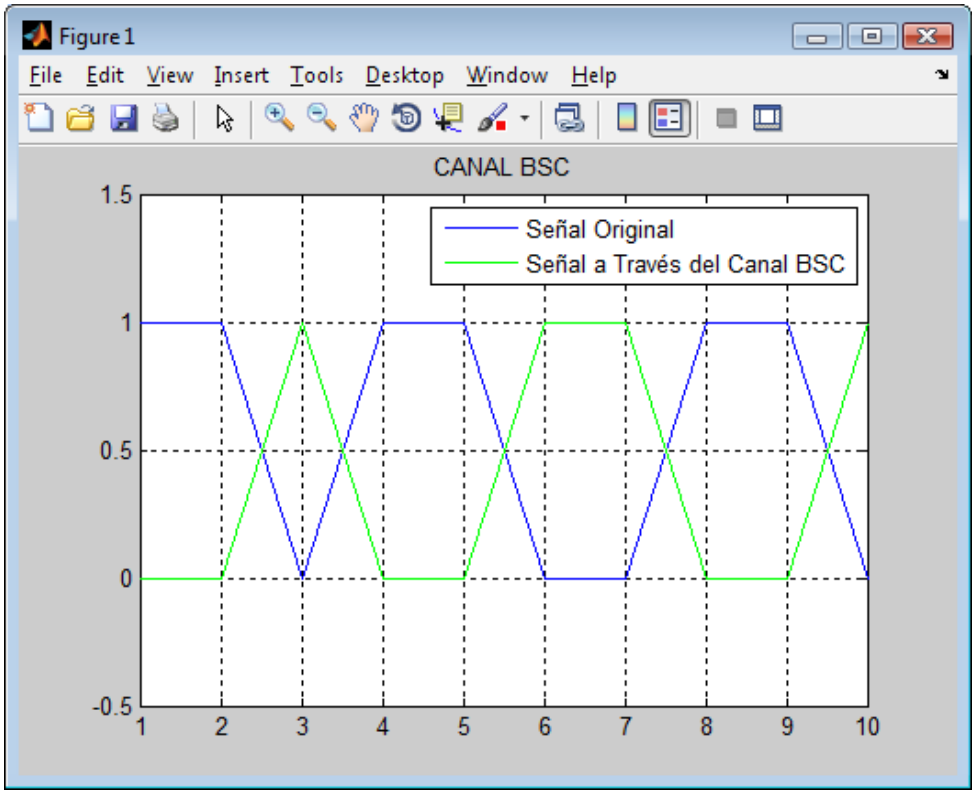


Figura 1.13. Ejemplo de un Canal BSC con Probabilidad de Error del 100%

3.3.5 DECODIFICADOR DE CANAL

[27]El decodificador de canal explota la redundancia introducida por el codificador de canal para corregir los errores introducidos. Como sugiere la Figura 1.1, demodulación, ecualización y decodificación deben ser combinados.

1.2.5.1 Demodulador

[28]El demodulador reconvierte las formas de onda nuevamente en una secuencia discreta de símbolos recibidos. [29]Esto normalmente implica muchas funciones, como el filtrado, demodulación, sincronización de portadora, la estimación símbolo de la sincronización, sincronización del marco, y filtrado emparejado, seguido por un paso de la detección en el que se hacen las decisiones sobre los símbolos transmitidos.

[27] FUENTE: MOON, Todd K.; Error Correction Coding: Mathematical Methods and Algorithms; Pág: 9.
[28] FUENTE: SCHLEGEL, Christian B.; PÉREZ Lance C.; Trellis and Turbo Coding; Pág: 3.
[29] FUENTE: MOON, Todd K.; Error Correction Coding: Mathematical Methods and Algorithms; Pág: 9.

1.2.5.1.1 Decisiones Duras y Decisiones Suaves

[30]La demodulación con decisiones duras y decisiones suaves se refieren al tipo de cuantización utilizada en la recepción de los bits. La demodulación con decisión dura utiliza 1 bit de cuantización al recibir los valores del canal. La demodulación con decisión suave utiliza cuantización multi-bit al recibir los valores del canal. Para un modelo ideal de demodulación con decisión suave (infinitos bits de cuantización), los valores recibidos por el canal son directamente usados en el decodificador de canal. La Figura 1.14 muestra la demodulación con decisiones duras y suaves.

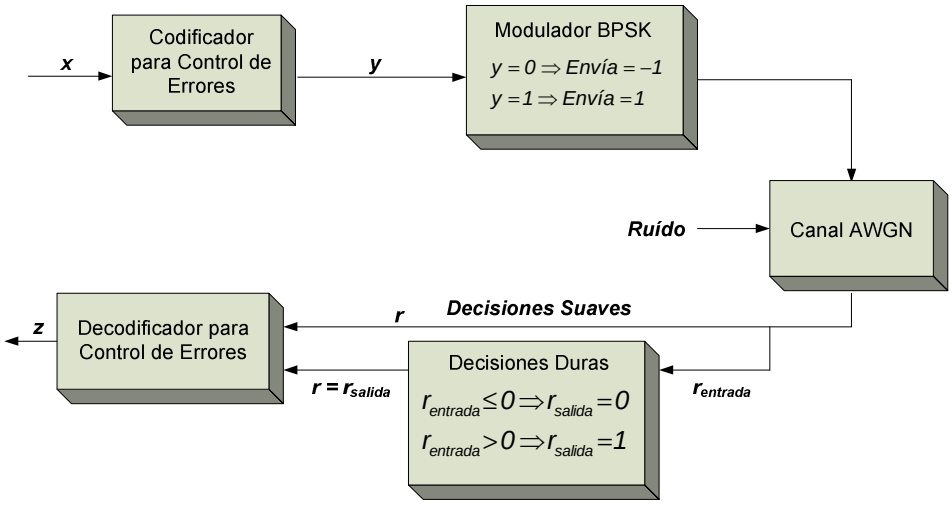


Figura 1.14. Demodulación con Decisiones Duras y Suaves. [31]

La mayoría de los sistemas digitales de comunicación codificados emplean un esquema de codificación binario con demodulación de decisiones duras debido a la simplicidad de la implementación en comparación con los sistemas no binarios. Sin embargo, la demodulación de decisiones suaves generalmente ofrece un mejor desempeño del sistema que el que ofrece un esquema de demodulación de decisiones duras.

El Espacio de Código 1.8 muestra a la señal modulada recibida, que se generó en el Espacio de Código 1.1 en una gráfica de dispersión, sin pasarla por ningún canal, desagrupada y demodulada por medio de la función ***demodulate***.

[30] FUENTE: HUANG, Fu-hua; Master's Thesis: Evaluation of Soft Output Decoding for Turbo Codes.; Pág: 10.
[31] FUENTE: HUANG, Fu-hua; Master's Thesis: Evaluation of Soft Output Decoding for Turbo Codes.; Pág: 10.

```
n_muestras = 1; % Número de muestras por simbolo
y_canal = y; % Canal es el mismo que el valor modulado (no existe alteración señal
modulada)
h = scatterplot(y_canal(1:n_muestras*5e3),n_muestras,0,'go'); hold on; % Gráfica de
Dispersión Con Canal
scatterplot(y(1:n_muestras*5e3),n_muestras,0,'k*',h); grid on; % Gráfica de Dispersión
Sin Canal
title('SEÑAL RECIBIDA'); % Título del Gráfico
legend('Señal Recibida','Constelación de la Señal'); % Leyenda para mostrar los detalles
de las dos señales
xlabel('En Fase'); ylabel('Cuadratura');% Título del Eje de las X e Y
figure; subplot(3,1,1); stem(y_canal(1:10));
title('BITS DEMODULADOS');
xlabel('Índice de los Bits'); ylabel('Valor Decimal');

z_deci = demodulate(modem.qamdemod(M),y_canal); % Demodulación
subplot(3,1,2); stem(z_deci(1:10))
title('SÍMBOLOS RECIBIDOS');
xlabel('Índice de los Bits'); ylabel('Valor Decimal');

z = de2bi(z_deci,'left-msb'); % Cambio de decimal a binario
z = reshape(z.',prod(size(z)),1); % Desagrupamiento de bits
subplot(3,1,3); stem(x(1:40),'filled');
title('BITS RECIBIDOS');
xlabel('Índice de los Bits'); ylabel('Valor Binario');
```

Espacio de Código 1.8. Ejemplo de Decompresión de Datos y Demodulación Sin Canal.

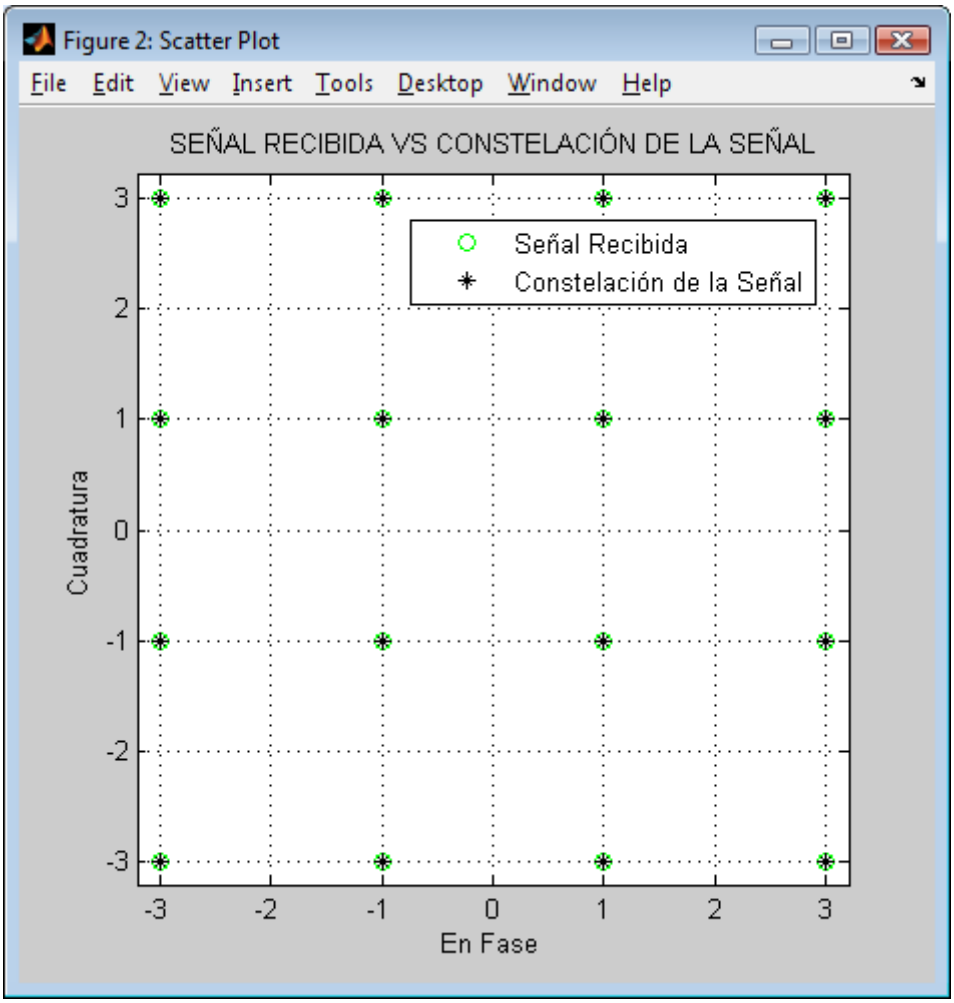


Figura 1.15. Diagrama de Señal Recibida Sin Canal y Constelación de la Señal.

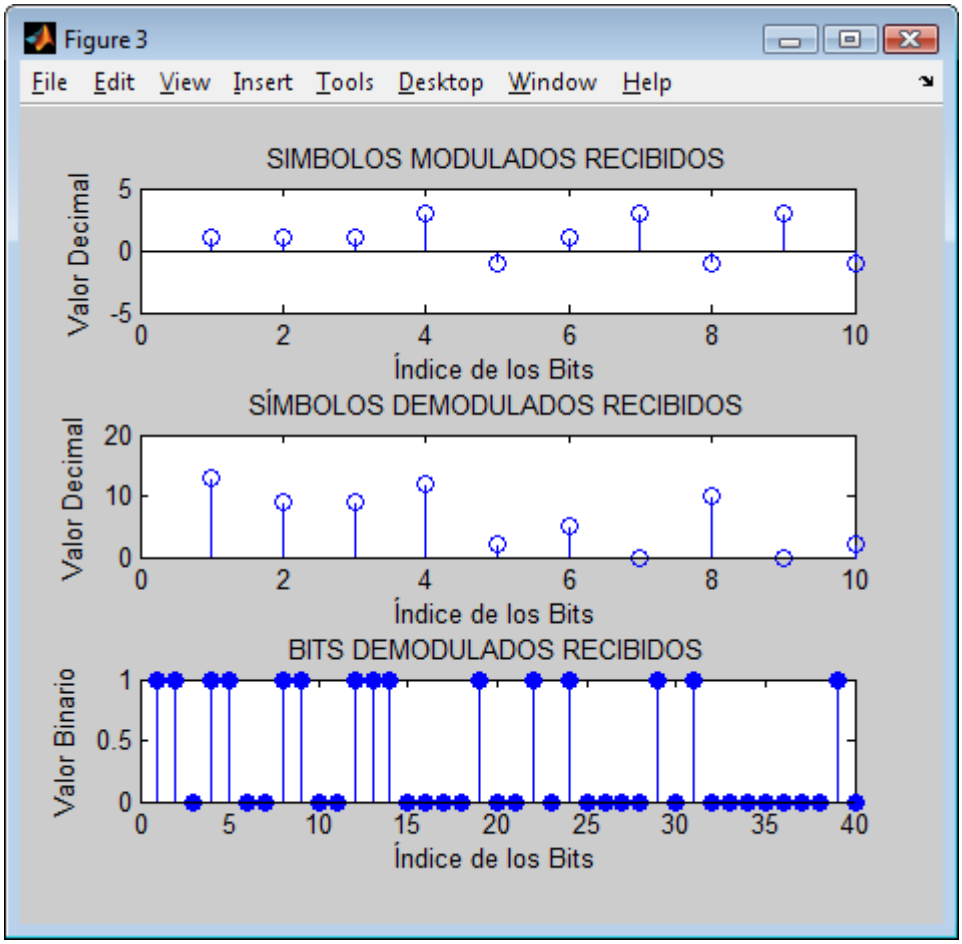


Figura 1.16. Ejemplo de Decompresión de Datos y Demodulación Sin Canal.

Con el objeto de visualizar el efecto del canal al mismo Espacio de Código 1.8 se le ha añadido un canal AWGN, con lo que el modelo quedaría de la siguiente manera:

```
n_muestras = 1; % Número de muestras por simbolo
EbNodB = 10; % Valor de la relación Eb/No en dB
SNRdB = EbNodB + 10*log10(n_bits_modulacion) - 10*log10(n_muestras); % Valor de la
relación S/N en dB
y_canal = awgn(y,SNRdB); % Canal AWGN
h = scatterplot(y_canal(1:n_muestras*5e3),n_muestras,0,'go'); hold on; % Gráfica de
Dispersión Con Canal
scatterplot(y(1:n_muestras*5e3),n_muestras,0,'k*',h); grid on; % Gráfica de Dispersión
Sin Canal
title('SEÑAL RECIBIDA VS CONSTELACIÓN DE LA SEÑAL'); % Título del Gráfico
legend('Señal Recibida','Constelación de la Señal'); % Leyenda para mostrar los detalles
de las dos señales
xlabel('En Fase'); ylabel('Cuadratura');% Título del Eje de las X e Y
figure; subplot(3,1,1); stem(y_canal(1:10));
title('SIMBOLOS MODULADOS RECIBIDOS');
xlabel('Índice de los Bits'); ylabel('Valor Decimal');

z_deci = demodulate(modem.qamdemod(M),y_canal); % Demodulación
subplot(3,1,2); stem(z_deci(1:10))
title('SÍMBOLOS DEMODULADOS RECIBIDOS');
xlabel('Índice de los Bits'); ylabel('Valor Decimal');
```

```
z = de2bi(z_deci,'left-msb'); % Cambio de decimal a binario
z = reshape(z.',prod(size(z)),1); z = z';% Desagrupamiento de bits
subplot(3,1,3); stem(x(1:40),'filled');
title('BITS DEMODULADOS RECIBIDOS');
xlabel('Índice de los Bits'); ylabel('Valor Binario');

[Numero_de_Errores,Bit_Error_Rate] = biterr(x,z)
```

Espacio de Código 1.9. Ejemplo de Decompresión de Datos y Demodulación Con Canal AWGN.

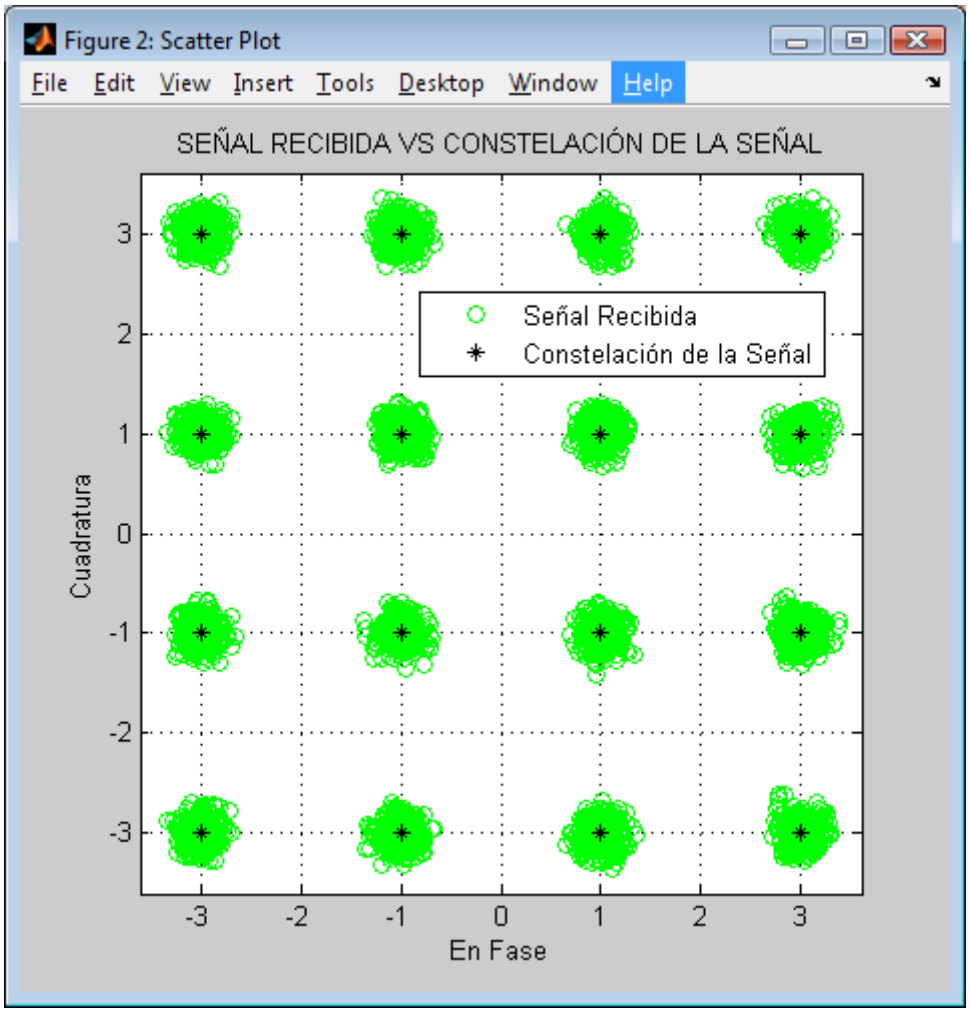


Figura 1.17. Diagrama de Señal Recibida Con Canal AWGN y Constelación de la Señal.

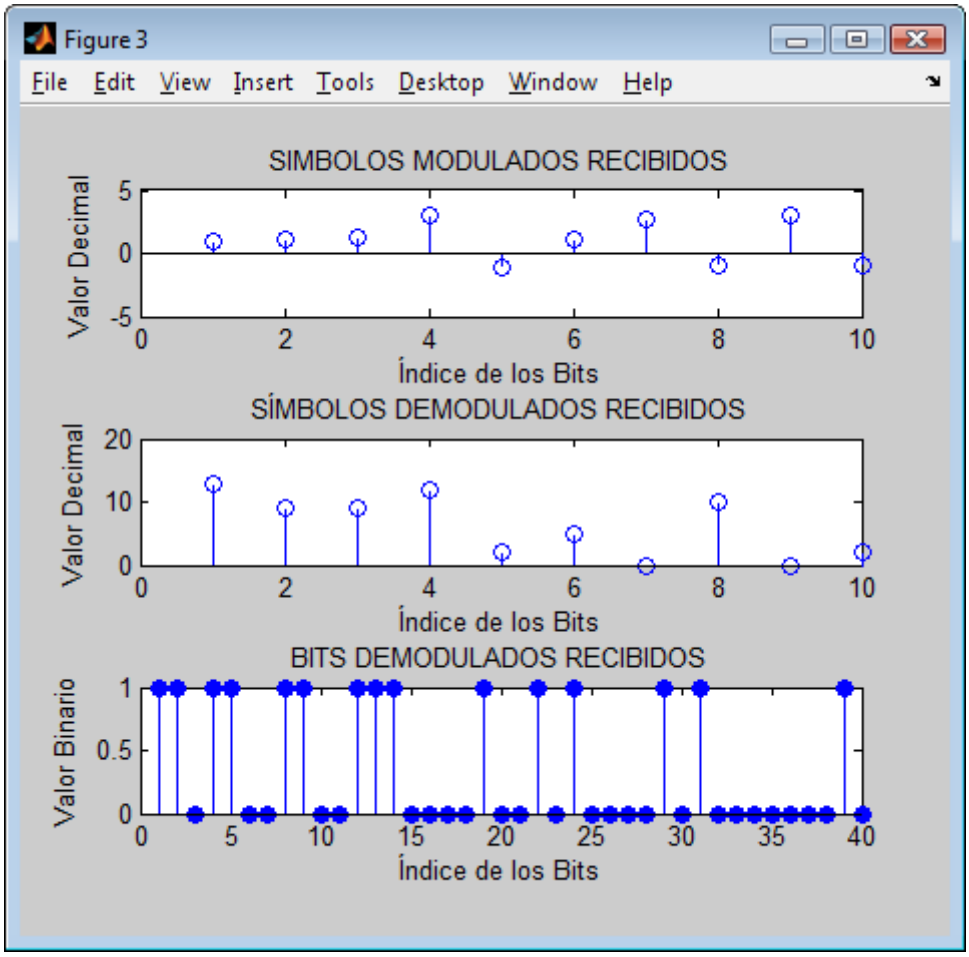


Figura 1.18. Ejemplo de Decompresión de Datos y Demodulación Con Canal AWGN.

Es necesario acotar que en este espacio de código, adicionalmente se hace un cálculo del BER mediante la función de Matlab ***biterr(x,z)***, tomando en cuenta que la señal de información generada es *x* y *z* es la señal recibida. El BER está dado por la Ecuación 1.15:

$$\text{BER} = \frac{\text{Número de Bits Errados}}{\text{Número de Bits Transmitidos}} \qquad \textbf{(Ecuación 1.15)}$$

[32]La probabilidad de error a la salida del decodificador del canal provee una medida importante del desempeño del sistema de comunicación. A menudo esta probabilidad de error la constituye el llamado BER, pero existen otras medidas de probabilidad de error como la *MSER* (*M*-ary *S*ymbol *E*rror *R*ate) que también llega a ser utilizada como referencia para medir el desempeño del sistema.

[32] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Págs: 27.

1.2.5.2 Decodificador para Control de Errores

[33]La salida del demodulador es recibida por el Decodificador para control de errores, cuya finalidad es convertir a la secuencia que pasó a través del canal en una secuencia binaria que idealmente será una réplica de la secuencia de bits de entrada. La estrategia de decodificado se basa en algoritmos de decodificación bajo las reglas de codificación del canal y características de ruido del canal. Dentro de los algoritmos de decodificado, que se abordarán en capítulos siguientes del presente proyecto de titulación, están el algoritmo de ***Viterbi*** para los ***códigos convolucionales*** y los algoritmos ***MAP*** y ***SOVA*** para el caso de los ***turbo códigos***, siendo estos últimos la parte medular de esta tesis.

1.2.6 DECODIFICADOR DE FUENTE

[34]El decodificador de fuente proporciona una representación de los datos decomprimidos; es decir que su trabajo consiste en entregar al Receptor de Información un estimado de la salida de la Fuente de Información, que para el mejor de los casos corresponderá a una reproducción fiel de dicha salida. Para el caso de una fuente analógica, esta tarea involucrará una conversión D/A[35].

1.2.7 RECEPTOR DE INFORMACIÓN

[36]Es el destino final de los datos.

1.3 TEOREMAS DE SHANNON

1.3.1 TEOREMA DE CODIFICACIÓN DE FUENTE

[37]El teorema de codificación de fuente estudia técnicas para convertir de manera eficiente fuentes arbitrarias de información en mensajes digitales, eliminando tanta redundancia como sea posible de la fuente de información.

---

[33] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Págs: 27.
[34] FUENTE: MOON, Todd K.; Error Correction Coding: Mathematical Methods and Algorithms; Pág: 9.
[35] D/A: Conversión digital- analógica, consiste en la transcripción de señales digitales en señales analógicas.
[36] FUENTE: MOON, Todd K.; Error Correction Coding: Mathematical Methods and Algorithms; Pág: 9.

1.3.2 TEOREMA DE CAPACIDAD DEL CANAL

[38]A los principales parámetros de un sistema de comunicación (ancho de banda del canal; potencia de la señal, y complejidad de la implementación física del sistema) Shannon añadió el nivel de la potencia del ruido presente en un canal de comunicación para poder derivar un parámetro denominado *Capacidad del Canal* "*C*" que definiría la máxima tasa de transmisión a la que aún es posible lograr que la información transmitida por un canal sea recibida por un usuario de manera confiable. Si la tasa de transmisión de un sistema permanece por debajo de *C*, entonces la probabilidad de error en la información transmitida podía hacerse muy baja empleando señales de transmisión codificadas suficientemente largas; por otro lado, si la tasa de transmisión sobrepasaba a la capacidad del canal entonces no era posible lograr una transmisión confiable de información por ningún medio.

[39]La capacidad de canal de un canal binario simétrico se obtiene a partir de la Ecuación 1.16:

$$C = 1 + p(log_2(p)) + (1-p)log_2(1-p) \qquad \textbf{(Ecuación 1.16)}$$

Esta capacidad del canal es igual a 1 si p = 0 o p = 1; para p = 0.5 la capacidad del canal es de 0. En contraste al canal binario simétrico, el cual toma símbolos discretos para la entrada y salida tomados desde un alfabeto binario, el llamado canal AWGN está definido en base de valores aleatorios reales continuos.

La capacidad del canal AWGN se puede calcular por medio de la Ecuación 1.17:

$$C = Wlog_2\left(1+\frac{S}{N}\right) \quad \longrightarrow \quad C = \frac{1}{2}log_2\left(1+\frac{S}{N}\right) \qquad \textbf{(Ecuación 1.17)}$$

---

[37] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Pág: 13.
[38] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Págs: 14-15.
[39] FUENTE: NEUBAUER, André; FREUDENBERGER, Jürgen; KÜHN, Volker; Coding Theory: Algorithms, Architectures, and Applications; Págs: 5-6.

La capacidad del canal exclusivamente depende de la relación señal a ruido *S/N*. Por tanto, la relación *S/N* en términos de E_b/N_o, se representa por la Ecuación 1.18:

$$\frac{S}{N} = \frac{E_b}{N_o/2} \qquad \textbf{(Ecuación 1.18)}$$

Con el fin de comparar la capacidad del canal del canal binario simétrico y el canal AWGN, se asume un esquema de transmisión digital con desplazamiento de fase binaria (BPSK) y una recepción óptima. Donde la probabilidad de error de bit del canal binario simétrico en función de E_b/N_o está dada por la Ecuación 1.19:

$$p = \frac{1}{2} erfc \sqrt{\frac{E_b}{N_o}} \qquad \textbf{(Ecuación 1.19)}$$

Donde, *erfc* denota la función de error complementaria (también conocida como función error de Gauss complementaria) que es una función no-elemental que se utiliza en el campo de la probabilidad, y está definida por la Ecuación 1.20:

$$erfc(x) = \frac{2}{\sqrt{\pi}} \int_{x}^{\infty} e^{-t^2}\, dt \qquad \textbf{(Ecuación 1.20)}$$

Sentados estos precedentes se puede hacer una comparación entre los canales BSC y AWGN que van a se representados por el Espacio de Código 1.10 desarrollado en Matlab, con el objetivo de poner de manifiesto cual de estos dos canales tiene mejor rendimiento:

```
clear all
EbNo = 0.01:0.1:100; % Variación de EbNo (en decimal)
p = 1/2 * erfc(sqrt(EbNo)); % Calcula la probabilidad en funcion de Eb/No para canal bsc
i = length (p); % Cálculo de la Capacidad de Canal BSC con fórmula: 1 + (p)log2(p) + (1-
p)(log2(1-p))
b = log2(p);
c = (1-p);
d = log2(1-p);
for k = 1:i
    e (1,k) = p(1,k) * b(1,k);
    f (1,k) = c(1,k) * d(1,k);
end
Cap_bsc = 1 + e + f; % Calcula la Capacidad de Canal para Canal BSC
Cap_AWGN = (1/2) * (log2(1+(2*EbNo))); % Calcula la Capacidad de Canal para Canal AWGN
Eje_x = 10 * log10(EbNo); % Convierte Eb/No de números decimales a dB
plot(Eje_x,Cap_AWGN,Eje_x,Cap_bsc), grid on,axis([-5 5 0 1.5]); hold on  % Grafica ambas
```

```
señales
title('CAPACIDAD DEL CANAL AWGN VS BSC'); % Título del Gráfico
xlabel('10 log (Eb/No) = (Eb/No en [dB])'),ylabel('Capacidad del Canal (C)'); % Título
del Eje de las X e Y
legend('AWGN','BSC'); % Leyenda para mostrar los detalles de las dos señales
```

Espacio de Código 1.10. Capacidad del Canal BSC Vs.AWGN

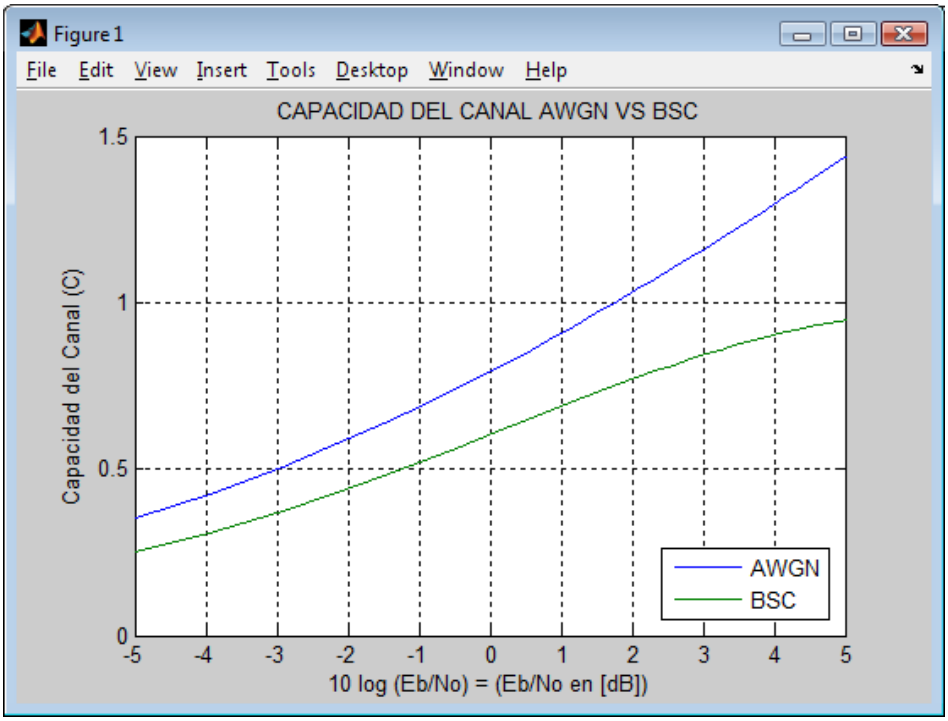


Figura 1.19. Capacidad del Canal BSC Vs.AWGN.

De la simulación se desprende que la relación señal a ruido *S/N* o la relación E_b/N_o, debe ser mayor para el canal binario simétrico en comparación con el canal AWGN con el fin de lograr la misma capacidad de canal.

[40]Aunque la teoría de la información dice que teóricamente es posible encontrar un código de canal que para un canal reduce la probabilidad de error en caso necesario, el diseño de un buen código de canal es generalmente difícil.

1.3.3 TEOREMA DE CODIFICACIÓN DEL CANAL

[41]Para transmitir información con un nivel de confiabilidad alto, Claude Shannon realizó uno de los enunciados más relevantes:

[40] NEUBAUER André; FREUDENBERGER Jürgen; KÜHN Volker; Coding Theory: Algorithms, Architectures, and Applications; Págs: 5-6.

"Si se toma secuencias largas en aumento de dígitos de fuente, y se las proyecta en formas de onda de transmisión correspondientemente largas, entonces la tasa de error en la información transmitida puede hacerse arbitrariamente cercana a cero mientras no se pretenda transmitir información a una tasa superior a la capacidad del canal C. Por lo tanto, a cualquier nivel distinto de cero de la relación Señal a Ruido (S/N) del canal, existe alguna tasa de transferencia de información distinta de cero bajo la cual puede conseguirse en principio una comunicación arbitrariamente confiable". Este resultado constituye el *Teorema de Codificación del Canal de Shannon*.

1.3.4 LÍMITE DE SHANON

[42]Para saber de manera cuantitativa y en base a las predicciones de Shannon, cual es el máximo desempeño que se puede lograr en un sistema de comunicación codificado se debe retomar la Ecuación 1.17 y reescribirla como:

$$\frac{C}{W} = log_2\left(1 + \frac{E_b R}{N_o W}\right) \qquad \textbf{(Ecuación 1.21)}$$

[43]Un sistema ideal es aquel cuya tasa de transmisión *R* es igual a la capacidad del canal *C*. Por lo que la Ecuación 1.21 se puede escribir como:

$$\frac{(E_b)_{min}}{N_o} = \frac{2^{C/W} - 1}{C/W} \qquad \textbf{(Ecuación 1.22)}$$

Donde $(E_b)_{\min}$ representa la mínima energía requerida por bit de información transmitido para una comunicación confiable. A la razón *R*/*W* se le conoce como *Eficiencia del Ancho de Banda*. En un sistema ideal la máxima eficiencia de

---

[41] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Págs: 17-18.
[42] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Págs: 20-21.
[43] FUENTE: SOLEYMANI, M. R.; GAO, Yingzy; VILAIPORNSAWAI, U.; Turbo Coding for Satellite and Wireless Communications; Págs: 13-14.

energía se alcanza cuando *C/W* = 0. En este punto, E_b/N_o es igual al llamado *Límite de Shannon* que corresponde a –1.6 dB y se muestra en la Figura 1.20.

```
clear all
x = 0:0.01:1; % Generación de un vector entre 0 y 1 (Tasa de Código)
EbNo = 10*log10((2.^(2*x)-1)./(2*x)); % Cálculo de la Eb/No
plot(EbNo,x), grid; % Grafica Eb/No Vs Tasa de código(x)
xlabel('Eb/No (dB)'); % Título del Eje de las X
ylabel('Tasa de Código'); % Título del Eje de las Y
title('LÍMITE DE CAPACIDAD DE SHANNON'); % Título del Gráfico
```

Espacio de Código 1.11. Límite de Capacidad de Shannon.

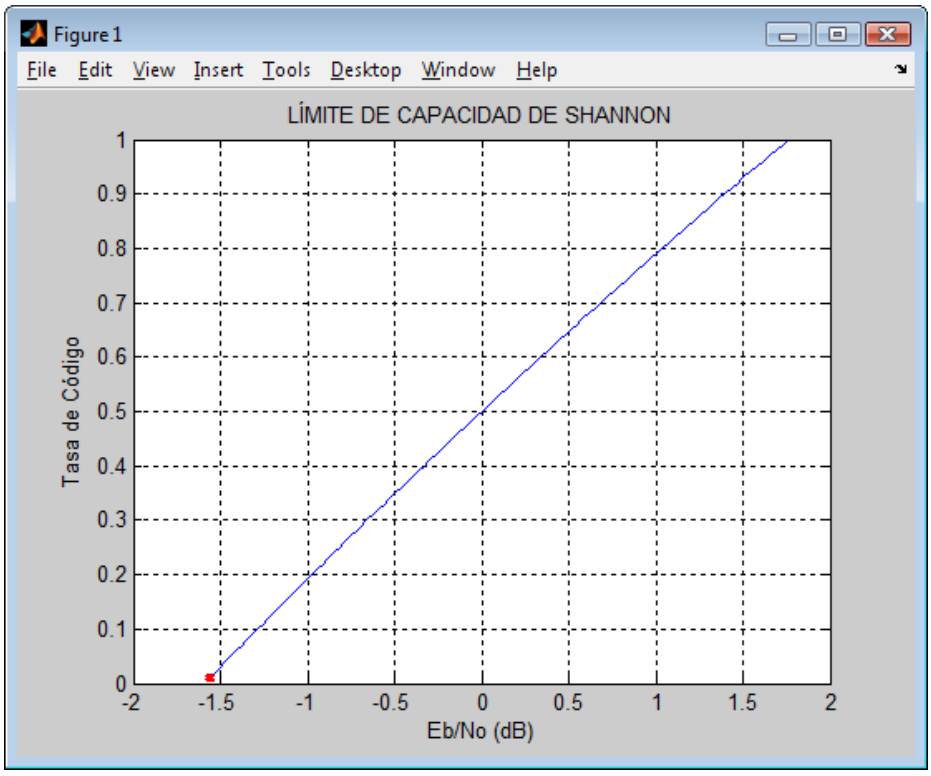


Figura 1.20. Límite de Capacidad de Shannon.

Finalmente, para realizar los cálculos de rendimiento de un canal sin codificar y observar la diferencia con respecto a los códigos para detección de errores analizados posteriormente (bloque, convolucionales, turbo) el Espacio de Código 1.12 elabora una gráfica del BER vs E_b/N_o, con modulación BPSK para un canal AWGN sin codificar, en el cual se realiza únicamente modulación, implementación del canal y demodulación con decisiones duras, como se muestra en el diagrama de bloques de la Figura 1.21. El objeto *modem.psk((M,pi))*, permite modular con *M* estados, en este caso el valor de *M* es 2, por lo que se determina que la modulación es BPSK, y tiene un *offset* de *pi* radianes (180°), con el objetivo de

que al 0 le corresponda un valor de -1+0i, y al 1 un valor de 1. La función *modulate(modem.pskmod(M,pi),x)* modula la señal de información *x*, tomando al objeto *modem* con los parámetros ya configurados para BPSK. Mientras que *demodulate(modem.pskmod(M,pi),y_canal)* realiza el proceso inverso de demodulación (con decisiones duras) de la señal con ruido gaussiano *y_canal*.

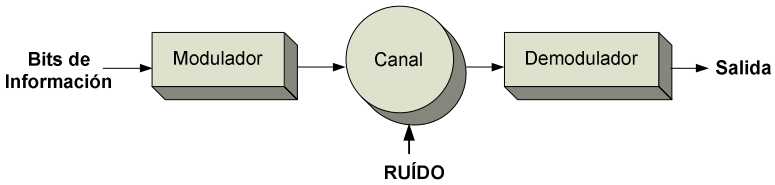


Figura 1.21. Esquema de un Canal AWGN sin Codificar.

```
clear all

load ('x.txt'); % Carga el archivo de texto con los bits generados
x = x; % Conserva el mismo nombre de la variable
% Si se desea generar una nueva secuencia de bits de información se debe:
% comentar las dos lineas anteriores de programa e introducir: x = randint(1,40000);
M = 2; % Número de niveles o estados de la señal modulada
n_bits_modulacion = log2(M); % Número de bits por símbolo
y = modulate(modem.pskmod(M,pi),x); % Bits Modulados (BPSK)
EbNodB = 0:11; % Variación de EbNo (en decibelios)
SNRdB = EbNodB + 3 + 10*log10(n_bits_modulacion); % Cálculo de la SNR (en decibelios)
w = 0; % Inicializa w, que permite realizar la variación de Eb/No para poder graficar
k = length(SNRdB);
for p1 = 1:k
    y_canal = awgn(real(y),SNRdB(1,p1)); % Canal AWGN
    demodu = demodulate(modem.pskdemod(M,pi),y_canal); % Demodulador de Decisiones Duras
    [numero_de_errores, bit_error_rate] = biterr(x,demodu); % Cálculo del Bit Error Rate
    w = w+1;
    BER_AWGN(w) = bit_error_rate; % Vector Bit Error Rate
end
BER_AWGN
semilogy(EbNodB, BER_AWGN); % Grafica en escala semilogarítmica BER Vs Eb/No
xlabel('Eb/No en [dB]'); ylabel('Bit Error Rate (BER)'); grid on; % Título del Eje de
las X e Y
title('RENDIMIENTO DE UN CANAL AWGN SIN CODIFICAR'); % Título del Gráfico
legend('Canal AWGN Sin Codificar'); % Leyenda para mostrar los detalles de la señal
```

Espacio de Código 1.12. Rendimiento de un Canal AWGN sin Codificar.

E_b/N_o (en dB)	0	1	2	3	4	5
BER (Sin Codificar)	0.079079	0.056903	0.036152	0.023051	0.012876	0.0063253
E_b/N_o (en dB)	6	7	8	9	10	11
BER (Sin Codificar)	0.0023001	0.00082504	0.00012501	5.0003e-005	0	0

Tabla 1.1. Resultados de la Simulación de un Canal AWGN sin Codificar.

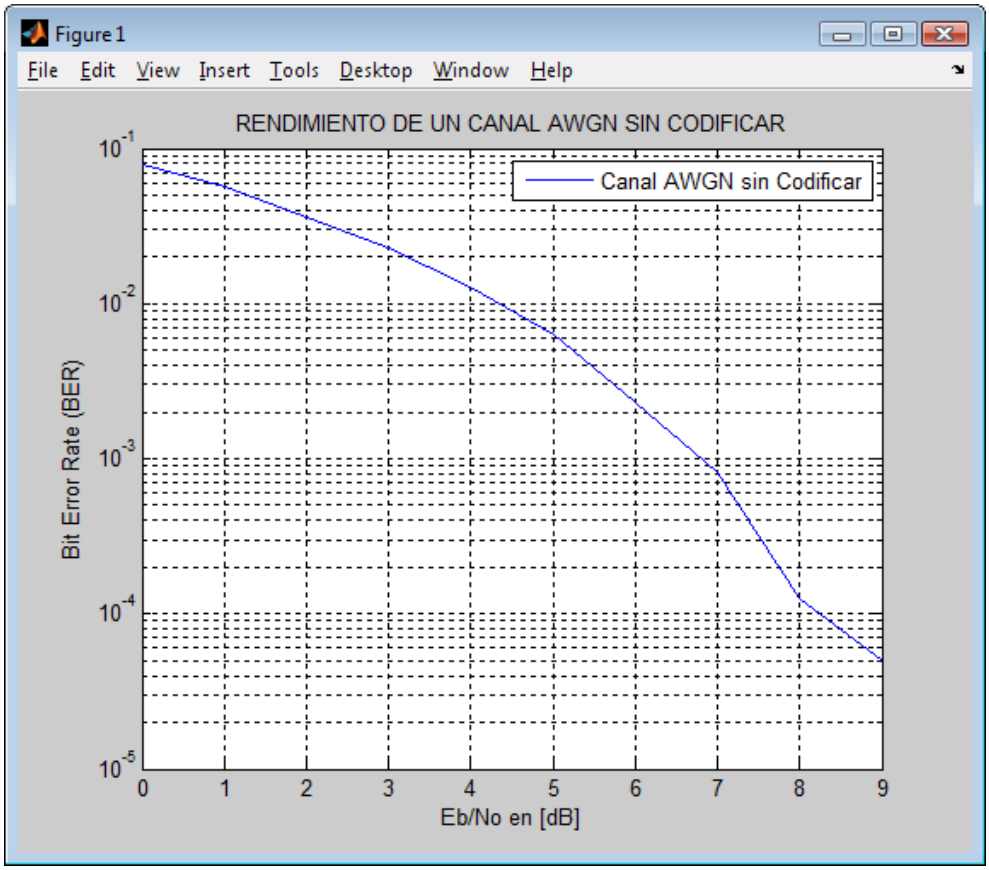


Figura 1.22. Rendimiento de un Canal AWGN sin Codificar.

CAPÍTULO 2

ESQUEMAS DE CODIFICACIÓN Y TURBO CÓDIGOS

2.1 INTRODUCCIÓN

La consecuencia de los efectos del canal (ruido, atenuación, distorsión, etc) es que la secuencia binaria que llega a su destino contenga errores. Uno de los objetivos más importantes del diseño del sistema de comunicaciones será la reducción de la probabilidad de error en esta secuencia. Por tanto, es necesario abordar en este capítulo el diseño de técnicas de codificación que consigan un compromiso adecuado entre redundancia del mensaje y eficiencia en la transmisión del mismo.

2.2 ESQUEMAS DE CODIFICACIÓN BEC Ó ARQ

[44]El *Automatic Repeat Request* (*ARQ*) o también llamada *Backward Error Correction (BEC)* es una estrategia para control de errores en la que el nivel deseado de precisión en la información recibida se consigue mediante retransmisiones del mensaje enviado, dado que se emplean esquemas de codificación que son capaces de detectar errores, pero no de corregirlos. En un sistema ARQ, cuando se detectan errores en el receptor, se envía una solicitud de retransmisión del mensaje al transmisor, y esto se repite hasta que el mensaje es recibido correctamente por el receptor.

2.2.1 FUNCIONAMIENTO

[45]ARQ funciona exclusivamente para sistemas de comunicación *half dúplex*[46] o *full dúplex*[47]. Existen dos tipos de sistemas ARQ en general:

---

[44] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Pág: 29.
[45] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Págs: 29-30.
[46] HALF DÚPLEX: Sistema capaz de mantener una comunicación bidireccional, enviando y recibiendo mensajes de forma no simultánea.

- *ARQ Stop and Wait*: el transmisor envía una palabra de código al receptor y espera una respuesta ACK[48] o NAK[49] de este último.
- *ARQ Continuous:* el transmisor envía palabras código al receptor constantemente y recibe respuestas de este último de la misma manera. Cuando un NAK es recibido, el transmisor debe iniciar una retransmisión desde la palabra de código que fue detectada errónea (*go back N ARQ*) o únicamente la palabra de código a la que corresponde el NAK recibido (*selective repeat ARQ*).

2.2.2 VENTAJAS

[50]La mayor ventaja de ARQ sobre FEC es que la detección de errores requiere un equipo de decodificación menos complejo que el que requiere la corrección de errores. Otra ventaja que presenta ARQ con respecto a FEC, es que con FEC puede darse el caso en que la secuencia decodificada binaria aún contenga algunos errores que serían ya incorregibles, mismos que podrían ser corregidos en la retransmisión de ARQ. Los esquemas de codificación más empleados por los sistemas ARQ son los *Códigos de Paridad* y los *Códigos de Redundancia Cíclica* (*CRC – Cyclic Redundancy Check codes*).

2.3 ESQUEMAS DE CODIFICACIÓN FEC

[51]*FEC (Forward Error Correction)* es un tipo de mecanismo de detección de errores que permite su corrección en el receptor sin retransmisión de la información original. Se utiliza en sistemas sin retorno o sistemas en tiempo real donde no se puede esperar a la retransmisión para mostrar los datos. Durante la década de 1970, los códigos FEC se han incorporado en varias aplicaciones de

---

[47] FULL DÚPLEX: Sistema capaz de mantener una comunicación bidireccional, enviando y recibiendo mensajes de forma simultánea.
[48] ACK: ACKnowledgement, o acuse de recibo, es un mensaje que se envía para confirmar que un mensaje o un conjunto de mensajes han llegado. El significado de ACK es "ha llegado y ha llegado correctamente".
[49] NAK: Negative ACKnoledgement, o asentimiento negativo, que se suele enviar cuando se ha detectado un error en la trama recibida o cuando se ha perdido una trama.
[50] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Págs: 30-31.
[51] FUENTE: HANZO, L.; LIEW, T.; YEAP, B.; Turbo Coding, Turbo Equalization and Space-Time Coding for Transmission over Wireless Channels; Pág: 13.

espacio profundo y sistemas de comunicaciones por satélite, y en la década de 1980 también se hizo común en casi todos los sistemas móviles celulares.

2.3.1 FUNCIONAMIENTO

[52]La posibilidad de corregir errores se consigue añadiendo al mensaje original unos bits de redundancia. La fuente digital envía la secuencia de datos al codificador, encargado de añadir dichos bits de redundancia. A la salida del codificador se obtiene la denominada palabra código. Esta palabra código es enviada al receptor y éste, mediante el decodificador adecuado y aplicando los algoritmos de corrección de errores, obtendrá la secuencia de datos original.

2.3.2 VENTAJAS

[53]FEC reduce el número de errores, mediante la corrección de la secuencia transmitida; así como también reduce los requisitos de potencia de los sistemas de comunicación e incrementa la efectividad de los mismos evitando la necesidad del reenvío de los mensajes dañados durante la transmisión.

Los dos principales tipos de codificación FEC usados son:

- *Códigos de Bloque:* Un código bloque (*n,k*) toma *k* bits de la fuente y produce un bloque de tamaño *n*. Cada bloque se codifica de forma independiente, sin memoria. La paridad en el codificador se introduce mediante un algoritmo algebraico aplicado al bloque de bits. El decodificador aplica el algoritmo inverso para poder identificar y, posteriormente corregir los errores introducidos en la transmisión. Los códigos de bloque pueden clasificarse como se muestra en la Figura 2.1:

[52] FUENTE: ALVARADO, Raúl; Códigos para Detección y Corrección de Errores en Comunicaciones Digitales; Págs: 52.
[53] FUENTE: SACANAMBOY, Maribell; Tesis de Maestría: Diseño e Implementación de los Turbo Codificadores definidos en los estándares de Telecomunicaciones cdma2000 (TIA/EIA 2002.2D) y WCDMA (3GPP TS 25.212 v7.2.0) usando Hardware Reconfigurable; Pág: 4.

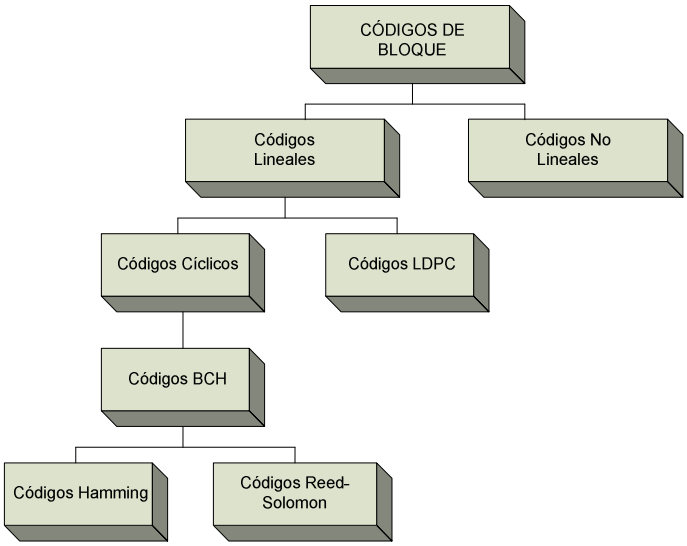


Figura 2.1. Clasificación de los Códigos de Bloque. 54

Una propiedad que se desea que tenga un código bloque, es la linealidad. Ya que con esta estructura, la complejidad de la codificación se reduce enormemente.

El Espacio de Código 2.1 elabora una gráfica del BER vs. E_b/N_o, con modulación BPSK a través de un canal AWGN con código de Hamming, siguiendo el esquema que se muestra en la Figura 2.2. La demodulación de los bits recibidos toma decisiones duras. Se ha hecho uso de las funciones ***encode*** y ***decode*** que se encuentran dentro del toolbox de comunicaciones de Matlab, para codificar y decodificar la secuencia de bits, respectivamente. La función *encode*, utilizada con la sintaxis *encode(x,n,k,'hamming/binary')*, toma como entrada a la señal de información *x* de datos binarios, a la longitud de la palabra código *n*, y a *k* que es la longitud del mensaje para incorporar un código de Hamming con entrada de datos binarios y no decimales (como usualmente se lo hace). Mientras que la función con sintaxis *decode(demodul,n,k,'hamming/binary'),* recibe la señal demodulada (*demodul*), con los mismos parámetros de *n* y *k* para decodificar a la señal. Se debe indicar el tipo de código para la decodificación mediante la instrucción *'hamming/binary'*.

[54] FUENTE: THE MATHWORKS; Communications Toolbox™ 4: User's Guide; Sección: 6, Pág: 4.

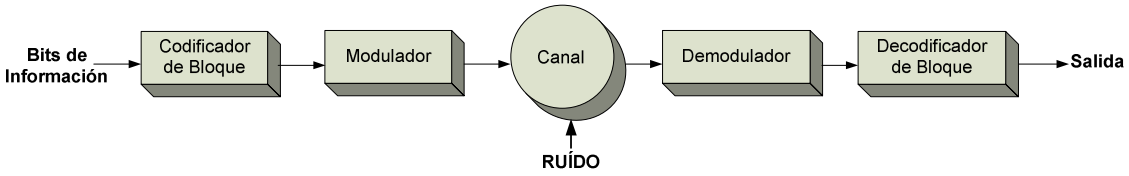


Figura 2.2. Esquema de un Código de Bloque sobre canal AWGN.

```
m = 4; % Longitud de la redundancia del código de bloque
n = 2^m-1; k = n-m; % Longitud de la palabra código y mensaje, respectivamente.
load ('x.txt'); % Carga el archivo de texto con los bits generados
x = x; % Conserva el mismo nombre de la variable
% Si se desea generar una nueva secuencia de bits de información se debe:
% comentar las dos lineas anteriores de programa e introducir: x = randint(1,40000);
codificador_bloque = encode(x,n,k,'hamming/binary'); % Codificación de Bloque (Hamming)
de la Trama
M = 2; % Número de niveles o estados de la señal modulada
n_bits_modulacion = log2(M); % Número de bits por símbolo
y = modulate(modem.pskmod(M,pi),codificador_bloque); % Bits Modulados (BPSK)
EbNodB = 0:11; % Variación de EbNo (en decibelios)
SNRdB = EbNodB + 3 + 10*log10(n_bits_modulacion); % Cálculo de la SNR (en decibelios)
w = 0; % Inicializa w, que permite realizar la variación de Eb/No para poder graficar
kk = length(SNRdB);
for p1 = 1:kk
    y_canal = awgn(real(y),SNRdB(1,p1)); % Canal AWGN
    demodul = demodulate(modem.pskdemod(M,pi),y_canal); % Demodulador de Decisiones
Duras
    decodificador_bloque = decode(demodul,n,k,'hamming/binary'); % Decodificación del
Código de Hamming
    [numero_de_errores, bit_error_rate] = biterr(x,decodificador_bloque(1:length(x))); %
Cálculo del Bit Error Rate
    w = w+1;
    BER_Bloque(w) = bit_error_rate; % Vector Bit Error Rate
end
BER_Bloque
semilogy(EbNodB,BER_Bloque); % Grafica en escala semilogarítmica BER Vs Eb/No
xlabel('Eb/No en [dB]'); ylabel('Bit Error Rate (BER)'); grid on; % Título del Eje de
las X e Y
title('RENDIMIENTO DE UN CÓDIGO DE BLOQUE CON CANAL AWGN'); % Título del Gráfico
legend('Código de Bloque con Canal AWGN'); % Leyenda para mostrar los detalles de la
señal
```

Espacio de Código 2.1. Rendimiento de un Código de Hamming sobre canal AWGN.

E_b/N_o (en dB)	0	1	2	3	4	5
BER (con código Bloque)	0.073529	0.042927	0.021651	0.0099255	0.0033002	0.00077504
E_b/N_o (en dB)	6	7	8	9	10	11
BER (con código Bloque)	5.0003e-005	5.0003e-005	0	0	0	0

Tabla 2.1. Resultados de la Simulación de un Código de Hamming sobre canal AWGN.

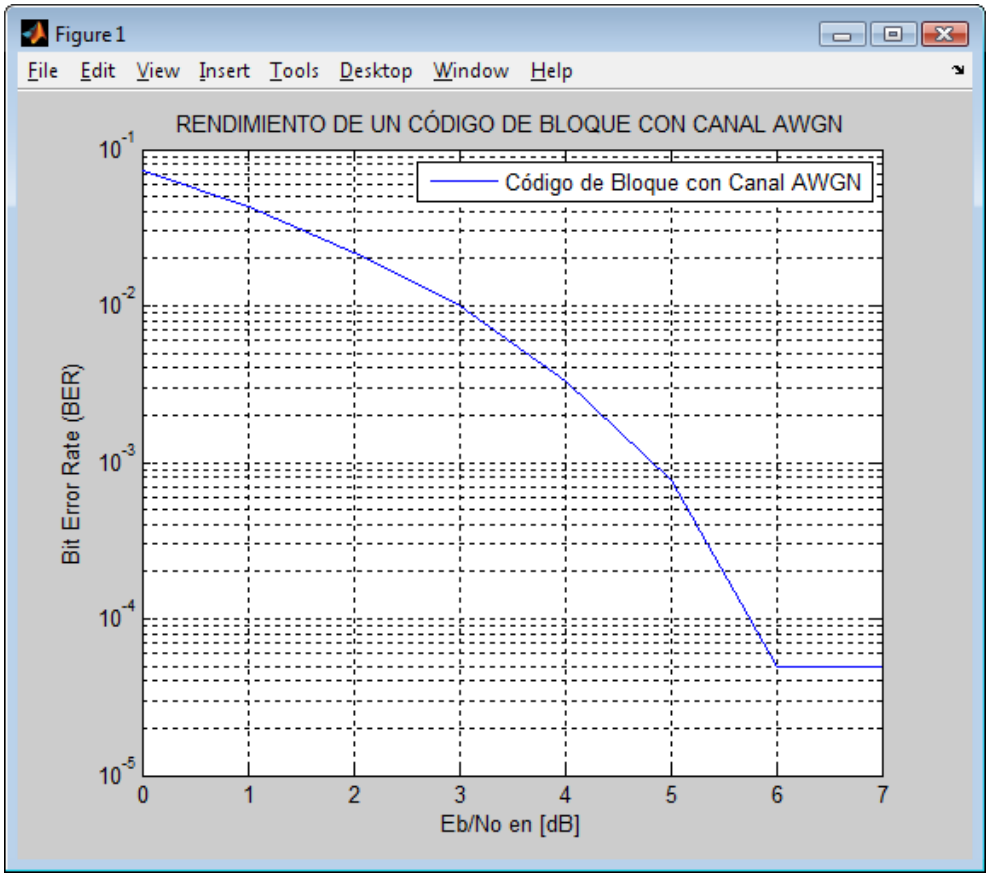


Figura 2.3. Rendimiento de un Código de Hamming sobre canal AWGN.

Con la finalidad de cuantificar la mejora de una secuencia de bits al ser incorporado un código para la detección y control de errores, se ha agregado el Espacio de Código 2.2 que permite realizar la comparación de rendimiento entre el canal AWGN sin codificar expuesto en el Espacio de Código 1.12 y el mismo canal con el código de bloque del Espacio de Código 2.1. Los resultados de la simulación se muestran en la Figura 2.4, donde se percibe claramente como el código de bloque mejora notablemente el rendimiento de la secuencia de bits transmitida.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%% AWGN SIN CODIFICAR %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load ('x.txt'); % Carga el archivo de texto con los bits generados
x = x; % Conserva el mismo nombre de la variable
% Si se desea generar una nueva secuencia de bits de información se debe:
% comentar las dos lineas anteriores de programa e introducir: x = randint(1,40000);
M = 2; % Número de niveles o estados de la señal modulada
n_bits_modulacion = log2(M); % Número de bits por símbolo
y = modulate(modem.pskmod(M,pi),x); % Bits Modulados (BPSK)
EbNodB = 0:11; % Variación de EbNo (en decibelios)
SNRdB = EbNodB + 3 + 10*log10(n_bits_modulacion); % Cálculo de la SNR (en decibelios)
w = 0; % Inicializa w, que permite realizar la variación de Eb/No para poder graficar
```

```
k = length(SNRdB);
for p1 = 1:k
    y_canal = awgn(real(y),SNRdB(1,p1)); % Canal AWGN
    demodu = demodulate(modem.pskdemod(M,pi),y_canal); % Demodulador de Decisiones Duras
    [numero_de_errores, bit_error_rate] = biterr(x,demodu); % Cálculo del Bit Error Rate
    w = w+1;
    BER_AWGN(w) = bit_error_rate; % Vector Bit Error Rate
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%% CODIGO DE BLOQUE  %%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

m = 4; % Longitud de la redundancia del código de bloque
n = 2^m-1; k = n-m; % Longitud de la palabra código y mensaje, respectivamente.
codificador_bloque = encode(x,n,k,'hamming/binary'); % Codificación de Bloque (Hamming)
de la Trama
y = modulate(modem.pskmod(M,pi),codificador_bloque); % Bits Modulados (BPSK)
w = 0; % Inicializa w, que permite realizar la variación de Eb/No para poder graficar
kk = length(SNRdB);
for p1 = 1:kk
    y_canal = awgn(real(y),SNRdB(1,p1)); % Canal AWGN
    demodul = demodulate(modem.pskdemod(M,pi),y_canal); % Demodulador de Decisiones
Duras
    decodificador_bloque = decode(demodul,n,k,'hamming/binary'); % Decodificación del
Código de Hamming
    [numero_de_errores, bit_error_rate] = biterr(x,decodificador_bloque(1:length(x))); %
Cálculo del Bit Error Rate
    w = w+1;
    BER_Bloque(w) = bit_error_rate; % Vector Bit Error Rate
end
BER_AWGN
BER_Bloque
semilogy(EbNodB,BER_AWGN,'b-x',EbNodB,BER_Bloque,'g-o'); % Grafica en escala
semilogarítmica BER Vs Eb/No
xlabel('Eb/No en [dB]'); ylabel('Bit Error Rate (BER)'); grid on; hold on; % Título del
Eje de las X e Y
title('COMPARACIÓN DE RENDIMIENTO'); % Título del Gráfico
legend('Canal AWGN sin Codificar','Código de Bloque con Canal AWGN'); % Leyenda para
mostrar los detalles de las señales
hold off;
```

Espacio de Código 2.2. Comparación de Rendimiento de un Canal AWGN sin Codificar y un Código de Hamming sobre canal AWGN.

E_b/N_o (en dB)	0	1	2	3	4	5
BER (Sin Codificar)	0.079079	0.056903	0.036152	0.023051	0.012876	0.0063253
BER (con Código de Bloque)	0.073529	0.042927	0.021651	0.0099255	0.0033002	0.00077504
E_b/N_o (en dB)	6	7	8	9	10	11
BER (Sin Codificar)	0.0023001	0.00082504	0.00012501	5.0003e-005	0	0
BER (con Código de Bloque)	5.0003e-005	5.0003e-005	0	0	0	0

Tabla 2.2. Resultados de la Comparación de Rendimiento de un Canal AWGN sin Codificar y un Código de Bloque sobre canal AWGN.

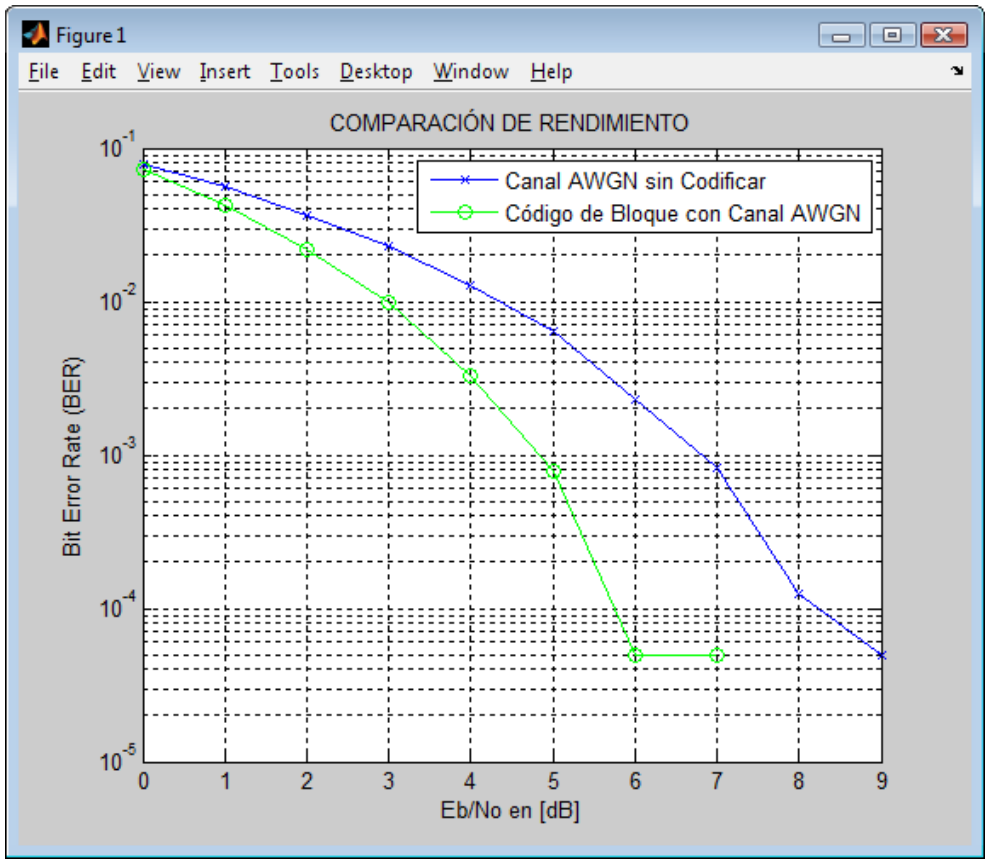


Figura 2.4. Comparación de Rendimiento Teórico de un Canal AWGN sin Codificar y un Código de Bloque sobre canal AWGN.

- *Códigos Convolucionales:* Los bits se van codificando en forma serial, además poseen una cantidad de memoria para estimar la secuencia de datos más probable para los bits recibidos; por lo que son más eficientes que los códigos de bloque.

2.4 CÓDIGOS CONVOLUCIONALES

[55]Los códigos convolucionales datan del año 1955, fecha en la que fueron descubiertos por Elias, mientras que Wozencraft y Reiffen, así como Fano y Massey, han propuesto varios algoritmos para su decodificación. Un hito importante en la historia de la corrección de errores con codificación

[55] FUENTE: HANZO, L.; LIEW, T.; YEAP, B.; Turbo Coding, Turbo Equalization and Space-Time Coding for Transmission over Wireless Channels; Pág: 13.

convolucional, fue la invención de un algoritmo de máxima probabilidad llamado Algoritmo de Viterbi en el año de 1967.

2.4.1 EL CODIFICADOR CONVOLUCIONAL

[56]Los códigos convolucionales son comúnmente especificados por tres parámetros: (*n*, *k*, *m*), (donde *n* > *k*) Donde:

- *n* = número de bits de salida (o llamada *Palabra Código*)
- *k* = número de bits de entrada
- *m* = número de registros de memoria (o memoria)

La cantidad *k/n* se llama ***tasa de código*** y es representada por *R*, siendo esta una medida de la eficiencia del código. Comúnmente *k* y *n* son parámetros con rango de 1 a 8, *m* de 2 a 10 y la tasa de código desde 1/8 a 7/8 con excepción de aplicaciones de espacio profundo donde las tasas de código son tan bajas como 1/100. A menudo, los fabricantes de chips de códigos convolucionales especifican dentro de los parámetros de código (*n*, *k*, *K*), la cantidad *K* se denomina ***constraint length*** y se define por la Ecuación 2.1:

$$\textit{Constraint Length}: K = m+1 \qquad \textbf{(Ecuación 2.1)}$$

El *constraint length*, *K* representa el número de bits en la memoria del codificador que afectan a la generación de los *n* bits de salida.

Un código convolucional se estructura teniendo en cuenta sus parámetros; primero se dibuja *m*+1 cuadros que representan los *m* registros de memoria y el cuadro adicional representará al bit de la secuencia de información a codificar (para el gráfico de la Figura 2.5 representado con un color menos intenso). A continuación, se dibuja *n* sumadores en módulo-2 para representar los bits de salida *n*. Luego se conectan los registros de memoria a los sumadores con el

[56] FUENTE: LANGTON, Charan; Intuitive Guide to Principles of Communications: Coding and Decoding with Convolutional Codes; Pág: 1.

polinomio generador, como se muestra en el Figura 2.5, para un código (2,1,3) con polinomios generadores g_0 = (1 1 1 1) y g_1 = (1 1 0 1).

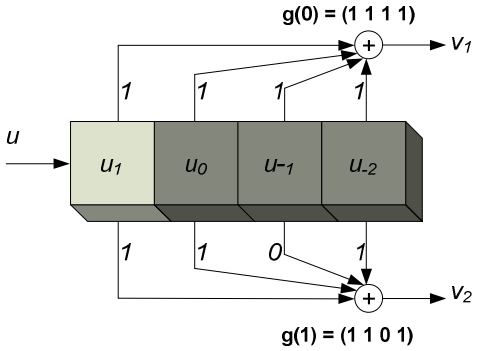


Figura 2.5. Codificador Convolucional Binario (2, 1, 3).[57]

[58]En este código de tasa 1/2, cada bit de entrada se codifica con 2 bits de salida, el *constraint length* es 4. Los 2 bits de salida son producidos por los 2 sumadores módulo-2, mediante la suma de ciertos bits en el registro de memoria. La selección de cuales bits van a ser sumados para obtener el bit de salida se llama polinomio generador *g*. El polinomio generador da al código esta calidad única de protección contra errores, es así que un mismo código (2,1,3) puede tener propiedades completamente diferentes dependiendo del polinomio generador elegido. Los bits de salida son la suma de estos bits:

$$v_1 = \text{mod-2}\ (u_1+u_0+u_{-1}+u_{-2})$$
$$v_2 = \text{mod-2}\ (u_1+u_0+u_{-2})$$

[59]Las sumas en módulo-2 pueden ser vistas como un sistema con cascada de compuertas *or* exclusivas (*xor*). La tabla de verdad que se va a seguir para obtener la salida del codificador en cada instante de tiempo, determina que cuando se tiene un número impar de 1s la salida es 1, en todos los otros casos la salida corresponderá a un 0.

---

[57] FUENTE: LANGTON, Charan; Intuitive Guide to Principles of Communications: Coding and Decoding with Convolutional Codes; Pág: 3.
[58] FUENTE: LANGTON, Charan; Intuitive Guide to Principles of Communications: Coding and Decoding with Convolutional Codes; Pág: 2.
[59] FUENTE: CABALLERO Z., Rafael; Master's Thesis: Use of Convolutional Codes for Collaborative Networks; Pág: 4.

A	B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 2.3. Tabla de verdad de la compuerta XOR. [60]

Hay muchas maneras de escoger el polinomio generador para cualquier código de orden *K*. No todos los resultados en las secuencias de salida tienen buenas propiedades de protección contra errores pero, generalmente, los polinomios generadores son encontrados por simulaciones computarizadas. Para las simulaciones de Turbo Códigos en este proyecto de titulación estos se han escogido de acuerdo a estándares de comunicaciones terrestres y móviles.

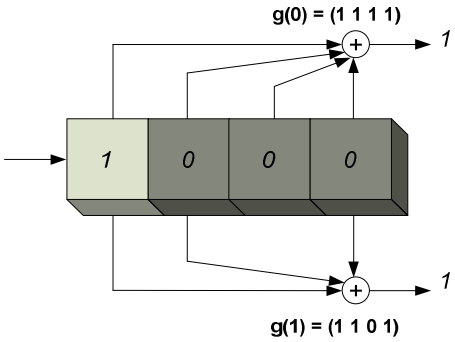
[61]Una de las características de un código convolucional es que la salida del codificador no sólo depende de los *k* bits de entrada sino de los *K* – 1 bits precedentes. Es así que el número de combinaciones de bits en los registros son llamados los estados del código y están definidos por 2^m, donde *m* es la memoria.

[62]La secuencia de la salida *v*, se puede calcular por convolución (de ahí su nombre) de la secuencia de entrada *u* con la respuesta impulso *g*, y se puede expresar como la Ecuación 2.2:

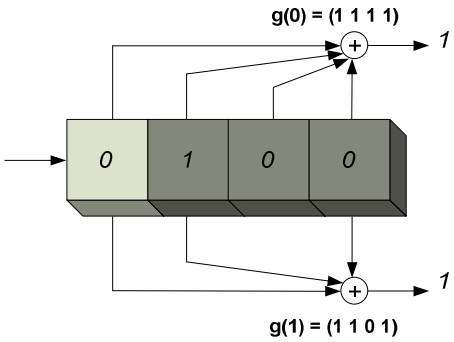
$$v = u * g \qquad \textbf{(Ecuación 2.2)}$$

Por ejemplo, para la generación de la palabra código para un codificador (2,1,3), considérese la secuencia de información *u* = (1 0 1 1). Las secuencias de salida serán:

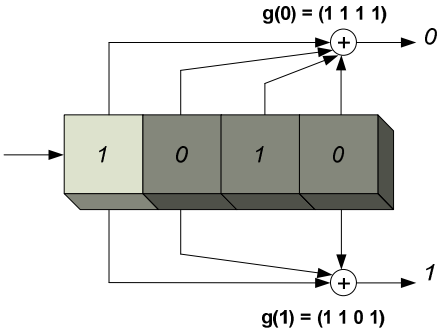
[60] FUENTE: CABALLERO Z., Rafael; Master's Thesis: Use of Convolutional Codes for Collaborative Networks; Pág: 4.
[61] FUENTE: ROQUE; SÁENZ; PEÑA; Comunicación de Datos: Códigos Convolucionales; Pág: 1.
[62] FUENTE: LANGTON, Charan; Intuitive Guide to Principles of Communications: Coding and Decoding with Convolutional Codes; Pág: 5.



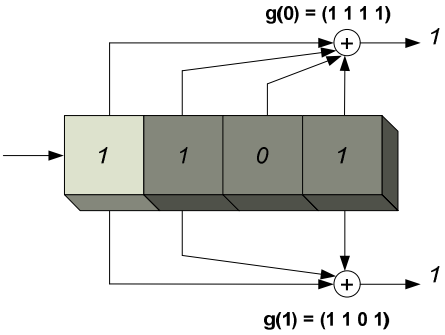
a) t =0; Estado de Entrada = 000;
Bit de Entrada = 1; Bits de Salida = 11



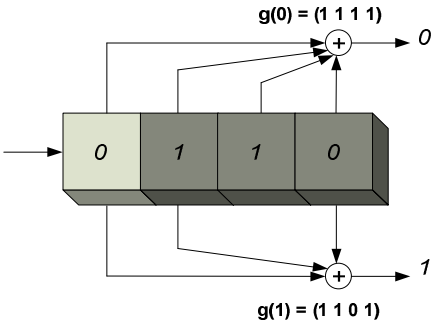
a) t =1; Estado de Entrada = 100;
Bit de Entrada = 0; Bits de Salida = 11



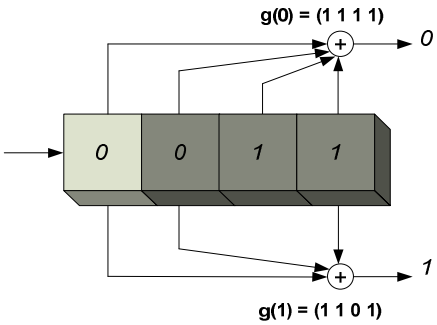
a) t =2; Estado de Entrada = 010;
Bit de Entrada = 1; Bits de Salida = 01



a) t =3; Estado de Entrada = 101;
Bit de Entrada = 1; Bits de Salida = 11



a) t =4; Estado de Entrada = 110;
Bit de Entrada = 0 (bit de cola); Bits de Salida = 01



a) t =5; Estado de Entrada = 011;
Bit de Entrada = 0 (bit de cola); Bits de Salida = 01

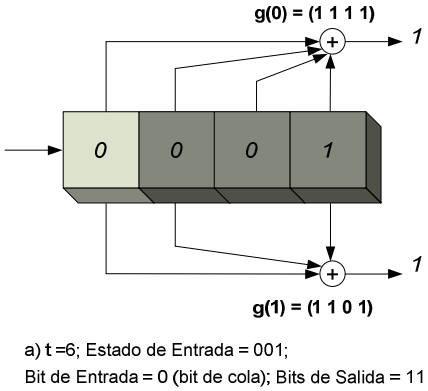


Figura 2.6. Codificador Convolucional Binario (2, 1, 3). [63]

[64]Los cuatro bits que conforman este mensaje *u* = (1 0 1 1), (en realidad, es un fragmento de una secuencia de bits de longitud indefinida) ingresan de a uno a la vez, en los tiempos t_0, t_1, t_2, y t_3, como se muestra en la Figura 2.6. Al final de la secuencia, en los tiempos t_4, t_5 y t_6, ingresan tres ceros para permitir que el último 1 del mensaje pueda llegar hasta el registro *K*. La secuencia de salida es 11110111010111, donde los bits de más a la izquierda se corresponden a los bits de mensaje que ingresaron primero.

También es posible ver el codificador en términos de su respuesta impulsiva; es decir, la respuesta a una entrada que es un solo bit "1" que se mueve a través de los registros, completado con los ceros necesarios. Teniendo en cuenta esto y considerando otra vez el codificador de la Figura 2.5, los registros y la salida varían de la siguiente manera:

Registros	V_1	V_2
1 0 0 0	1	1
0 1 0 0	1	1
0 0 1 0	1	0
0 0 0 1	1	1

Tabla 2.4. Respuesta Impulsiva de bit "1". [65]

[63] FUENTE: LANGTON, Charan; Intuitive Guide to Principles of Communications: Coding and Decoding with Convolutional Codes; Pág: 1.
[64] FUENTE: ROQUE; SÁENZ; PEÑA; Comunicación de Datos: Códigos Convolucionales; Págs: 2-3.
[65] FUENTE: CABALLERO Z., Rafael; Master's Thesis: Use of Convolutional Codes for Collaborative Networks; Pág: 4.

Luego, para conocer la salida correspondiente al mensaje de entrada *u* = (1 0 1 1), se debe hacer la suma superpuesta del producto entrada por respuesta impulsiva, de la siguiente manera:

Entrada	**Respuesta Impulsiva**						
1	1 1	1 1	1 0	1 1			
0		0 0	0 0	0 0	0 0		
1			1 1	1 1	1 0	1 1	
1				1 1	1 1	1 0	1 1
Suma:	1 1	1 1	0 1	1 1	0 1	0 1	1 1

2.4.1.1 Representación del Codificador Convolucional

2.4.1.1.1 Vectores de Conexión o Polinomiales

[66]Una manera de representar al codificador es mediante polinomios. En este caso la representación se hace mediante *n* generadores polinomiales, uno para cada sumador. Cada polinomio es de grado *K* – 1 ó menos y describe la conexión de los registros de desplazamiento con los sumadores, de manera parecida a los vectores de conexión. El coeficiente de cada término del polinomio es 1 ó 0, según haya o no haya conexión con el sumador, respectivamente. Una vez más para el codificador de la Figura 2.5, la representación está dada por las Ecuaciones 2.3 y 2.4:

$$g_0 = 1 + x + x^2 + x^3 \qquad \textbf{(Ecuación 2.3)}$$

$$g_1 = 1 + x + x^3 \qquad \textbf{(Ecuación 2.4)}$$

El término de menor grado corresponde al registro de entrada. La secuencia de salida se obtiene haciendo, $v(x) = u(x)g_0(x)$ intercalado con $u(x)g_1(x)$. Siguiendo con el ejemplo de la Figura 2.5 la operación sería:

[66] FUENTE: ROQUE; SÁENZ; PEÑA; Comunicación de Datos: Códigos Convolucionales; Págs: 2-3.

$$u(x)g_0(x) = (1+x^2+x^3)(1+x+x^2+x^3) = 1+x+x^3+x^6$$
$$u(x)g_1(x) = (1+x^2+x^3)(1+x+x^3) = 1+x+x^2+x^3+x^4+x^5+x^6$$

$$u(x)g_0(x) = 1+1x+0x^2+1x^3+0x^4+0x^5+1x^6$$
$$u(x)g_1(x) = 1+1x+1x^2+1x^3+1x^4+1x^5+1x^6$$

$$v(x) = (1,1)+(1,1)x+(0,1)x^2+(1,1)x^3+(0,1)x^4+(0,1)x^5+(1,1)x^6$$
$$v = 1\ 1 \quad 1\ 1 \quad 0\ 1 \quad 1\ 1 \quad 0\ 1 \quad 0\ 1 \quad 1\ 1$$

[67]Generalmente estos polinomios se los obtienen a partir de una tabla y están dados en formato octal. La forma de leer el polinomio en este formato es:

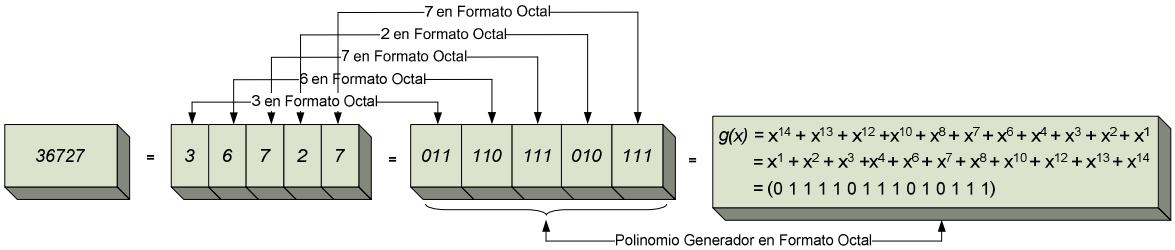


Figura 2.7. Representación de un Polinomio Generador en Formato Octal.

2.4.1.1.2 Diagrama de Estado

[68]El estado de un codificador convolucional de tasa de código k/*n* está representado por el contenido de los *m* registros de memoria. El conocimiento de este estado más el conocimiento del bit de entrada son suficientes para determinar la salida y el estado siguiente. Para el ejemplo ocupado existen ocho estados posibles, ya que $m = K - 1 = 3$ y cada círculo representa un estado. En el diagrama de estados, por convención, los estados de transición se indican con una línea continua cuando la entrada es un 0 y con una línea punteada cuando la entrada es un 1. Por cada transición se indica además la salida correspondiente. [69]El diagrama de estados no tiene al tiempo como una dimensión, es decir, con la observación del diagrama de estados no es posible determinar cuál es la secuencia de salida del codificador; y por lo tanto tiende a ser no intuitivo. En este caso, el diagrama de estados queda como se representa en la Figura 2.8.

[67] FUENTE: THE MATHWORKS; Communications Toolbox™ 4: User's Guide; Sección: 6, Pág: 4.
[68] FUENTE: ROQUE; SÁENZ; PEÑA; Comunicación de Datos: Códigos Convolucionales; Pág: 296.
[69] FUENTE: LANGTON, Charan; Intuitive Guide to Principles of Communications: Coding and Decoding with Convolutional Codes; Pág: 11.

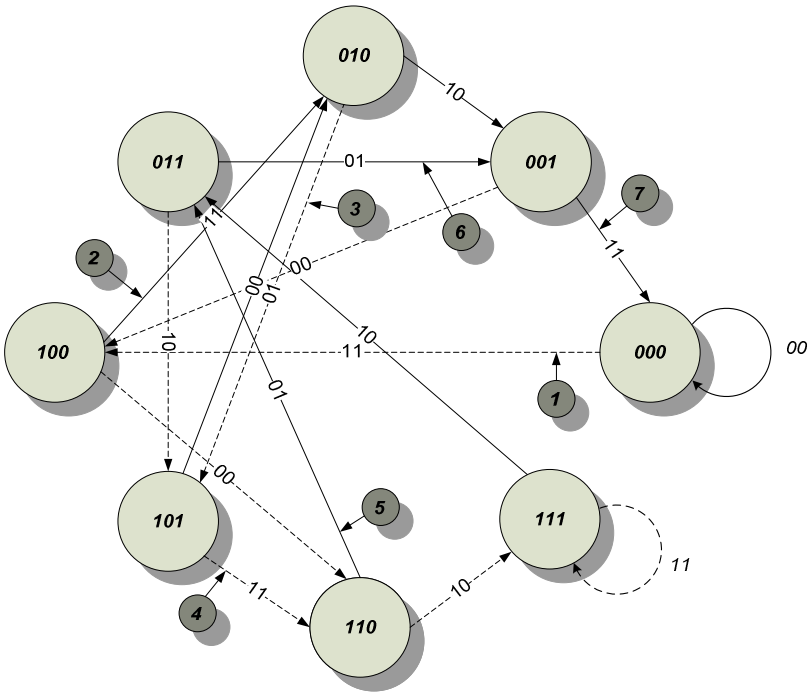


Figura 2.8. Diagrama de Estados del Codificador (2, 1, 3). [70]

2.4.1.1.3 Diagrama de Árbol

[71]Para darle una dimensión temporal al diagrama de estado, se recurre al diagrama de árbol, cuyo ejemplo para el codificador (2,1,3) se ve en la Figura 2.9. Por cada bit de entrada se manifiesta un movimiento hacia la derecha en el diagrama de árbol. Cada rama del árbol indica la salida del codificador. Este diagrama es construido a partir del diagrama de estados. Una vez construido el diagrama de árbol se puede armar el recorrido para determinar la secuencia de salida correspondiente a una determinada entrada.

Estando en un estado determinado, la regla es moverse hacia abajo cuando el bit de entrada es un 1, o moverse hacia arriba cuando el bit de entrada es un 0, siempre avanzando hacia la derecha. Si el mensaje de entrada es 1011 y el movimiento es dentro del árbol, según la regla descripta, se obtiene el camino resaltado en negro en la Figura 2.9, que da la secuencia de salida *v* = 11110111010111.

[70] FUENTE: LANGTON, Charan; Intuitive Guide to Principles of Communications: Coding and Decoding with Convolutional Codes; Pág: 11.
[71] FUENTE: ROQUE; SÁENZ; PEÑA; Comunicación de Datos: Códigos Convolucionales; Pág: 296.

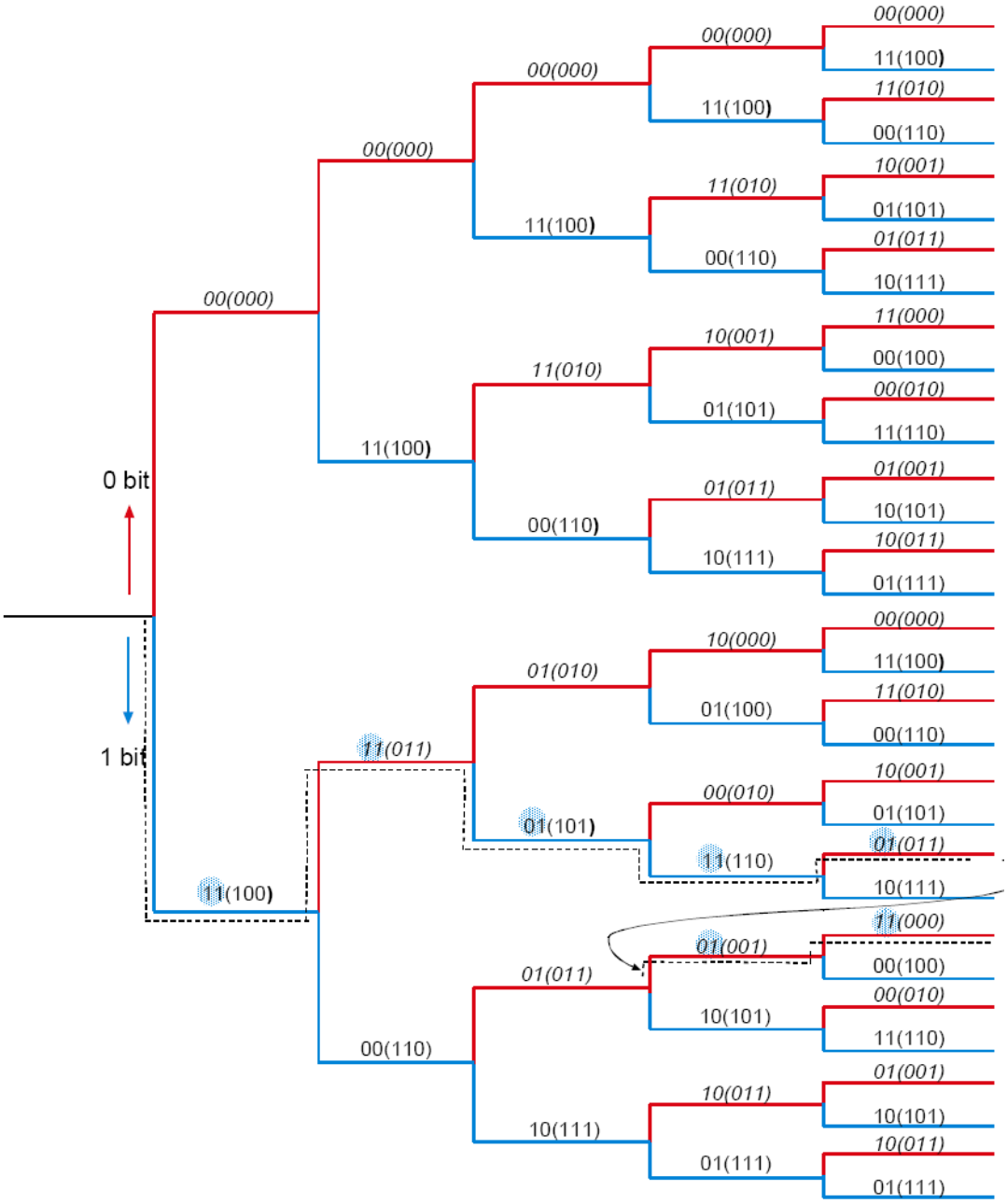


Figura 2.9. Diagrama de Estados del Codificador (2, 1, 3). [72]

2.4.1.1.4 Diagrama de Trellis

[73]El diagrama de Trellis es más complejo pero en general se lo prefiere sobre los diagramas de estado y de árbol ya que representan la secuencia lineal de tiempo de los acontecimientos. En el eje X está el tiempo discreto y todos los posibles

[72] FUENTE: LANGTON, Charan; Intuitive Guide to Principles of Communications: Coding and Decoding with Convolutional Codes; Pág: 13.
[73] FUENTE: LANGTON, Charan; Intuitive Guide to Principles of Communications: Coding and Decoding with Convolutional Codes; Pág: 13-14.

estados se muestran en el eje Y. Se puede mover horizontalmente a través del Trellis con el tiempo. Cada transición significa que nuevos bits han llegado.

El diagrama de Trellis se dibuja alineando todos los estados posibles 2^m en el eje vertical. Luego se conecta cada estado al próximo estado mediante las palabras código permitidas para ese estado. Sólo hay dos opciones posibles en cada estado. Estas son determinadas por la llegada de un 0 o un 1. Las flechas indican el bit de entrada, y los bits de salida se muestran en paréntesis. Las flechas hacia arriba representan un bit 0 y las de bajada representan un bit 1. El diagrama de Trellis es único para cada código; siempre se comienza en el estado 000.

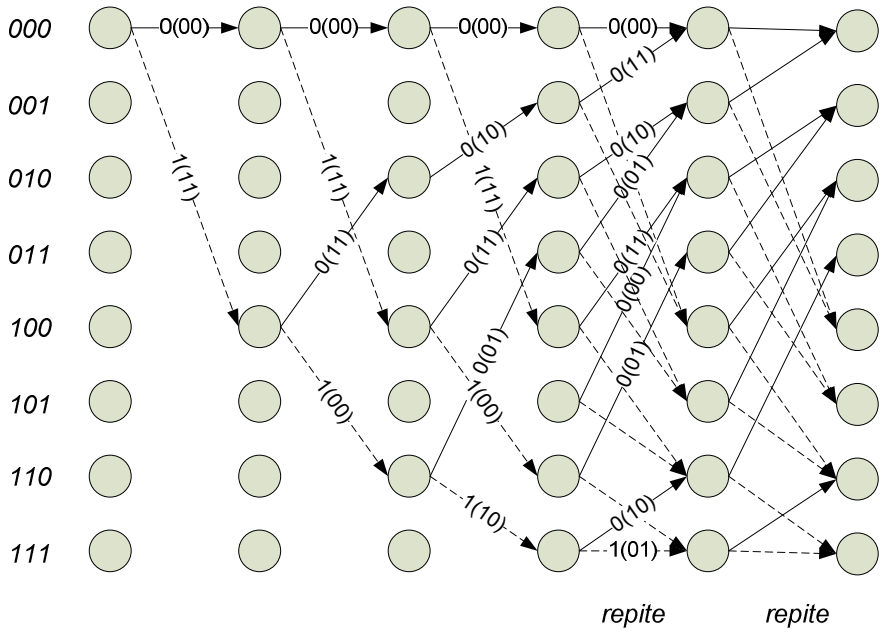


Figura 2.10. Diagrama de Trellis del Codificador (2, 1, 3). [74]

En la Figura 2.11, los bits de entrada se muestran en la parte superior. Para la codificación se sube cuando es un 0 y se baja cuando se tiene un bit 1. El camino seguido por los bits del ejemplo secuencia (1011000) se muestra por la línea con flechas. El diagrama de Trellis da exactamente la misma salida de secuencia como los otros tres métodos: respuesta impulso, diagramas de estado y árbol.

[74] FUENTE: LANGTON, Charan; Intuitive Guide to Principles of Communications: Coding and Decoding with Convolutional Codes; Pág: 14.

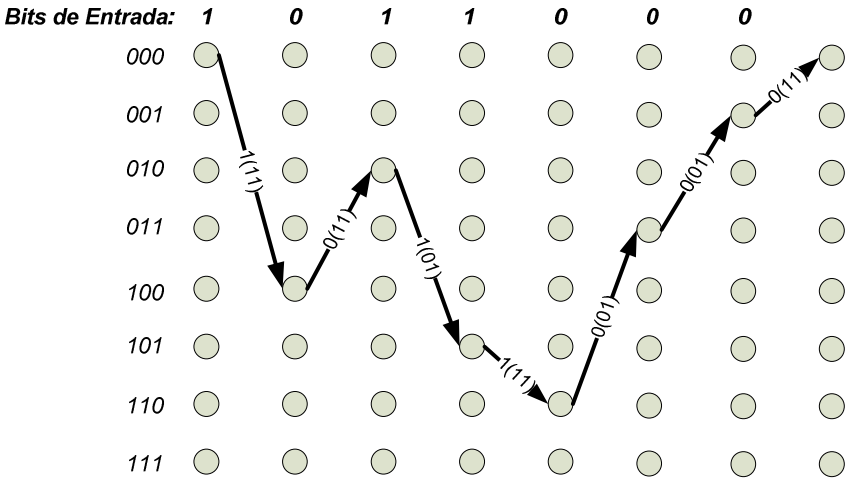


Figura 2.11. Secuencia Codificada, Bits de Entrada 1011000, Bits de Salida 11011111010111. [75]

De esta manera se determina los estados en los que se encuentra el bit y cuales son los próximos estados que va a adquirir dicho bit. Entre las múltiples herramientas que tiene Matlab se encuentra una que permite la conversión directa del polinomio generador al Trellis, esta es *poly2trellis*. Dicha función, con la sintaxis *poly2trellis (K, [g_0 g_1])*, necesita como parámetros de entrada para el procesamiento de los datos el *constraint length K* y los polinomios generadores g_0 y g_1 del código *G* en formato octal, arrojando como resultado de manera resumida únicamente los resultados que van a ser utilizados en la decodificación.

```
>> trellis = poly2trellis (4,[17 15])

trellis =

     numInputSymbols: 2
    numOutputSymbols: 4
           numStates: 8
          nextStates: [8x2 double]
             outputs: [8x2 double]
```

Figura 2.12. Ejecución de Prueba para la Visualización del Trellis del Codificador (2, 1, 3).

2.4.1.1.5 Terminación del Código

[75] FUENTE: LANGTON, Charan; Intuitive Guide to Principles of Communications: Coding and Decoding with Convolutional Codes; Pág: 15.

[76]En teoría, las secuencias de código de los códigos convolucionales son de longitud infinita, pero para aplicaciones prácticas se suelen emplear secuencias finitas. Existen tres métodos para obtener las secuencias finitas de código:

- ***Truncamiento:*** se termina de codificar después de un cierto número de bits sin utilizar ningún esfuerzo adicional. Esto lleva a tener probabilidades de error altas para los últimos bits en una secuencia.
- ***Terminación:*** se agregan algunos bits de cola para la secuencia de código en orden, con el fin de garantizar un estado final predefinido del codificador, lo que lleva a bajar las probabilidades de error para los bits finales en una secuencia.
- ***Tail-Biting:*** se escoge un estado inicial que asegura que los estados inicial y final sean los mismos; esto conduce a tener igual probabilidad para los últimos bits de una secuencia. El tail-biting aumenta la complejidad de decodificación, y para la terminación es requerida redundancia adicional.

Suponiendo que se desea codificar exactamente *u* bloques de información. El procedimiento más fácil para obtener una secuencia de código finito es el truncamiento de código. Con este método se detiene el codificador después de *u* bloques de información y también trunca la secuencia de código después de *u* bloques de código. Sin embargo, este enfoque sencillo conduce a una degradación sustancial de la protección contra errores para los últimos bits de información codificada, ya que los últimos bits de información codificados influyen únicamente en un pequeño número de bits del código. Por ejemplo, los últimos *k* bits de información determinan los últimos *n* bits de código. Por lo tanto, la terminación o tail-biting se suele preferir sobre el truncamiento.

Para obtener una secuencia de código terminada, se empieza la codificación con todos los estados del codificador en cero, asegurándose que después del proceso de codificación, todos los elementos de la memoria estén en cero otra vez.

[76] FUENTE: NEUBAUER André; FREUDENBERGER Jürgen; KÜHN Volker; Coding Theory: Algorithms, Architectures, and Applications; Págs: 104-106.

2.4.2 EL DECODIFICADOR CONVOLUCIONAL

[77]Hay varias maneras diferentes para decodificar los códigos convolucionales. Estos se agrupan en dos categorías básicas; ambos métodos representan dos enfoques diferentes a la misma idea básica de decodificación:

2.4.2.1 Decodificación Secuencial

[78]La decodificación secuencial fue uno de los primeros métodos propuestos para decodificar convolucionalmente una cadena de bits codificados. Se propuso por primera vez por Wozencraft y más tarde una versión mejorada fue propuesta por Fano. Este tipo de decodificación no se explica ya que este proyecto de titulación únicamente utiliza el Algoritmo de Viterbi para la decodificación pero se puede encontrar mayores referencias bibliográficas del tema en el Capítulo XII del Libro "Error Control Coding: Fundamentals and Applications" de los autores Shu Lin y Daniel Costello.

2.4.2.2 Decodificación de Máxima Probabilidad

[79]La *Decodificación de Máxima Probabilidad* se da cuando un decodificador para control de errores encuentra, dentro del conjunto de todas las palabras posibles que pudieran transmitirse, la palabra código que más se asemeje a la información recibida, es decir, la palabra código que minimice la probabilidad de error del decodificador.

La idea básica del decodificador convolucional es: recibida una palabra, se la decodifica como la palabra código de máxima probabilidad (palabra código con la mínima distancia de Hamming) con respecto a la palabra recibida.

---

[77] FUENTE: LANGTON, Charan; Intuitive Guide to Principles of Communications: Coding and Decoding with Convolutional Codes; Pág: 15.
[78] FUENTE: LANGTON, Charan; Intuitive Guide to Principles of Communications: Coding and Decoding with Convolutional Codes; Pág: 16.
[79] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Págs: 50-51.

2.4.2.2.1 Algoritmo de Viterbi

[80]Aproximadamente hace cuatro décadas atrás, el decodificado de máxima probabilidad no parecía ser un esquema de decodificación preferido para códigos convolucionales por la gran carga computacional que implicaba el tener que hacer búsquedas en el espacio completo de un código. Pero, en 1967 Viterbi introdujo un algoritmo de decodificación de máxima probabilidad para códigos convolucionales práctico, para códigos con una *constraint length* baja. Este algoritmo hace uso de la estructura altamente repetitiva del árbol del código para reducir la cantidad de operaciones requeridas para hacer búsquedas en todo el espacio del código.

El algoritmo de Viterbi elige el camino en el Trellis que difiere en el menor número posible de bits de la secuencia recibida. Para su implementación se define la distancia de un camino particular (métrica) como el número de bits en que difieren la secuencia producida por ese camino y la secuencia recibida. En la decodificación se compara para cada estado los dos posibles caminos que llegan al nodo y se retiene el camino que tenga una distancia menor, descartándose el otro camino. Si se tienen distancias iguales se elige aleatoriamente uno. Los caminos seleccionados son llamados sobrevivientes (*survivors*).

Considérese que una secuencia de información $u = (u_0, u_1, \ldots, u_{kL-1})$ de longitud kL es codificada en una palabra código binaria $v = (v_0, v_1, \ldots, v_{N-1})$ de longitud $N = n(L + m)$ y que una secuencia $r = (r_0, r_1, \ldots, r_{N-1})$ es recibida a la salida de un canal. Un decodificador para control de errores debe producir un estimado v' de la palabra código v basado en la secuencia recibida r. Un *Decodificador de Máxima Probabilidad* es aquel que elige v' como la palabra de código v que maximiza la función *log-likelihood* dada por la Ecuación 2.5:

$$logP(\mathbf{r}|\mathbf{v}) = \sum_{i=0}^{L+m-1} log\, P(\mathbf{r}_i|\mathbf{v}_i) = \sum_{i=0}^{N-1} log\, P(r_i|v_i) \qquad \textbf{(Ecuación 2.5)}$$

[80] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Págs: 51-54.

Donde $P(r_i|v_i)$ es la probabilidad de que el símbolo r_i sea recibido a la salida del canal, dado que el símbolo v_i fue enviado a través de dicho canal. Este tipo de probabilidades, conocidas como *Probabilidades de Transición*, pueden calcularse a partir de un conocimiento de las señales utilizadas, de la distribución de probabilidad del ruido y del umbral de cuantización de la salida del demodulador.

2.4.2.2.2 Métricas

[81]La función log-likelihood, log $P(\mathbf{r}|\mathbf{v})$ se conoce como la *Métrica Asociada con la Trayectoria* **v**, y se denota como $M(\mathbf{r}|\mathbf{v})$. Los términos log $P(\mathbf{r}_i|\mathbf{v}_i)$ en la sumatoria de la Ecuación 2.5 se conocen como *Métricas de Rama* y se denotan como $M(\mathbf{r}_i|\mathbf{v}_i)$, mientras que los términos log $P(r_i|v_i)$ se conocen como *Métricas de Bit* y se denotan como $M(r_i|v_i)$. De este modo, la ecuación anterior se puede reescribir como la Ecuación 2.6:

$$M(\mathbf{r}|\mathbf{v}) = \sum_{i=0}^{L+m-1} M(\mathbf{r}_i|\mathbf{v}_i) = \sum_{i=0}^{N-1} M(r_i|v_i) \qquad \textbf{(Ecuación 2.6)}$$

Una *Métrica de Trayectoria Parcial* para las primeras *j* ramas de una trayectoria puede escribirse como la Ecuación 2.7:

$$M([\mathbf{r}|\mathbf{v}]_j) = \sum_{i=0}^{j-1} M(\mathbf{r}_i|\mathbf{v}_i) = \sum_{i=0}^{nj-1} M(r_i|v_i) \qquad \textbf{(Ecuación 2.7)}$$

De esta manera, finalmente, el algoritmo de Viterbi se puede enunciar considerando, recibida una secuencia **r** de un canal discreto encuentra la trayectoria de mayor $M(\mathbf{r}|\mathbf{v})$ a través del diagrama de Trellis del código que se considere. El algoritmo de Viterbi por tanto se estructuraría como:

- *Paso 1:* En el diagrama de Trellis, comenzando en la unidad de tiempo *j* = *m,* calcular la métrica de trayectoria parcial de la única trayectoria que entra

[81] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Págs: 54-55.

a cada estado y que comienza en el nivel 0. Almacenar la trayectoria (llamada *sobreviviente*) junto con su métrica para cada estado.

- *Paso 2:* Incrementar *j* en 1. Calcular la métrica parcial para todas las trayectorias que entran a un estado añadiendo la métrica de la rama que entra a ese estado a la métrica de la trayectoria sobreviviente en la unidad de tiempo anterior que se conecta a esa rama. Para cada uno de los estados, almacenar la trayectoria con la métrica más alta (trayectoria sobreviviente) junto con su métrica, y eliminar todas las otras trayectorias.
- *Paso 3:* Si $j < L + m$, repetir el Paso 2. De otra manera, parar. La trayectoria buscada será la trayectoria sobreviviente del nivel 0 al nivel $L + m$ a través del diagrama de Trellis considerado. Generalmente, es más conveniente y sencillo utilizar enteros positivos como métricas en vez de las métricas de bit reales. Una métrica de bit $M(r_i \mid v_i)$ puede reemplazarse por la siguiente ecuación sin afectar el desempeño del algoritmo de Viterbi. La constante c_1 puede ser cualquier número real y la constante c_2 puede ser cualquier número real positivo.

$$M(r_i|v_i) = c_2\left[logP(r_i|v_i) + c_1\right] \qquad \textbf{(Ecuación 2.8)}$$

[82]El método de decodificación de Viterbi es un método óptimo para códigos convolucionales; sin embargo su desempeño depende de la calidad del canal, y el esfuerzo de decodificado crece exponencialmente con la *constraint length* de cada código. A pesar de esto, para códigos convolucionales con *constraint lengths* bajas la decodificación de Viterbi provee suficientes ganancias de codificación en muchas aplicaciones, y permite que la implementación del decodificador no sea tan compleja. [83]Se debe entender por ganancia de codificación a la reducción en decibeles que presenta la relación E_b/N_o para alcanzar una BER determinada en un sistema codificado en comparación con uno no codificado, empleando un mismo esquema de modulación.

[82] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Págs: 58-59.
[83] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Pág: 20.

Para finalizar el capítulo, el Espacio de Código 2.3 elabora una gráfica del BER vs. E_b/N_o, con modulación BPSK a través de un canal AWGN con codificación convolucional siguiendo el esquema que se muestra en la Figura 2.13. La demodulación, al igual que en los casos anteriores toma decisiones duras. Se ha hecho uso de las funciones ***convenc*** y ***vitdec*** que se encuentran dentro del toolbox de comunicaciones de Matlab para codificar y decodificar la secuencia de bits, respectivamente. La función *convenc(x,t)* toma como entrada la secuencia de bits de información *x*, y la configuración del Trellis que ha sido almacenada en la variable *t*. Mientras que la función *vitdec(demodula,t,tb,'trunc','hard')*, en primera instancia requiere de la secuencia de bits demodulados (*demodula*), luego la configuración del Trellis (*t*), el *traceback length* (*tb),* modo en el cual se ha terminado el Trellis que para el particular es por truncamiento ("*trunc*") y por último se específica si en la demodulación se utilizaron decisiones duras o suaves y que será definido para este caso por "*hard*".

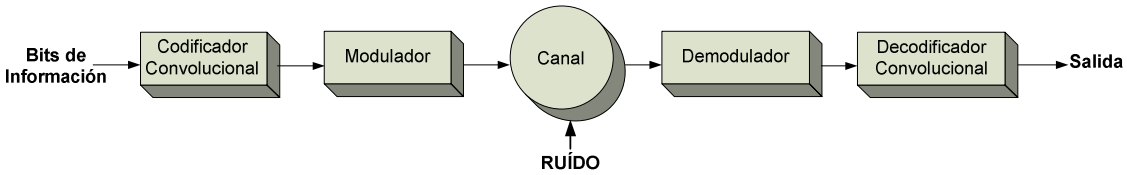


Figura 2.13. Esquema de un Código Convolucional sobre canal AWGN.

```
clear all

G = [1 1 1;1 0 1]; % Código Generador por default
[n,K] = size(G); % Cálculo tamaño Código Generador para obtener constraint length
cad_G = num2str(G); % Conversión código generador a cadena de caracteres
gen = str2num(dec2base(bin2dec(cad_G),8)); % Conversión cadena de caracteres binaria a
octal
t = poly2trellis(K,[gen(1,1) gen(2,1)]); % Definición del trellis.
load ('x.txt'); % Carga el archivo de texto con los bits generados
x = x; % Conserva el mismo nombre de la variable
% Si se desea generar una nueva secuencia de bits de información se debe:
% comentar las dos lineas anteriores de programa e introducir: x = randint(1,40000);
codificador_conv = convenc(x,t); % Codificación Convolucional de la Trama
M = 2; % Número de niveles o estados de la señal modulada
n_bits_modulacion = log2(M); % Número de bits por símbolo
y = modulate(modem.pskmod(M,pi),codificador_conv); % Bits Modulados (BPSK)
EbNodB = 0:11; % Variación de EbNo (en decibelios)
SNRdB = EbNodB + 3 + 10*log10(n_bits_modulacion); % Cálculo de la SNR (en decibelios)
w = 0; % Inicializa w, que permite realizar la variación de Eb/No para poder graficar
k = length(SNRdB);
for p1 = 1:k
    y_canal = awgn(real(y),SNRdB(1,p1)); % Canal AWGN
    demodula = demodulate(modem.pskdemod(M,pi),y_canal); % Demodulador de Decisiones
Duras
    tb = 2; % Traceback length para la decodificación
    decodificador_conv = vitdec(demodula,t,tb,'trunc','hard'); % Decodificación por
Viterbi
```

```
    [numero_de_errores, bit_error_rate] = biterr(x,decodificador_conv); % Cálculo del
Bit Error Rate
    w = w+1;
    BER_Conv(w) = bit_error_rate; % Vector Bit Error Rate
end

BER_Conv
semilogy(EbNodB,BER_Conv); % Grafica en escala semilogarítmica BER Vs Eb/No
xlabel('Eb/No en [dB]'); ylabel('Bit Error Rate (BER)'); grid on; % Título del Eje de
las X e Y

title('RENDIMIENTO DE UN CÓDIGO CONVOLUCIONAL CON CANAL AWGN'); % Título del Gráfico
legend('Código Convolucional con Canal AWGN'); % Leyenda para mostrar los detalles de la
señal
```

Espacio de Código 2.3. Rendimiento de un Código Convolucional sobre canal AWGN.

Eb/No (en dB)	**0**	**1**	**2**	**3**	**4**	**5**
BER (con Código Convolucional)	0.075179	0.039077	0.017601	0.0051753	0.0023251	0.00027501
Eb/No (en dB)	**6**	**7**	**8**	**9**	**10**	**11**
BER (con Código Convolucional)	0.00010001	0	0	0	0	0

Tabla 2.5. Resultados de la Simulación de un Código Convolucional sobre canal AWGN.

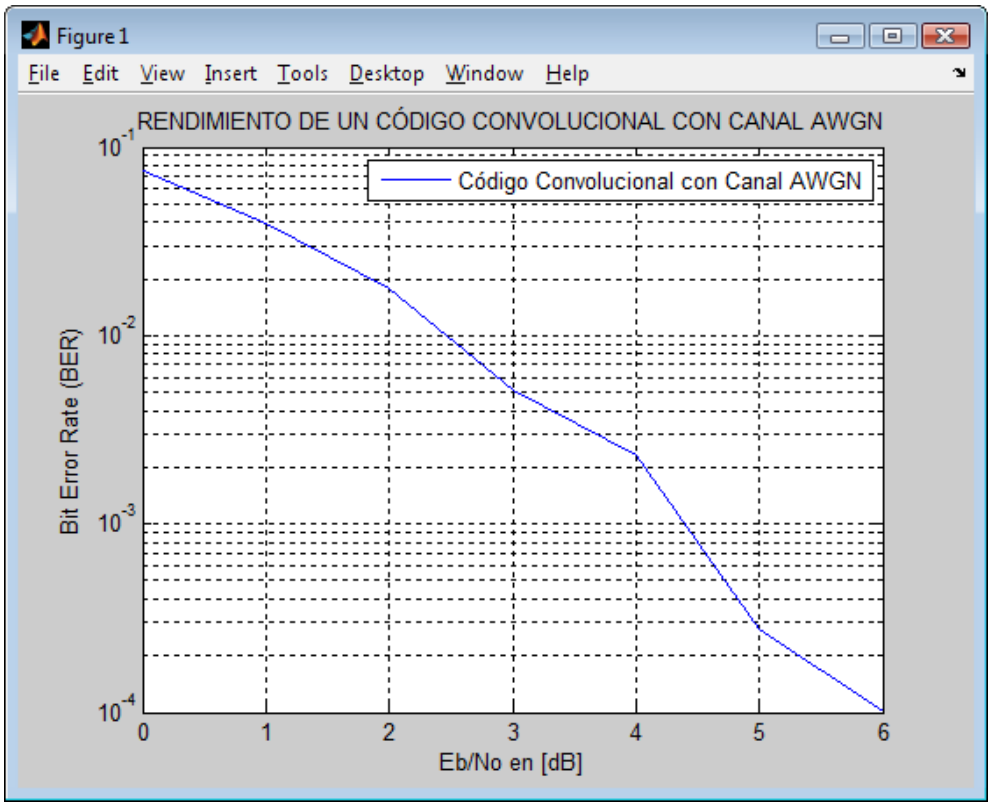


Figura 2.14. Rendimiento de un Código Convolucional sobre canal AWGN.

El Espacio de Código 2.4 permite visualizar el rendimiento de una misma secuencia de bits a través de un canal AWGN sin codificar generado en el Espacio de Código 1.12, a la misma que posteriormente se le ha agregado el control de errores mediante un código de bloque y un código convolucional, por separado, con el objeto de verificar la mejora en el rendimiento de una misma secuencia transmitida cuando es utilizada la detección y corrección de errores.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%% AWGN SIN CODIFICAR %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load ('x.txt'); % Carga el archivo de texto con los bits generados
x = x; % Conserva el mismo nombre de la variable
% Si se desea generar una nueva secuencia de bits de información se debe:
% comentar las dos lineas anteriores de programa e introducir: x = randint(1,40000);
M = 2; % Número de niveles o estados de la señal modulada
n_bits_modulacion = log2(M); % Número de bits por símbolo
y = modulate(modem.pskmod(M,pi),x); % Bits Modulados (BPSK)
EbNodB = 0:11; % Variación de EbNo (en decibelios)
SNRdB = EbNodB + 3 + 10*log10(n_bits_modulacion); % Cálculo de la SNR (en decibelios)
w = 0; % Inicializa w, que permite realizar la variación de Eb/No para poder graficar
k = length(SNRdB);
for p1 = 1:k
    y_canal = awgn(real(y),SNRdB(1,p1)); % Canal AWGN
    demodu = demodulate(modem.pskdemod(M,pi),y_canal); % Demodulador de Decisiones Duras
    [numero_de_errores, bit_error_rate] = biterr(x,demodu); % Cálculo del Bit Error Rate
    w = w+1;
    BER_AWGN(w) = bit_error_rate; % Vector Bit Error Rate
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%% CODIGO DE BLOQUE  %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

m = 4; % Longitud de la redundancia del código de bloque
n = 2^m-1; k = n-m; % Longitud de la palabra código y mensaje, respectivamente.
codificador_bloque = encode(x,n,k,'hamming/binary'); % Codificación de Bloque (Hamming)
de la Trama
y = modulate(modem.pskmod(M,pi),codificador_bloque); % Bits Modulados (BPSK)
w = 0; % Inicializa w, que permite realizar la variación de Eb/No para poder graficar
kk = length(SNRdB);
for p1 = 1:kk
    y_canal = awgn(real(y),SNRdB(1,p1)); % Canal AWGN
    demodul = demodulate(modem.pskdemod(M,pi),y_canal); % Demodulador de Decisiones
Duras
    decodificador_bloque = decode(demodul,n,k,'hamming/binary'); % Decodificación del
Código de Hamming
    [numero_de_errores, bit_error_rate] = biterr(x,decodificador_bloque(1:length(x))); %
Cálculo del Bit Error Rate
    w = w+1;
    BER_Bloque(w) = bit_error_rate; % Vector Bit Error Rate
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%% CODIGO CONVOLUCIONAL  %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

G = [1 1 1;1 0 1]; % Código Generador por default
[n,K] = size(G); % Cálculo tamaño Código Generador para obtener constraint length
cad_G = num2str(G); % Conversión código generador a cadena de caracteres
gen = str2num(dec2base(bin2dec(cad_G),8)); % Conversión cadena de caracteres binaria a
octal
```

```
t = poly2trellis(K,[gen(1,1) gen(2,1)]); % Definición del trellis.
codificador_conv = convenc(x,t); % Codificación Convolucional de la Trama
y = modulate(modem.pskmod(M,pi),codificador_conv); % Bits Modulados (BPSK)
w = 0; % Inicializa w, que permite realizar la variación de Eb/No para poder graficar
k = length(SNRdB);
for p1 = 1:k
    y_canal = awgn(real(y),SNRdB(1,p1)); % Canal AWGN
    demodula = demodulate(modem.pskdemod(M,pi),y_canal); % Demodulador de Decisiones
Duras
    tb = 2; % Traceback length para la decodificación
    decodificador_conv = vitdec(demodula,t,tb,'trunc','hard'); % Decodificación por
Viterbi
    [numero_de_errores, bit_error_rate] = biterr(x,decodificador_conv); % Cálculo del
Bit Error Rate
    w = w+1;
    BER_Conv(w) = bit_error_rate; % Vector Bit Error Rate
end
BER_AWGN
BER_Bloque
BER_Conv
semilogy(EbNodB,BER_AWGN,'b-x',EbNodB,BER_Bloque,'g-o',EbNodB,BER_Conv,'r-d'); % Grafica
en escala semilogarítmica BER Vs Eb/No
xlabel('Eb/No en [dB]'); ylabel('Bit Error Rate (BER)'); grid on; hold on; % Título del
Eje de las X e Y
title('COMPARACIÓN DE RENDIMIENTO'); % Título del Gráfico
legend('Canal AWGN sin Codificar','Código de Bloque con Canal AWGN','Código
Convolucional con Canal AWGN'); hold off; % Leyenda para mostrar detalles de las señales
```

Espacio de Código 2.4. Comparación de Rendimiento de un Canal AWGN sin Codificar, un Código de Bloque y un Código Convolucional sobre canal AWGN.

Eb/No (en dB)	**0**	**1**	**2**	**3**	**4**	**5**
BER (Sin Codificar)	0.079079	0.056903	0.036152	0.023051	0.012876	0.0063253
BER (con Código de Bloque)	0.073529	0.042927	0.021651	0.0099255	0.0033002	0.00077504
BER (con codigo Convolucional)	0.075179	0.039077	0.017601	0.0051753	0.0023251	0.00027501
Eb/No (en dB)	**6**	**7**	**8**	**9**	**10**	**11**
BER (Sin Codificar)	0.0023001	0.00082504	0.00012501	5.0003e-005	0	0
BER (con Código de Bloque)	5.0003e-005	5.0003e-005	0	0	0	0
BER (Convolucional)	0.00010001	0	0	0	0	0

Tabla 2.6. Resultados de la Comparación de Rendimiento de un Canal AWGN sin Codificar, un Código de Bloque y un Código Convolucional sobre canal AWGN.

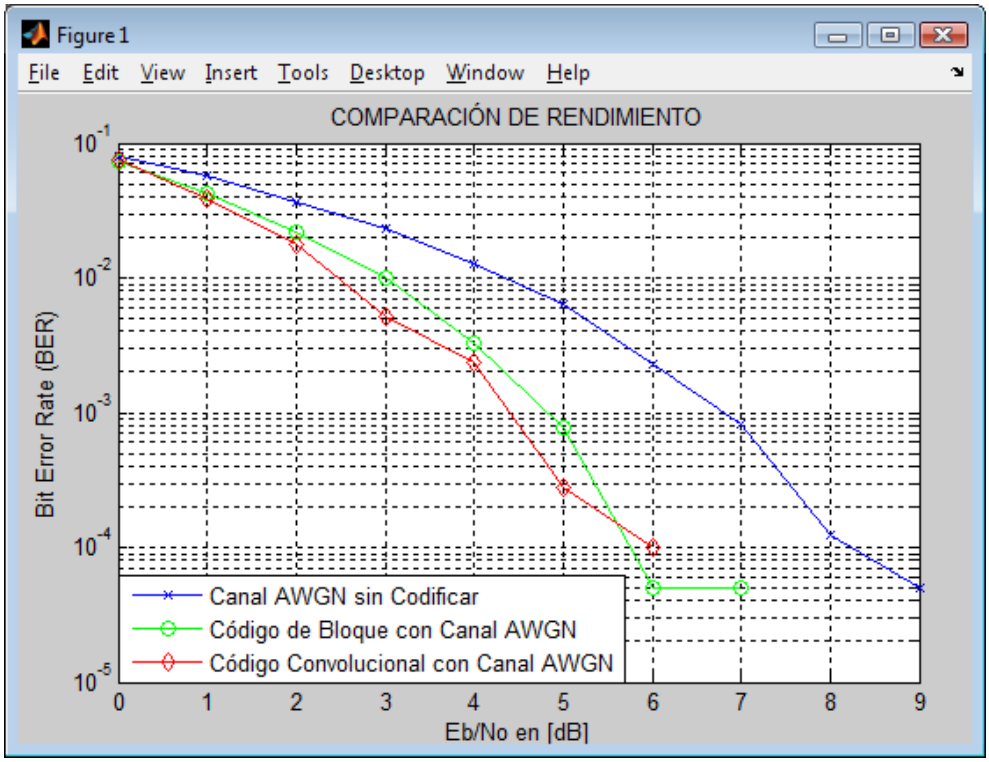


Figura 2.15. Comparación de Rendimiento de un Canal AWGN sin Codificar, un Código de Bloque y un Código Convolucional sobre canal AWGN.

Como resultado de la simulación, se puede observar claramente en la Figura 2.15 como se produce una notable reducción de los bits erróneos transmitidos conforme se utiliza una técnica diferente de corrección de errores; es así que para el caso del canal AWGN que no utiliza corrección de errores el BER se reduce a 0 cuando la E_b/N_o es de 10 dB, mientras que para el caso de los códigos de bloque es de 8 dB y para los códigos convolucionales este valor de E_b/N_o es 0 a partir de los 7 dB. Con esto queda demostrado como es necesaria una menor cantidad de potencia para transmitir una misma secuencia de información si se aplica una técnica que detecta y corrige los errores; esta reducción de potencia en un sistema físico y comercializable se ve reflejada en un menor costo del equipo.

2.5 APLICACIONES DE LOS TURBO CÓDIGOS

2.5.1 TURBO CÓDIGOS UTILIZADOS EN SISTEMAS MÓVILES 3G

[84]Los estándares de 3G, UMTS y cdma2000, satisfacen los requerimientos definidos por la ITU-R para los sistemas de radio móviles 3G. Por el desarrollo y popularidad que han adquirido los servicios inalámbricos, el número de usuarios móviles se ha incrementado en los últimos años, lo que conlleva a la necesidad de hacer un uso lo más eficientemente posible del espectro disponible. El tráfico de multimedia requiere de tasas de transmisión y ancho de banda mucho mayores, que los requeridos para transmitir voz, además de que la información multimedia es más sensible a errores producidos durante su transmisión por el canal, por lo que, para servicios de transmisión multimedia en tiempo real, una transmisión rápida y libre de errores se convierte en una prioridad.

La ganancia de codificación extra que ofrecen los Turbo Códigos, permite la posibilidad de conseguir una transmisión que sea tanto rápida como confiable. La ganancia de codificación de este esquema FEC también puede utilizarse en los dispositivos inalámbricos para reducir la fuerza de sus señales, lo que permite que más dispositivos puedan compartir el mismo espectro de frecuencia al reducirse la interferencia *inter-dispositivo*. Además, los Turbo Códigos ofrecen la posibilidad de enviar la misma cantidad de información, pero empleando únicamente la mitad del ancho de banda. Por estas razones los sistemas 3G adoptaron a este esquema de codificación para control de errores como parte de su estándar.

Ambos estándares, en la Capa Física incluyen tanto a los Códigos Convolucionales, en canales de voz y control; como a los Turbo Códigos en canales de transferencia de datos, aunque también se considera su uso en algunos canales de control. Los Turbo Códigos no hayan sido considerados para los canales de voz, debido al ligero retardo que presenta el Turbo Decodificador al emplear un proceso de decodificación iterativa. Sin embargo, este retardo no representa problema para el caso de transmisiones de datos; por el contrario, este

[84] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Págs: 119-121.

esquema de codificación resultó ser una excelente opción para la transmisión de contenido multimedia, pues este tipo de transmisión, requiere de tasas de transmisión elevadas y de un buen nivel de confiabilidad (BER baja).

2.5.1.1 El Turbo Código Utilizado en UMTS

2.5.1.1.1 Turbo Codificador UMTS

[85]El Turbo Codificador empleado por UMTS, está formado una estructura PCCC, de dos codificadores RSC, como el mostrado en la Figura 2.16, y su matriz de función de transferencia es: $G(D) = \left[1 \quad \frac{1+D+D^3}{1+D^2+D^3}\right]$.

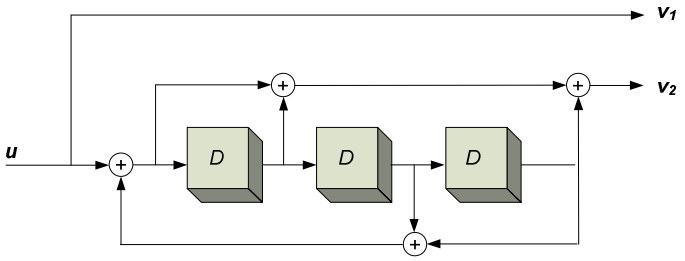


Figura 2.16. Codificador RSC del Turbo Codificador UMTS.[86]

La estructura del Turbo Codificador UMTS, con tasa *r* = 1/3, es por tanto:

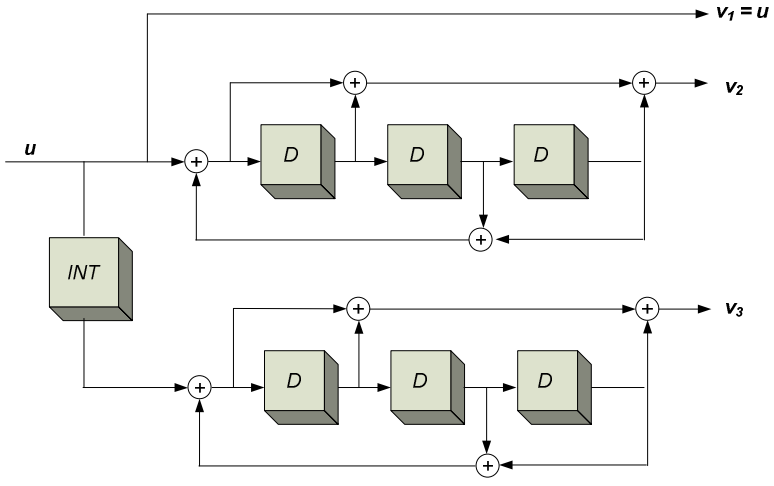


Figura 2.17. Turbo Codificador UMTS. [87]

[85] FUENTE: European Telecommunications Standards Institute (ETSI); Universal Mobile Telecommunications System (UMTS): Multiplexing and Channel Coding (FDD) – Release 4; Págs: 15-16.
[86] FUENTE: European Telecommunications Standards Institute (ETSI); Universal Mobile Telecommunications System (UMTS): Multiplexing and Channel Coding (FDD) – Release 4; Págs: 15-16.

2.5.1.1.2 Interleaver UMTS

[88]El tamaño del bloque de información *u*, y por consiguiente del Interleaver, puede variar desde 40 bits hasta 5114 bits. El Interleaver consiste en una matriz que puede tener 5, 10 ó 20 filas, y entre 8 y 256 columnas, dependiendo del tamaño del bloque de información *u*. La información es vaciada en el Interleaver fila por fila, con el primer bit de información colocado en la esquina superior izquierda de la matriz. Una vez que ha sido vaciado todo el bloque de información en la matriz, se efectúan una serie de permutaciones dentro de cada fila de la matriz, con base en un algoritmo complejo. Posteriormente, se efectúan una serie de permutaciones "inter-fila" (entre filas), de forma tal que se cambie el orden de las filas, pero sin alterar el orden de los elementos dentro de cada fila. Finalmente, la información es leída del Interleaver columna por columna, comenzando por el bit de información ubicado en la esquina superior izquierda de la matriz transformada.

2.5.1.1.3 Turbo Decodificador UMTS

La Turbo Decodificación en UMTS, es idéntico al proceso de decodificación iterativa de la Figura 3.44. En este estándar de 3G se decidió implementar el algoritmo Log-MAP para el proceso de Turbo Decodificación.

2.5.1.2 El Turbo Código Utilizado en CDMA-2000

2.5.1.2.1 Turbo Codificador CDMA-2000

[89]El Turbo Codificador empleado por cdma2000, está formado una estructura PCCC, de dos codificadores RSC, como el mostrado en la Figura 2.18, y su matriz de función de transferencia es: $G(D) = \left[1 \quad \frac{1+D+D^3}{1+D^2+D^3} \quad \frac{1+D+D^2+D^3}{1+D^2+D^3} \right]$.

---

[87] FUENTE: European Telecommunications Standards Institute (ETSI); Universal Mobile Telecommunications System (UMTS): Multiplexing and Channel Coding (FDD) – Release 4; Págs: 15-16.
[88] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Pág: 124.
[89] FUENTE: Third Generation Partnership Project 2 (3GPP2); Physical Layer Standard for cdma2000 Spread Spectrum Systems - Release C; Pág: 88.

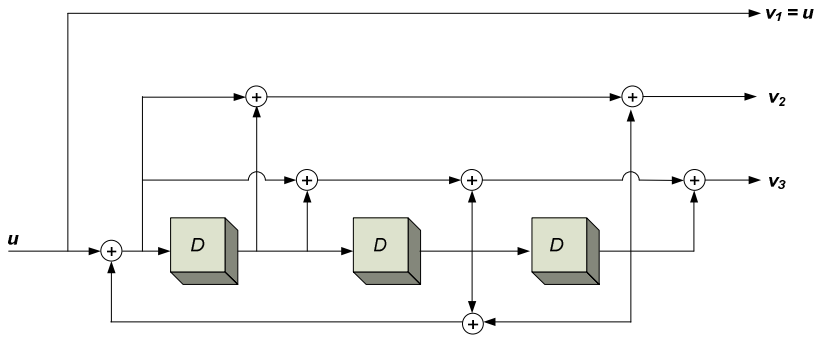


Figura 2.18. Codificador RSC empleado por el Turbo Codificador cdma2000. [90]

[91]Cada uno de los codificadores RSC que forman al Turbo Codificador cdma2000 posee una tasa *r* = 1/3, solo que la parte sistemática del codificador RSC inferior se ha eliminado para formar al Turbo Codificador. De este modo, por cada bit de información que ingrese al Turbo Codificador, este producirá un total de cinco bits codificados (el bit sistemático, y cuatro bits de paridad), dando por resultado una tasa del Turbo Codificador igual a *r* = 1/5.

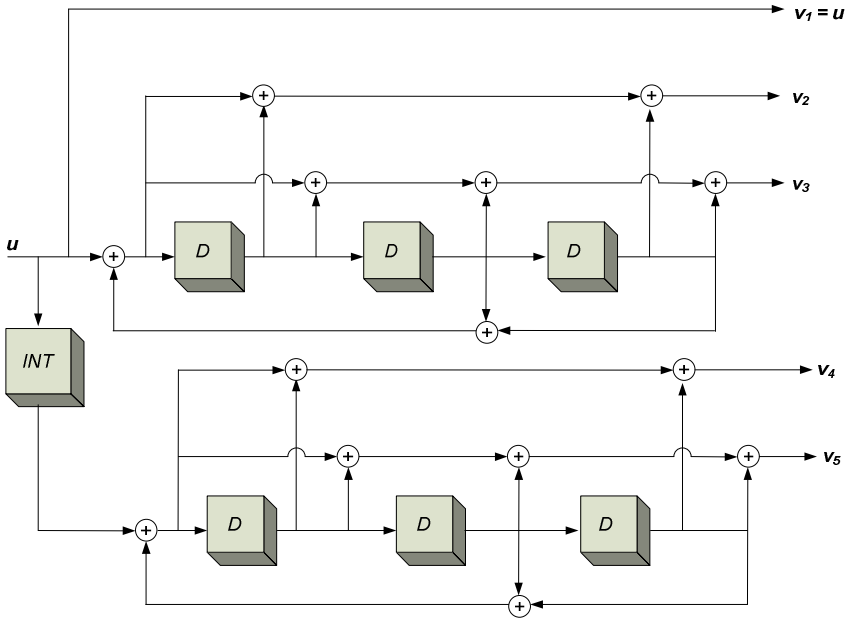


Figura 2.19. Turbo Codificador cdma2000. [92]

[90] FUENTE: Third Generation Partnership Project 2 (3GPP2); Physical Layer Standard for cdma2000 Spread Spectrum Systems - Release C; Pág: 90.
[91] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Pág: 132.
[92] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Pág: 131.

Una tasa de *r* = 1/5 resulta indeseable para muchas aplicaciones, por el incremento en el ancho de banda de transmisión. Por ello, el Turbo Codificador cdma2000 incluye *puncturing* para lograr tasas de código más altas. Mediante este mecanismo, el Turbo Codificador puede ofrecer tasas de código *r* de 1/2, 1/3 y 1/4. Por ejemplo, para lograr una tasa *r* = 1/3, el Turbo Codificador no toma en cuenta la segunda salida de paridad de cada codificador RSC (v_3 y v_5).

2.5.1.2.2 Interleaver CDMA-2000

[93]El tamaño del bloque de información *u*, y por consiguiente del Interleaver, debe tener uno de los siguientes valores específicos: 378, 570, 762, 1146, 1530, 2398, 3066, 4602, 6138, 9210, 12282, ó 20730 bits.

El Interleaver utilizado en cdma2000 tiene una estructura especial, denominada *S-random Interleaver* (Interleaver "S-aleatorio"). La idea de este Interleaver, consiste en evitar que los bits sean escritos en posiciones vecinas dentro de una misma ventana de tamaño *S*. Entre más grande sea *S*, el Turbo Decodificador tendrá un mejor desempeño. El procedimiento de este Interleaver es el siguiente: la información es vaciada en el Interleaver en una única fila de tamaño *u*. Luego, se selecciona un bit de manera aleatoria y su posición dentro del Interleaver, es comparada con la posición que tenían los *S* bits que salieron del Interleaver previamente, de forma tal que el bit en consideración saldrá del Interleaver, solo si su ubicación está a una distancia mayor a *S* posiciones de cualquiera de las ubicaciones antiguas de los *S* bits previamente aceptados. El parámetro *S* se define en base a un algoritmo que está en función del tamaño *u* del Interleaver. Este procedimiento es repetido hasta que el Interleaver queda vacío.

2.5.1.2.3 Turbo Decodificador CDMA-2000

La Turbo Decodificación en cdma2000, es exactamente igual al proceso de decodificación iterativa de la Figura 3.44.

[93] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Pág: 133.

2.5.2 TURBO CÓDIGOS UTILIZADOS EN SISTEMAS DE COMUNICACIÓN TERRESTRE

2.5.2.1 Turbo Códigos en Módems ADSL

[94]La aplicación de las técnicas de Turbo Codificación para la corrección de errores ha sido puesta en práctica en la transmisión de datos y sistemas de grabación, desde entonces ha atraído un considerable interés. Ahora se presenta una nueva aplicación de Turbo Codificación para la transmisión en sistemas que utilizan varios tipos de modulación. En particular, el método propuesto se utiliza en Módems ADSL[95] que utilizan tecnología DMT (Discrete Multitone Modulation). La Modulación DMT puede considerarse como una variación de la técnica OFDM[96] que se utiliza sobre todo en el contexto de las comunicaciones inalámbricas.

En el estándar ANSI T1.413, que es el estándar de ADSL publicado en 1995, se especifica el uso de Reed-Solomon (RS) para la codificación de control de errores hacia adelante (FEC). Más tarde, el estándar ANSI introdujo la concatenación de códigos RS con 16 estados y código de trellis dimensional 4, conocido como código de Wei, como una técnica de codificación interna. En esta parte del capítulo del presente proyecto de titulación, el rendimiento de los Turbo Códigos es comparado con el código de Wei con el objeto de mostrar la mayor ganancia de codificación que alcanzan los Turbo Códigos con respecto al código recomendado por el estándar. Los Turbo Códigos se describen también como un esquema de codificación que permite el uso eficaz del ancho de banda, que es una característica importante, dado que el ancho de banda de una señal para la transmisión por cables de par trenzado es limitado. Este esquema emplea un código convolucional concatenado en paralelo en combinación con modulación multitono. La modulación de amplitud en cuadratura (QAM) es adecuada para módems ADSL.

---

[94] FUENTE: KEATTISAK, Sripimanwat; Turbo Code Applications: A Journey From a Paper to Realization; Págs: 67-77.

[95] ADSL: (Asymmetric Digital Subscriber Line) transmisión de datos asimétricos a alta velocidad, que es aplicada sobre los bucles de abonado de la red telefónica. Para ello utiliza frecuencias más altas que las empleadas en el servicio telefónico sin interferir en ellas, permitiendo así el uso simultáneo de datos y voz.

[96] OFDM: (Orthogonal Frequency Division Multiplexing) es una modulación que consiste en enviar un conjunto de ondas portadoras de diferentes frecuencias, donde cada una transporta información, la cual es modulada en QAM o en PSK.

Uno de los retos en el diseño de estos códigos es dimensionar de manera adecuada la longitud del interleaver. Los Turbo Códigos tienen un rendimiento cercano al límite de Shannon, cuando la longitud del interleaver está en el orden de los varios miles de bits. Sin embargo, muchas aplicaciones prácticas, incluyendo ADSL, tienen limitaciones en términos de retraso o latencia. Esta demora es una de las limitaciones porque fuerzan al diseñador a limitar la longitud del interleaver en los Turbo Códigos. Por esta razón, se debe prestar especial atención al diseño de los interleavers. Los interleavers pseudo-aleatorios son diseñados con longitud de bloque corta, ya que como resultado se tiene altas ganancias de codificación. Además los interleavers pseudo-aleatorios evitan los efectos de piso de error en el alcance práctico de las tasas de error de bit (BER) para aplicaciones ADSL.

Dado que el tamaño de la constelación de símbolos en las diferentes sub-portadoras del módem ADSL puede ser diferente, la tasa de código para cada constelación QAM varía dentro del rango de un símbolo DMT (trama). Por esta razón, se debe prestar especial atención para decodificar este sistema, teniendo posiblemente una diferente tasa de código en cada sub-portadora.

2.5.2.1.1 Diseño del Turbo Codificador para Módems ADSL

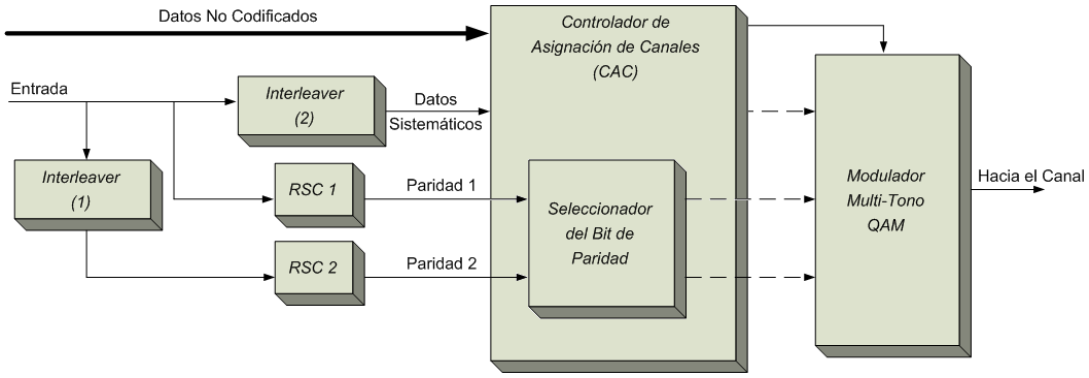


Figura 2.20. Diagrama de Bloques del Turbo Codificador. [97]

[97] FUENTE: KEATTISAK, Sripimanwat; Turbo Code Applications: A Journey From a Paper to Realization; Pág: 69.

La Figura 2.20 describe el diagrama de bloques de un Turbo Codificador que puede ser usado en un módem ADSL. La estructura del Turbo Codificador es similar al esquema clásico con un interleaver adicional en la ruta de acceso de datos sistemáticos. El Turbo Codificador recibe los bits de información como entrada y estos generan un bit de datos sistemáticos entrelazados y dos bits de paridad por cada bit de información. El interleaver adicional previsto en el camino de los datos sistemáticos está diseñado para combatir el ruido de impulso que a veces afecta a las señales transmitidas por cables de par trenzado. A los bits de paridad se les hace puncturings basados en la información proporcionada por el *Controlador de Asignación de Canales* (CAC). Durante la inicialización del módem, el CAC recoge la información sobre el nivel de ruido y la condición de canal de cada sub-portadora. Basado en las mediciones para un lazo en particular del *power mask* y el *power budget constraint*, el CAC asigna una combinación de datos no codificados, datos entrelazados sistemáticos, y bits de paridad con un puncturer para cada sub-portadora.

En particular, un algoritmo de carga de bits se utiliza para asignar el número de información y bits de paridad para cada sub-portadora, y por lo tanto, determina el tamaño de constelación para cada sub-portadora. El CAC determina la adecuada combinación de bits sin codificar, bits sistemáticos entrelazados, y se hace un puncturer en los bits de paridad para mapear un símbolo M-QAM. Se debe notar que algunos fragmentos de información no pasan a través del Turbo Codificador. Estos bits se suelen asignar a los bits más protegidos en un símbolo M-QAM para asegurar una mayor protección. El bit más protegido es el que tiene la menor probabilidad de detección incorrecta en comparación con los otros bits en un símbolo. Esto puede ser definido por un símbolo o por cada dimensión (I o Q) de un símbolo complejo. El Código de Gray es la aplicación más común para la asignación de los bits a una constelación M-QAM. En este caso, cada componente I o Q de un símbolo M-QAM se asigna usando un código de Gray por separado. En el código de Gray, el bit más protegido es el bit más significativo. La principal ventaja del código de Gray es su simplicidad en términos de complejidad de la implementación. De los datos que se generan en la salida del Turbo Codificador, se asignan los bits de paridad a los bits más importantes y los bits

sistemáticos restantes a los bits menos significativos. Debe haber un número igual de bits de paridad con puncturing de cada componente del Turbo Codificador.

Cada turbo bloque contiene *N* bits de información. También el número total de bits que puede transmitirse durante una trama DMT en ADSL depende de las características de los cables de par trenzado que llevan la señal. En general, no hay ninguna relación entre el tamaño del turbo bloque *N*, y el número de bits cargados en cada trama ADSL, $N_f . N_f$ depende de las características del canal, es decir, de la función de atenuación del canal, el ruido y el nivel de las interferencias en una línea particular. La función CAC velará por que los límites del turbo bloque estén adecuadamente tratados en el transmisor, para que el receptor pueda decodificar cada turbo bloque. El CAC también controla el flujo de datos en el receptor. En el receptor, la probabilidad condicional de cada bit transmitido puede ser directamente calculada a partir de la constelación M-QAM.

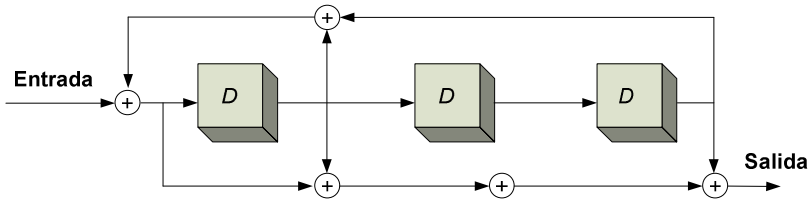


Figura 2.21. Codificador RSC para Turbo Códigos con Matriz Generadora $\left[1, \frac{17_{oct}}{15_{oct}}\right]$.[98]

En lo que respecta a los codificadores convolucionales sistemáticos recursivos, cada codificador de RSC tiene una matriz generadora $\left[1, \frac{g_{ff}(D)}{g_{fb}(D)}\right]$, donde $g_{ff}(D)$ y $g_{fb}(D)$ son los polinomios *feedforward* y *feedback* del codificador RSC, respectivamente. El polinomio *feedback* es usualmente seleccionado como un polinomio primitivo. La Figura 2.21 muestra un ejemplo del codificador RSC recomendado para los Turbo Códigos con polinomios generadores *feedback* y *feedforward* iguales a 15_{oct} y 17_{oct} respectivamente.

[98] FUENTE: KEATTISAK, Sripimanwat; Turbo Code Applications: A Journey From a Paper to Realization; Pág: 70.

2.5.2.1.2 Diseño del Turbo Decodificador para Módems ADSL

El algoritmo MAP es computacionalmente complejo y sensible a la relación S/N e inexacto para la estimación de la varianza del ruido. Este algoritmo requiere funciones no lineales para el cálculo de las probabilidades y para las multiplicaciones adiciones necesarias. Por ejemplo, para representar el *fixed-point* de las variables de decodificación MAP, se suele requerir entre 16 a 24 bits para una constelación de la señal QPSK.

Basados en los requisitos de hardware, el algoritmo MAP no es práctico para ser implementado en un chip. La versión logarítmica del algoritmo MAP y el algoritmo SOVA, son los algoritmos de decodificación usados en la práctica. Estos algoritmos son menos sensibles a la relación S/N y un poco más exactos a la estimación de la varianza de ruido, y la representación del *fixed-point* de sus variables requieren alrededor de 8 bits para una constelación de la señal QPSK. Todas las diferentes versiones logarítmicas del algoritmo MAP solo implican adiciones y una operación MAX que puede realizarse utilizando una simple tabla de *look-up* o un detector de umbral.

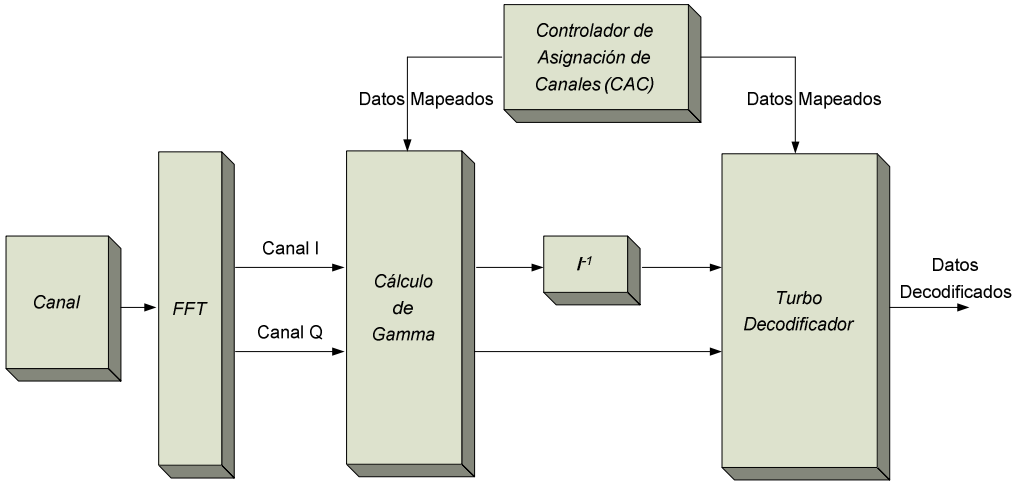


Figura 2.22. Diagrama de Bloques del Turbo Decodificador. [99]

La Figura 2.22 muestra el diagrama de bloques del Turbo Decodificador en el receptor del módem ADSL. Las muestras de la señal recibida son procesadas por

[99] FUENTE: KEATTISAK, Sripimanwat; Turbo Code Applications: A Journey From a Paper to Realization; Pág: 73.

un bloque FFT[100]. A la salida de este bloque, se obtienen símbolos complejos QAM representando información en las dimensiones real (I) e imaginaria (Q). Esta información se utiliza para calcular las variables Gamma y los resultados se introducen en el Turbo Decodificador, después de realizar el de-interleaving las variables Gamma son relacionadas a los datos sistemáticos. El CAC controla los límites de la trama FFT y el Turbo Código.

2.5.2.1.3 Diseño del Interleaver para Turbo Códigos en Módems ADSL

Un interleaver π es una permutación que mapea una secuencia de datos de N símbolos de entrada $d_1,d_2,...,d_N$ dentro de la misma secuencia en un nuevo orden. Si la entrada de secuencia de datos es $d = [d_1, d_2, . . . , d_N]$, entonces la secuencia de datos permutado es dP, donde P es una matriz de interleaving con un solo 1 en cada fila y columna, todas las demás entradas están en cero. Cada interleaver tiene su correspondiente de-interleaver (π^{-1}) que actúa sobre la secuencia de datos intercalados y los restaura a su orden original. La matriz de-interleaving es la traspuesta de la matriz interleaving (P^T).

Un interleaver aleatorio es simplemente una permutación aleatoria π. Para grandes valores de N, la mayoría de interleavers aleatorios utilizados en Turbo Códigos tienen un buen desempeño. Sin embargo, a medida que disminuye el tamaño del bloque del interleaver, el rendimiento del Turbo Código se degrada considerablemente, hasta un punto en el que su rendimiento es peor que el BER de un código convolucional con complejidad computacional similar. Por esto, el diseño de interleavers cortos para los Turbo Códigos es un problema importante. Un interleaver aleatorio S (donde S = 1, 2, 3,...) es un interleaver “semi-aleatorio” construido de la siguiente manera. Cada número entero seleccionado aleatoriamente es comparado con S, que previamente es seleccionado de enteros aleatorios. Si la diferencia entre la actual selección y la selección anterior de S es menor que S, el número entero aleatorio se rechaza. Este proceso se repite hasta que N enteros distintos hayan sido seleccionados. Las simulaciones

[100] FFT: (Fast Fourier Transform) es un eficiente algoritmo que permite calcular la transformada de Fourier discreta (DFT) y su inversa. La FFT es de gran importancia en una amplia variedad de aplicaciones, desde el tratamiento digital de señales y filtrado digiital los algoritmos de multiplicación rápida de grandes enteros. El algoritmo pone algunas limitaciones en la señal y en el espectro resultante.

computacionales han demostrado que si $S \leq \sqrt{\frac{N}{2}}$, entonces este proceso converge en un tiempo razonable. Este interleaver diseñado asegura que los eventos de ciclo corto sean evitados. Un evento de ciclo corto se produce cuando dos bits están cerca uno del otro antes y después del interleaving.

El Turbo Código propuesto para módems ADSL requiere dos interleavers. Es bien sabido que para una longitud de bloque dada, el diseño del interleaver puede tener un efecto significativo sobre el rendimiento de decodificación, debido a los diferentes valores de la distancia libre mínima efectiva que cada interleaver puede tener. El primer tipo es el interleaver que se encuentra en la entrada del codificador RSC (ver Figura 2.21). Para longitudes cortas del turbo bloque, este interleaver es responsable de producir grandes valores de distancia libre mínima efectiva d_{min}, a la salida del Turbo Codificador. Cuando la longitud del turbo bloque es corta, entonces un interleaver aleatorio puede crear pequeñas d_{min} lo que se traduce en un pobre rendimiento BER para el Turbo Código. Hay varios diseños de interleavers cortos para los Turbo Códigos.

Un diseño de interleaver basado en el desempeño iterativo de decodificación de los Turbo Códigos está fundamentado en el algoritmo MAP o en otros algoritmos que pueden proporcionar una salida suave. En cada paso de decodificación, algo de información relacionada con los bits de paridad de un decodificador se introducen en el otro decodificador, junto con la secuencia de datos sistemáticos y los bits de paridad correspondiente a ese decodificador. La Figura 2.23 muestra este sistema iterativo de decodificación.

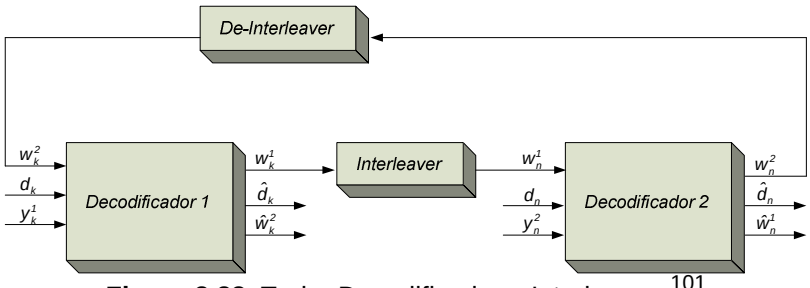


Figura 2.23. Turbo Decodificador e Interleaver.[101]

[101] FUENTE: KEATTISAK, Sripimanwat; Turbo Code Applications: A Journey From a Paper to Realization; Pág: 77.

Las entradas a cada decodificador son la secuencia de entrada de datos d_k, los bits de paridad y_k^1 o y_k^2, y el LLR asociado con los bits de paridad del otro decodificador (w_k^1 o w_k^2), que es utilizado como información a priori. Todas estas entradas son utilizadas por el decodificador para crear tres salidas correspondientes a la versión ponderada de estas entradas.

En la Figura 2.23, $\hat{d}_k$ representa la versión ponderada de la secuencia de entrada de datos d_k; en la misma figura d_n muestra que la secuencia de datos de entrada es introducida en el segundo decodificador después del interleaving. La entrada a cada decodificador desde el otro decodificador se utiliza como información a priori en los próximos pasos de decodificación y corresponde a la versión ponderada de los bits de paridad. Esta información será más eficaz en el desempeño de decodificación iterativa si esta es menos correlacionada a la secuencia de datos de entrada (realizado el interleaving de la secuencia de datos de entrada). Por lo tanto, es razonable utilizar esto como un criterio para el diseño del interleaver.

Para interleavers de gran tamaño de bloque, la mayoría de interleavers aleatorios proporcionan una baja correlación entre w_k^i y la secuencia de datos de entrada d_k. El coeficiente de correlación $r^1_{w^1_{k_1}, d_{k_2}}$, se define como la correlación entre $w^1_{k_1}$ y d_{k_2}.

2.5.3 TURBO CÓDIGOS UTILIZADOS EN SISTEMAS SATÉLITALES

2.5.3.1 Turbo Códigos para Digital Video Broadcasting

[102]El Proyecto *Digital Video Broadcasting* (DVB) fue fundado en 1993 por la *European Telecommunications Standards Institute* (ETSI) con el objetivo de normalizar los servicios de televisión digital. Su estándar inicial estuvo orientado a la entrega televisión digital por satélite y es conocido como DVB-S, el cual usa una concatenación de combinación externa (204,188) bytes de código abreviado

[102] FUENTE: KEATTISAK, Sripimanwat; Turbo Code Applications: A Journey From a Paper to Realization; Págs: 301-306.

Reed Solomon, un constraint length interior de 7 y código convolucional de tasa variable (rangos de *r* desde 1/2 a 7/8).

La misma infraestructura utilizada para ofrecer televisión vía satélite puede también ser utilizada para entregar servicios de Internet y datos para el abonado. El Internet a través de DVB-S es un competidor natural del cable módem y la tecnología DSL, y su cobertura universal permite que incluso las zonas más remotas puedan tener servicio. Debido a que DVB-S solo proporciona un enlace descendente, es también necesario tener un enlace ascendente para permitir aplicaciones interactivas, como navegación por Internet. El enlace ascendente y descendente no necesitan ser simétricos. Una alternativa para el enlace ascendente, es utilizar un módem telefónico, pero esto no permite tener un servicio permanente, tiene modestas tasas de datos, y puede ser costoso en zonas remotas. Una alternativa más atractiva es que los equipos de abonado puedan transmitir una señal de enlace ascendente de nuevo al satélite con la misma antena utilizada para la recepción de la señal de enlace descendente. Sin embargo, dada la apertura de una antena pequeña, la exigencia de un bajo costo y el amplificador de baja potencia, hay muy poco excedente en el enlace ascendente. Por lo tanto, se desea un esquema de codificación FEC fuerte. Por esta razón, el Proyecto DVB ha adoptado a los Turbo Códigos para el canal de retorno por satélite en su estándar DVB-RCS (*Return Channel Via Satellite*).

Al mismo tiempo que el Proyecto DVB estaba desarrollando la tecnología de Turbo Codificación para el canal de retorno, se actualizó el enlace descendente con moderna tecnología de codificación. El último estándar, llamado DVB-S2, sustituye a los concatenados Reed-Solomon/Convolucionales, método de codificación de DVB-S con una concatenación de un código exterior BCH y un código interior LDPC[103] (*Low Density Parity Check*). El resultado es un aumento del 30% de la capacidad sobre DVB-S.

[103] LDPC: Los códigos LDPC son un tipo de códigos de bloque lineal, caracterizados por una matriz de chequeo de paridad H esparcida (o dispersa), es decir con pocos unos en relación al número de ceros. Como los códigos turbo, los códigos LDPC se decodifican en forma iterativa. En general, la complejidad por iteración de los LDPC es menor que la de los códigos turbo. Sin embargo, pueden necesitarse muchas más iteraciones (entre 30 y 100, comparado con 5 a 18).

2.5.3.1.1 DVB-RCS

El Turbo Código DVB-RCS fue optimizado para tramas de corto tamaño y datos con altas tasas. Doce tamaños de trama son soportados, que van desde 12 bytes hasta 216 bytes, incluyendo una trama de 53 bytes compatible con ATM y una trama de 188 bytes compatible tanto con MPEG-2 como con el estándar original de DVB-S. El enlace de retorno soporta velocidades de datos de 144 kbps a 2 Mbps y es compartida entre los terminales mediante el uso de MF-TDMA (*Multi-Frequency Time-Division Multiple-Access*) y técnicas DAMA (*Demand-Assigned Multiple-Access*). Ocho tipos de código son soportados, desde r = 1/3 a r = 6/7.

Al igual que los Turbo Códigos utilizados en otros estándares, un par de codificadores RSC constituyentes se utilizan junto con el algoritmo Log-MAP o Max-log-MAP para la decodificación. El decodificador para cada componente de código funciona mejor si el codificador comienza y termina en un estado conocido, como el estado todo ceros. Esto se puede lograr de forma independiente terminado el trellis de cada codificador con una cola que obliga al codificador a regresar al estado todo ceros. Sin embargo, para las tramas de longitudes pequeñas soportadas por DVB-RCS, una cola impone una reducción no despreciable en la tasa de código y esta es indeseable. Como una alternativa para la terminación del trellis del código, DVB-RCS utiliza codificación CRSC (*Circular Recursive Systematic Convolucional*), que se basa en el concepto de *tailbiting*[104]. Los códigos CRSC no utilizan colas, sino que están codificados de tal manera que el estado final coincide con el estado de partida.

La mayoría de los Turbo Códigos utilizan codificadores binarios definidos sobre un *Galois Field*[105]. Sin embargo, para facilitar la decodificación rápida en hardware, el código de DVB-RCS utiliza codificadores *duobinary*[106] constituyentes definidos

[104] TAILBITING: Esquema de transmisión trama por trama, en el que los datos están convolucionalmente codificados, para que el codificador comience y termine en el mismo estado, el que sin embargo es desconocido para el decodificador. La ventaja de este esquema es que no se añade cola (*overhead*) a los datos para obligar al codificador a entrar por el decodificador a un estado conocido.
[105] GALOIS FIELD: En álgebra abstracta, un cuerpo finito, campo finito o campo de Galois (llamado así por Évariste Galois) es un cuerpo que contiene un número finito de elementos.
[106] DUOBINARY: La codificación duobinaria consiste en generar en el transmisor, a partir de la suma de dos bits consecutivos de una señal binaria a *B* bits/s una señal de tres niveles a *B/2* bits/s. Por eso para una velocidad *B* solo se precisa un ancho de banda de *B/2* y por tanto puede incrementarse la máxima distancia.

sobre un *Galois Field*. Durante cada ciclo de reloj, el codificador toma dos bits de datos y las salidas de dos bits de paridad para que cuando los bits sistemáticos sean incluidos, la tasa de código sea de $r = 2/4$. A fin de evitar las transiciones paralelas en el código de trellis, la memoria del codificador debe exceder el número de bits de entrada, así DVB-RCS utiliza codificadores constituyentes con memoria tres (una constraint length de cuatro).

Hay varias ventajas en la utilización de codificadores *duobinary*. En primer lugar, el trellis contiene la mitad de estados que tiene un código binario de idéntica constraint length (pero el mismo número de bordes) y por lo tanto necesita de la mitad de la cantidad de memoria, y el hardware de decodificación puede ser sincronizado a la mitad de la tasa como un código binario. En segundo lugar, los códigos *duobinary* pueden ser decodificados con el subóptimo, pero eficiente algoritmo Max-log-MAP, a un costo de solo 0.1 a 0.2 dB en relación con el algoritmo óptimo MAP. En contraste con los códigos binarios, mismos que pierden alrededor de 0.3 a 0.4 dB cuando se decodifica con el algoritmo Max-log-MAP. Además, los códigos *duobinary* son menos afectados por la incertidumbre de los estados inicial y final cuando se utiliza *tailbiting* y tienen un mejor desempeño que sus homólogos binario cuando se realiza el punturer a tasas más altas.

- *Codificador:* El codificador constitutivo CRSC utilizado por DVB-RCS se muestra en la Figura 2.24. El codificador se alimenta de bloques de k bits de mensaje que se agrupan en $N = k/2$ parejas. El número de parejas por cada bloque puede ser $N \in (48, 64, 212, 220, 228, 424, 432, 440, 752, 848, 856, 864)$. El número de bytes por bloque es $N/4$. En la Figura 2.24, A representa el primer bit de la pareja, y B representa el segundo bit. Los dos bits de paridad se representan como W e Y. Para facilitar la exposición, los subíndices se quedan fuera de la figura, pero por debajo de un único subíndice que se utiliza para indicar el tiempo de índice $k \in (0, ..., N - 1)$ y un segundo índice opcional se utiliza en los bits de paridad W e Y para indicar cual de los dos codificadores constituyentes los produjo.

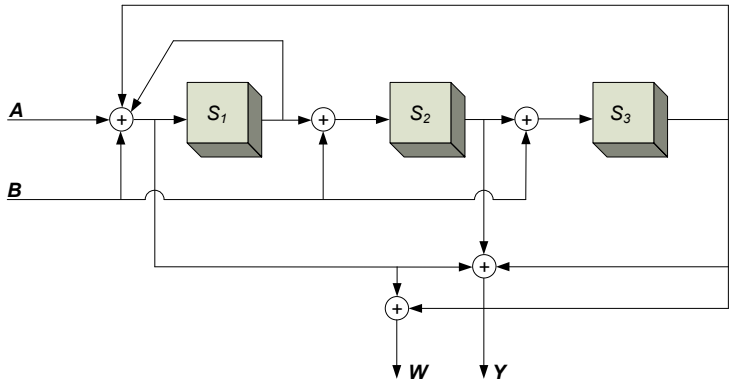


Figura 2.24. Codificador Constituyente Duobinario CRSC usado por DVB-RCS. [107]

- *Decodificador:* La decodificación del código DVB-RCS es complicada por el hecho de que los códigos constituyentes son *duobinary* y circular. Al igual que en los Turbo Códigos convencionales, la decodificación implica el intercambio iterativo de información extrínseca entre los dos decodificadores componentes. Mientras que la decodificación se puede realizar en el dominio de la probabilidad, el dominio logarítmico es preferible desde el máximo la baja complejidad del algoritmo Max-log-MAP puede ser aplicado. A diferencia del decodificador para un Turbo Código binario, en el que puede representarse cada símbolo binario como una simple relación log-likelihood, el decodificador para un código *duobinary* requiere tres relaciones log-likelihood. Por ejemplo, la relación likelihood para un par de mensajes (A_k, B_k) se puede representar en la forma:

$$\Lambda_{a,b}(A_k, B_k) = \log\frac{P(A_k = a, B_K = b)}{P(A_k = 0, B_K = 0)} \qquad \textbf{(Ecuación 2.9)}$$

Donde (*a, b*) puede ser (0, 1), (1, 0), o (1, 1).

[107] FUENTE: KEATTISAK, Sripimanwat; Turbo Code Applications: A Journey From a Paper to Realization; Pág: 303.

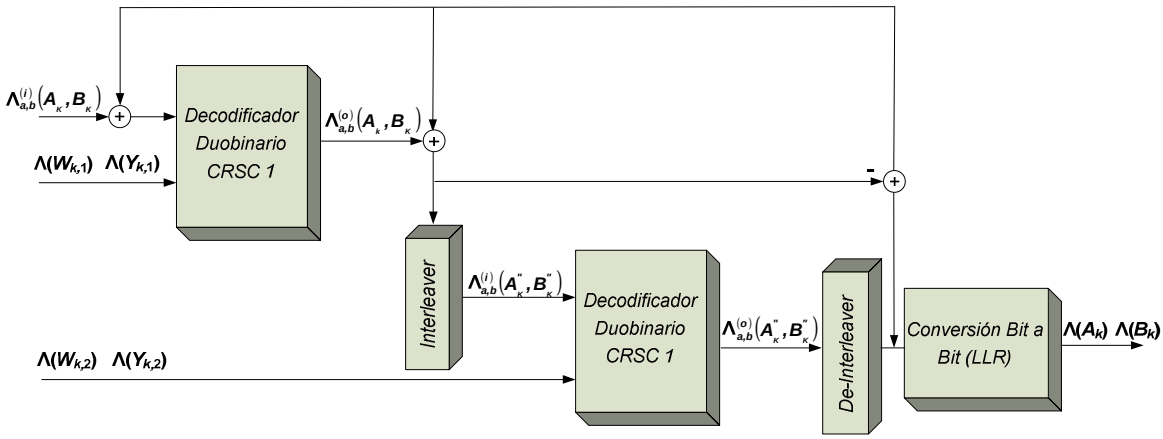


Figura 2.25. Decodificador para el Código DVB-RCS. [108]

Un decodificador iterativo que puede ser usado para decodificar el Turbo Código DVB-RCS es el que se muestra en la Figura 2.25. El objetivo de cada uno de los dos decodificadores constituyentes es actualizar el conjunto de relaciones log-likelihood asociadas a cada par de mensajes. En la figura, $\{\Lambda_{a,b}^{(i)}(A_k, B_K)\}$ denota el conjunto de LLRs correspondientes al par de mensajes en la entrada del decodificador y $\{\Lambda_{a,b}^{(o)}(A_k, B_K)\}$ es el conjunto de LLRs en la salida del decodificador. Cada decodificador es siempre previsto con $\{\Lambda_{a,b}^{(i)}(A_k, B_K)\}$ junto con los valores recibidos de los bits de paridad generados por el codificador correspondiente (en forma LLR). El uso de estas entradas y el conocimiento de los constraints del código, es capaz de producir la actualización de los LLRs $\{\Lambda_{a,b}^{(o)}(A_k, B_K)\}$ en su salida.

Al igual que con los Turbo Códigos binarios, la información extrínseca pasa al otro decodificador constituyente en lugar de las LLRs primas. Esto evita la retroalimentación positiva de la información previamente resuelta. La información extrínseca se encuentra simplemente restando la entrada LLR adecuada de cada salida LLR, como se indica en la Figura 2.25.

A la información extrínseca que es pasada entre los dos decodificadores se la debe realizar un interleaving o de-interleaving porque está en la misma

[108] FUENTE: KEATTISAK, Sripimanwat; Turbo Code Applications: A Journey From a Paper to Realization; Pág: 305.

secuencia que la entrada del otro decodificador. El interleaving y de-interleaving entre los dos decodificadores constituyentes debe ser hecho en un símbolo de base racional, asegurando que las tres relaciones likelihood $\{\Lambda_{0,1}(A_k, B_k), \Lambda_{1,0}(A_k, B_k), \Lambda_{1,1}(A_k, B_k)\}$ pertenecientes a la misma pareja no estén separadas.

2.5.3.2 Servicios Satelitales que Usan Tecnología de Turbo Códigos

[109]Los Turbo Códigos convolucionales y los Turbo Códigos de bloque, junto con los códigos LDPC representan los diferentes tipos de tecnología de Turbo Codificación para la nueva generación de servicios por satélite. Esta sección del presente proyecto de titulación se centra en la aplicación de estos Turbo Códigos en equipos satelitales y algunos de los fabricantes de este tipo de tecnología.

Aparte de los Turbo Códigos originales, la mayoría de los codecs son soluciones propietarias. Los grandes fabricantes de FPGA[110], como Xilinx o Altera también ofrecen ejemplares de Turbo Codecs.

Una aplicación particular de los Turbo Códigos de bloque es el TPC (*Turbo Product Code*) para los que están disponibles decodificadores ASIC muy rápidos, en Comtech AHA Corporation. La familia de los TPCs inicia con el codec AHA4501 Astro de 36 Mbps. Existe el AHA4522 Astro LE 2k y el AHA4524 Astro LE 4k que utilizan interleavers de 2k o 4k respectivamente. El codec AHA4525 es compatible con IEEE 802.16 (WiMax). El AHA4540 Astro OC-3 es un dispositivo de un solo chip TPC codificador/decodificador con velocidad de 155 Mbps para tasas de código mayores a 0.9. En la parte superior del código del TPC hay un dispositivo de cálculo del CRC que se usa para comprobar la integridad de datos al final del decodificador. El chip puede manejar tasas de codificación de 0.25 a 0.98 en tamaños de bloque fijo. Este incluye un módulo de cálculo de métricas

[109] FUENTE: KEATTISAK, Sripimanwat; Turbo Code Applications: A Journey From a Paper to Realization; Págs: 258-269.

[110] FPGA: (Field Programmable Gate Array) Es un dispositivo semiconductor que contiene bloques de lógica cuya interconexión y funcionalidad se puede programar. La lógica programable puede reproducir desde funciones tan sencillas como las llevadas a cabo por una puerta lógica o un sistema combinacional hasta complejos sistemas en un chip. Las FPGA son el resultado de la convergencia de dos tecnologías diferentes, los dispositivos lógicos programables PLDs y los circuitos integrados de aplicación específica ASIC.

suaves para esquemas BPSK, QPSK, 8PSK, 16QAM, 64QAM y 256QAM. El AHA4540B es una versión mejorada del AHA4540A, que mecánicamente tiene el mismo tamaño y la misma pisada. La versión B es aproximadamente 15% más rápida, lo que permite una selección más amplia de códigos que respondan al estándar OC3. El AHA4541 es un codec que puede funcionar a velocidades de hasta 311 Mbps (o 360 Mbps de velocidad de canal). El rendimiento para la modulación QPSK es relativamente buena para BERs por debajo de 10^{-8}. Algunas implementaciones ofrecen una concatenación entre el TPC y los códigos BCH con el fin reducir la tasa mínima de error.

Iterative Connections lanzó a finales del 2003, la familia S - tec™ de códigos convolucionales concatenados en serie. Estos códigos permiten comunicaciones satelitales con eficiencias menores a 1 dB de la capacidad del canal, para una BER menor a 10^{-10}. Además de la eficiencia en potencia, ofrecen al usuario la opción de cambiar a un modo de comunicación segura; esto se logra por medio de una configuración única del codec entre el transmisor y el receptor. En principio este modo no puede ser utilizado, por ningún esquema serial concatenado en el que la información no se envía en forma clara. La familia S - tec™ integró a la línea de módems de Sistemas *Datum* la opción de *plug-in*[111] y también ofrece una nueva línea de módems satelitales, los *Premier 5/20/45*, de *Iterative Connections*.

Hasta hace poco, los únicos módems satelitales disponibles en el mercado eran los que incorporaban códigos Turbo-Like, además de los módems compatibles con el estándar DVB-RCS, que usaban solo TPCs. A pesar de que los TPCs son un importante avance en relación con el estándar Viterbi/Reed-Solomon, su rendimiento puede ser mejorado aún más para conseguir más y más capacidad. Con los nuevos Turbo Codecs es probable ver una subdivisión del mercado en redes cerradas que usan un esquema de codificación específico optimizado para sus aplicaciones en particular.

[111] PLUG-IN: Un complemento (o plug-in) es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy especifica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la API.

A continuación se exponen algunos de los fabricantes de Turbo Módems Satelitales más relevantes:

2.5.3.2.1 Advantech

Advanced Microwave Technologies, lanzó a principios del 2004 su módem satelital AMT-70 con un rango de 8 kbps a 140 Mbps, BPSK a 16 QAM, 70/140 MHz o banda L con una versión mejorada de la opción TPC. El SPL-ACT (también una empresa *Advantech AMT*), ofrece la serie AMS 8192 de módems satelitales que son compatibles con los estándares IESS y DVB-S, con tasas de datos de hasta 8192 kbps en la banda L e interfaz de 70 Mhz.

2.5.3.2.2 iDirect

La tecnología *iDirect* utilizada en su línea de módems satelitales, se basa en TPCs. La compañía afirma que alcanza un rendimiento IP, 25% mayor que el estándar DVB, o 62% más de datos sobre el ancho de banda satelital equivalente.

2.5.3.2.3 ViaSat

ViaSat, ofrece una pequeña lista de productos basados en Turbo Códigos, como: *LinkStar*, *Surfbeam* y *WildBlue*. *LinkStar* es un sistema VSAT[112] de banda ancha, de dos vías con ancho de banda bajo demanda que utiliza el estándar *DVB-Multi-Protocol Encapsulation* en el enlace hacia adelante y el estándar *DVB-RCS* para el enlace de retorno. *Surfbeam* y *WildBlue* se basan en una versión satelital habilitada del estándar DOCSIS 1.1 (*Data Over Cable Service Interface Specifications*). Esta es una tecnología con beneficios muy significativos, como terminales de abonado de bajo costo debido a conjuntos de chips de alto volumen, tiempo rápido de salida al mercado, disponibilidad de la infraestructura de productos para el control de la red, gestión del sistema, gestión de abonados, y sistemas de facturación. Estos terminales también se utilizarán para el último haz puntual por satélite de Banda Ka Canadiense, el *Anik F2*. *Intelsat* y *Eutelsat*,

[112] VSAT: Las redes VSAT (Very Small Aperture Terminals) son redes privadas de comunicación de datos vía satélite para intercambio de información punto-punto o, punto-multipunto (broadcasting) o interactiva.

actualmente permiten a los consumidores elegir entre los módems satelitales *LinkStar* o *SufBeam*.

2.5.3.2.4 Iterative Connections

Iterative Connections ha integrado su propio codec de tecnología S - tec™ en un mejorado módem de Sistemas *Datum* que puede lograr la sincronización en una E_b/N_o por debajo de 1 dB. Este nuevo módem satelital llamado *Premier 5*, que se muestra en la Figura 2.26, puede proporcionar tasas de datos de hasta 5 Mbps en modo QPSK con tasas codificación de 1/2, 3/4 y 7/8.

El *Premier 5* es el más eficiente módem satelital actualmente disponible en el mercado. Este es el primero de una serie de módems satelitales que incluyen al *Premier 20* y el *Premier 45* que proporcionan modulaciones 8PSK y 16QAM con tasas de hasta 20 y 45 Mbps, respectivamente. Una de las características del codec S - tec™ es la capacidad de trabajar en modo seguro que permite una configuración única para cada módem, basado en una única configuración de interleaver. Esto hace que sea extremadamente difícil interceptar las comunicaciones en un enlace punto a punto. En este momento hay hasta 45000 posibles combinaciones para el interleaver pseudo-aleatorio. Incluso si el atacante encuentra el 90% del interleaver usado para una trama en particular, el BER estaría cerca de 0.3 lo que hace que los datos decodificados sean inútiles.

Figura 2.26. Módem Satelital Premiere 5.[113]

2.5.3.2.5 Sistemas Datum

Los módems satelitales de los Sistemas *Datum*, aceptan tarjetas *plug-in* para el codec con el uso de la nueva tecnología S - tec™.

[113] FUENTE: KEATTISAK, Sripimanwat; Turbo Code Applications: A Journey From a Paper to Realization; Pág: 266.

2.5.3.2.6 STM Networks

La familia de productos *Solante*, producidos por *STM Networks*, es una malla completa interconectada de redes VSAT que utiliza MF-TDMA con superposición de SCPC o enlaces de alta velocidad de datos reconfigurables para la demanda. El nuevo Turbo Codec *Solante* con tecnología S - tec™ puede ser utilizado para una variedad de aplicaciones como se muestra en la Figura 2.27. El producto se ha construido con características de escalabilidad, además de un puerto IP que soporta el protocolo SIP. La modulación y la tasa de codificación pueden ser configuradas sobre la marcha en cada enlace demandado, para cumplir con los requerimientos de rendimiento utilizando la capacidad satelital disponible.

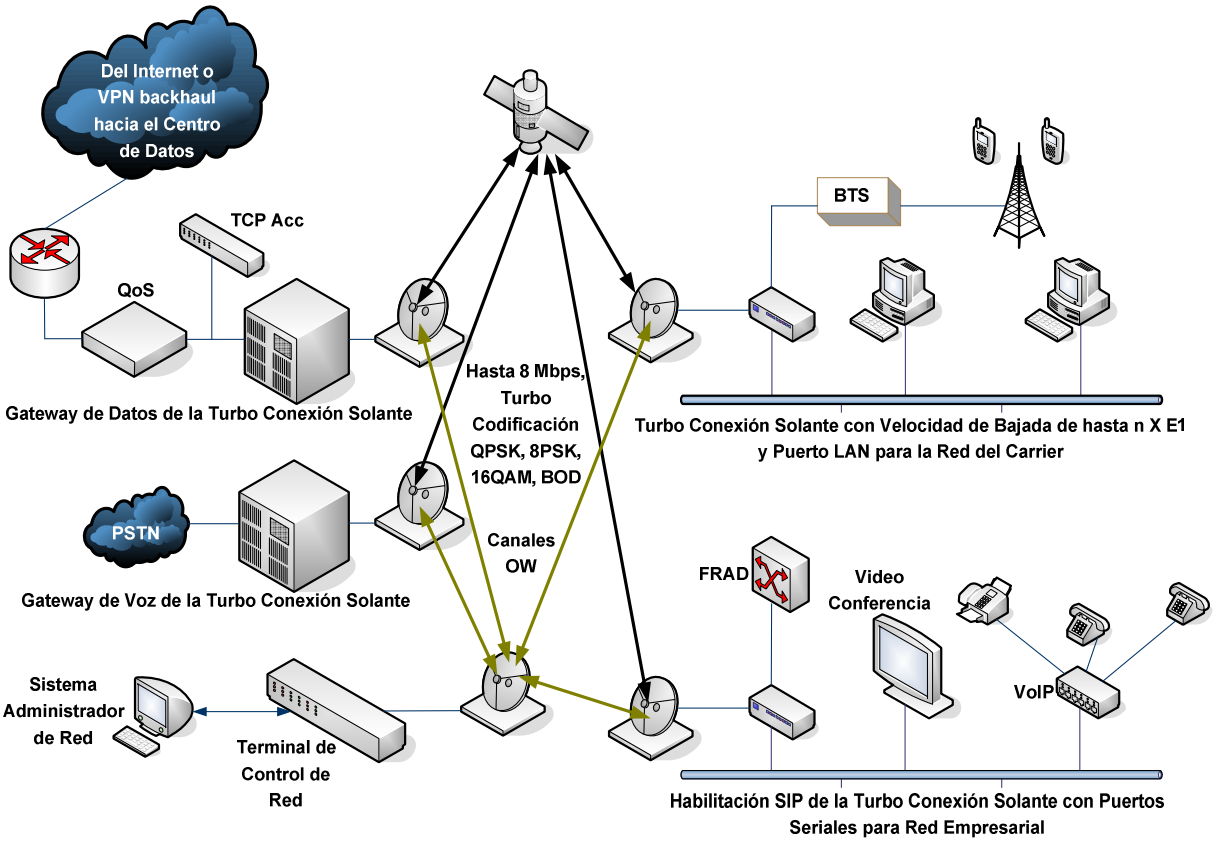


Figura 2.27. Ejemplo de una Turbo Conexión de Red Solante (Empresa STM Networks).[114]

[114] FUENTE: KEATTISAK, Sripimanwat; Turbo Code Applications: A Journey From a Paper to Realization; Pág: 269.

CAPÍTULO 3

DISEÑO, SIMULACIÓN, PRUEBAS Y ANÁLISIS DE RESULTADOS DEL SOFTWARE

3.1 INTRODUCCIÓN

Los Turbo Códigos son el avance más importante en los últimos tiempos, en el área de codificación de canal en sistemas de comunicación digital. Debido a la necesidad de su constante estudio para el conocimiento correcto y total de su funcionamiento, se hacen continuamente diferentes simulaciones, modificando todas las posibles variables que afectan su desempeño. Estas simulaciones generalmente se hacen en formatos muy básicos, ya que son desarrolladas por las mismas personas que las analizan, y en general constan de ventanas sencillas que muestran los resultados numéricos de las simulaciones planteadas, y si es necesario, se grafican estos datos por medio de cualquier herramienta.

Surge entonces la idea de diseñar una herramienta de simulación para Turbo Códigos, que no solo permita un acercamiento directo a este esquema de codificación, sino que además posea una interfaz para sus usuarios, que admita la comprensión y el análisis de resultados de forma clara e inmediata.

Este capítulo del presente proyecto de titulación, muestra el desarrollo de un Turbo Codificador, canal y Turbo decodificador, con su respectiva interfaz gráfica en Matlab como parte del desarrollo de la simulación de un Turbo Codec; además se incluye una ligera explicación de los conceptos que gobiernan los Turbo Códigos y las funciones más relevantes de Matlab que permiten desarrollarlo.

Finalmente se presentan los resultados de las simulaciones cuando los parámetros del Turbo Codec han sido variados y un análisis de dichos resultados.

3.2 TURBO CÓDIGOS

[115]Los Turbo Códigos son un método de corrección de errores basado en los códigos convolucionales más intercalación y realimentación. Consiste en una estructura de codificación concatenada más un algoritmo iterativo; estos fueron introducidos en 1993 por Berrou y Glavieux en la conferencia internacional de la IEEE en Ginebra, Suiza. El esquema propuesto en dicho trabajo alcanzaba un BER de 10^{-5} usando una tasa de codificación de 1/2 sobre un canal AWGN, modulación BPSK con una relación E_b/N_o de 0.7 dB, lo cual está cercano al límite de Shannon que es 0.1dB.

[116]Los Turbo Códigos han sido adoptados por varios sistemas de comunicación debido a la ganancia de codificación extra ofrecida por los Turbo Códigos puede utilizarse en los dispositivos inalámbricos para reducir la potencia de sus señales, lo que conlleva a que más dispositivos puedan compartir el mismo espectro de frecuencia al reducirse la interferencia *inter-dispositivo*. Decrementar la potencia de las señales decrementa también los requerimientos de potencia del sistema, lo cual repercute de manera importante para dispositivos inalámbricos alimentados por batería.

3.3 DISEÑO DEL TURBO CODEC

El grupo “Mobile and Portable Radio Research Group (MPRG)” del Departamento de Ingeniería Eléctrica de la Universidad de Virginia Tech ha desarrollado una serie de funciones en la plataforma computacional Matlab que permite la simulación de algunos esquemas en los que los Turbo Códigos están presentes. El presente proyecto de titulación se ha valido de algunas ideas de las funciones ya desarrolladas por este grupo de investigadores, para la implementación del Turbo Codec que permita simular el esquema clásico de Turbo Codificación y Decodificación de manera didáctico.

---

[115] FUENTE: SACANAMBOY, Maribell; Tesis de Maestría: Diseño e Implementación de los Turbo Codificadores definidos en los estándares de Telecomunicaciones cdma2000 (TIA/EIA 2002.2D) y WCDMA (3GPP TS 25.212 v7.2.0) usando Hardware Reconfigurable; Pág: 4.

[116] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Págs: 64-65.

Para el diseño del modelo que permitirá la simulación del esquema de Turbo Codificación se hizo el análisis de investigaciones y estudios previos, buscando implementar bloques básicos y óptimos, que al final obtuvieran resultados con datos y gráficas consistentes.

Los bloques que conforman el modelo realizado se observan en la Figura 3.1. Estos elementos fueron desarrollados utilizando el programa computacional Matlab versión R2008a, aprovechando las características de eficiencia y rapidez de su compilador, ya que los cálculos que implican estas simulaciones así lo requieren.

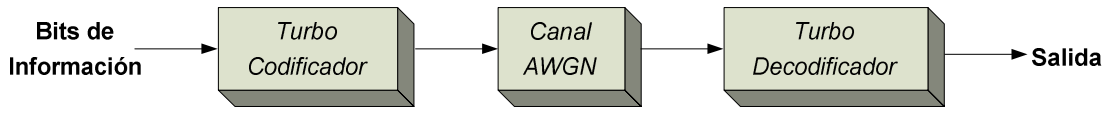


Figura 3.1. Diagrama de Bloques del Modelo.

Se observa en este diagrama de bloques que el modelo desarrollado va desde el bloque codificador en el transmisor, hasta el bloque decodificador en el receptor, de un sistema de comunicación digital. Las sub-funciones que permiten el trabajo del Turbo Codec, están organizadas y articuladas como se muestra en la Figura 3.2:

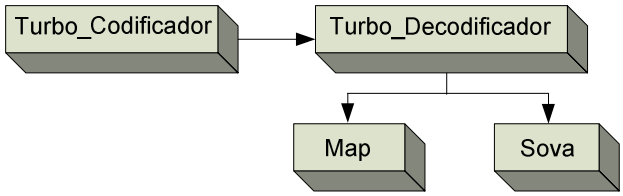


Figura 3.2. Esquema de Trabajo de las Sub-funciones del Turbo Codec.

La labor en conjunto de estas sub-funciones es la base de la herramienta de simulación de Turbo Códigos. Las funciones que implementan este modelo tienen varios parámetros de entrada y de salida que ofrecen la posibilidad de realizar diferentes simulaciones de este esquema de codificación. Se han utilizado varias funciones propias del lenguaje de programación Matlab, mismas que se pueden observar en el Anexo A.

3.3.1 EL TURBO CODIFICADOR

[117]Los Turbo Códigos son una clase de códigos convolucionales concatenados en paralelo, que codifican la información mediante la combinación de dos o más codificadores (esto significa que un Turbo Codificador codifica la misma información dos o más veces), junto con un cierto número de *interleavers* que permutan la información a ser transmitida, de esta forma se producen secuencias no correlacionadas que son codificadas y transmitidas; siendo este concepto la clave del desempeño tan bueno de los Turbo Códigos.

3.3.1.1 Arquitecturas del Codificador

3.3.1.1.1 Arquitectura PCCC (Parallel Concatenated Convolutional Code)

[118]Es la arquitectura más usada por los Turbo Códigos y recibe ese nombre debido a que la misma secuencia de información es codificada dos (o más) veces, en paralelo, utilizando las secuencias normal y permutada de los bits de información. La Figura 3.3 muestra el esquema general de la arquitectura PCCC:

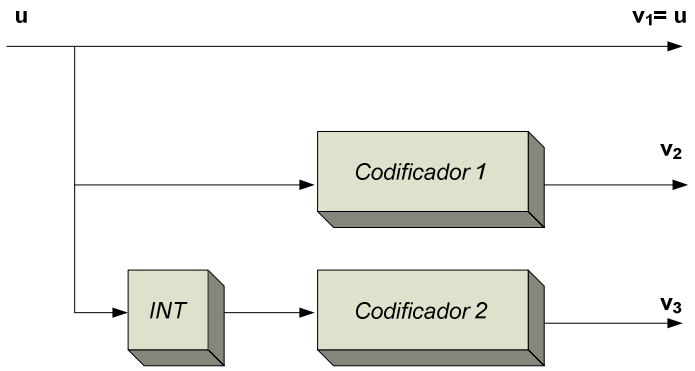


Figura 3.3. Arquitectura PCCC Básica. [119]

La Figura 3.3 muestra un Turbo Codificador que utiliza dos codificadores convolucionales (generalmente del tipo RSC - Recursive Systematic Convolutional

[117] FUENTE: RODRÍGUEZ, Nibaldo; PALMA, Wenceslao; SOTO, Ricardo; Evaluación de Rendimiento de Turbo Código con Diferentes Intercaladores de Bits para Comunicaciones Satelitales; Pág: 2.
[118] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Págs: 67-68.
[119] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Pág: 67.

Code), un *Interleaver* (*INT*) y una conexión directa entre una de las salidas y la entrada del codificador. El *Codificador 1* toma la secuencia de información *u* y produce la secuencia de paridad v_2, mientras que el *Codificador 2* toma la secuencia permutada o 'desordenada' y produce la secuencia de paridad v_3. La secuencia de información $v_1 = u$, junto con las secuencias de paridad v_2 y v_3 forman la salida del Turbo Codificador, que está dada por la Ecuación 3.1:

$$v = \left(v_1^{(1)}, v_2^{(1)}, v_3^{(1)}, v_1^{(2)}, v_2^{(2)}, v_3^{(2)}, \ldots\ldots, v_1^{(k)}, v_2^{(k)}, v_3^{(k)}\right) \qquad \textbf{(Ecuación 3.1)}$$

Existen tres bits de salida por cada bit de entrada, por lo que el Turbo Código tiene una tasa de código r = 1/3. Los codificadores que forman parte de un Turbo Codificador no tienen que ser idénticos, aunque usualmente lo son.

3.3.1.1.2 Arquitectura SCCC (Serial Concatenated Convolutional Code)

[120]La Figura 3.4 muestra un ejemplo de una arquitectura SCCC cuya tasa de codificación es r = 1/3.

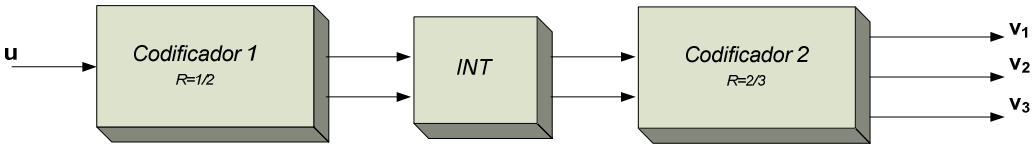


Figura 3.4. Arquitectura SCCC de Tasa r = 1/3. [121]

En la Figura 3.4, el codificador 1 se conoce como *Codificador Externo* y el codificador 2 como *Codificador Interno*. El *interleaver* entre ambos codificadores convolucionales, provoca que este dispositivo produzca resultados distintos en el desempeño de las arquitecturas PCCC y SCCC. Una característica de los esquemas PCCC es que presentan un mejor desempeño que las arquitecturas SCCC a SNR´s bajas; sin embargo, a SNR´s elevadas los esquemas SCCC superan en desempeño a las arquitecturas PCCC.

[120] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Págs: 70-71.
[121] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Pág: 71.

3.3.1.1.3 Arquitectura HCCC (Hybrid Concatenated Convolutional Code)

[122]La arquitectura HCCC, se conoce como *híbrida* ya que es una combinación de las concatenaciones paralela y serial, como muestra la Figura 3.5:

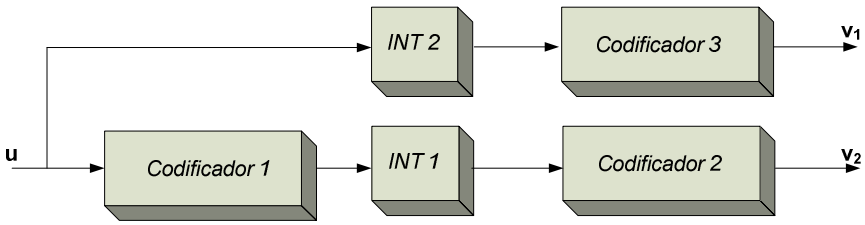


Figura 3.5. Arquitectura HCCC de tasa r = 1/2. [123]

Esta arquitectura no ha sido estudiada con profundidad, pero se sabe que constituye una mejora de la estructura SCCC y PCCC.

3.3.2 DISEÑO DEL TURBO CODIFICADOR

La Figura 3.6 muestra la ubicación en el diagrama de bloques del Turbo Codec, del Turbo Codificador.

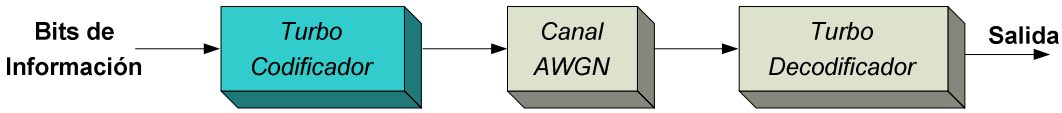


Figura 3.6. Ubicación en el diagrama de bloques del Turbo Codificador.

La arquitectura PCCC es la arquitectura que representa la estructura típica de la Turbo Codificación y es la estructura empleada en este proyecto de titulación, se han concatenado en paralelo dos codificadores RSC y se hace uso de un interleaver. Los bits de información son codificados por el primer componente codificador directamente, mientras que el segundo componente codificador opera con una versión desordenada (por el interleaver) de ellos.

[122] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Págs: 71-72.
[123] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Pág1: 71.

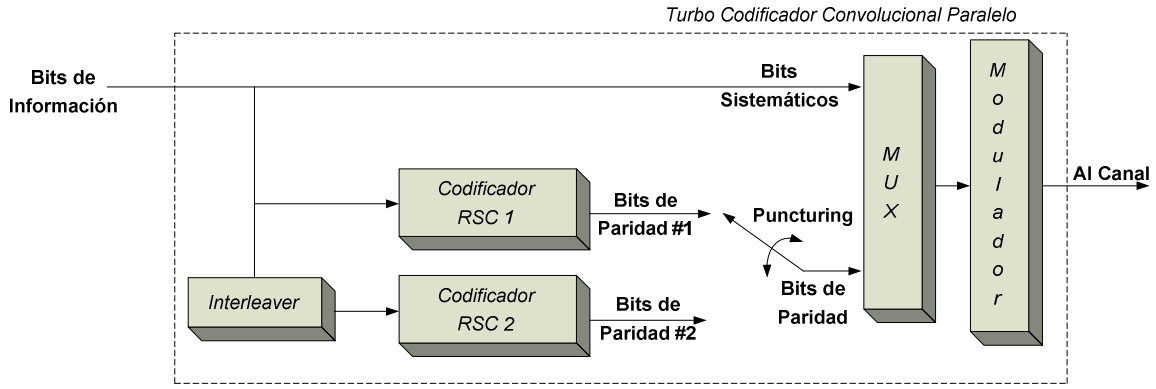


Figura 3.7. Diagrama de Bloques del Turbo Codificador. [124]

Con ello se producen los bits de paridad del primer y segundo codificador. La señal a enviar por el canal estará formada por los bits sistemáticos que coinciden con los de información y los bits de paridad de ambos componentes codificadores. Si se desea obtener una tasa de r = 1/3, todos los bits sistemáticos y de paridad generados son enviados al canal. En cambio, si la tasa deseada es r = 1/2 a los bits de paridad producidos por los componentes codificadores se les aplica un proceso de puncturing alternado, esto significa que por cada bit de información se envía un bit de paridad de cada componente codificador alternadamente. La Figura 3.7 muestra un diagrama de bloques que ejemplifica el proceso de Turbo Codificación implementado.

3.3.2.1 Bits de Información

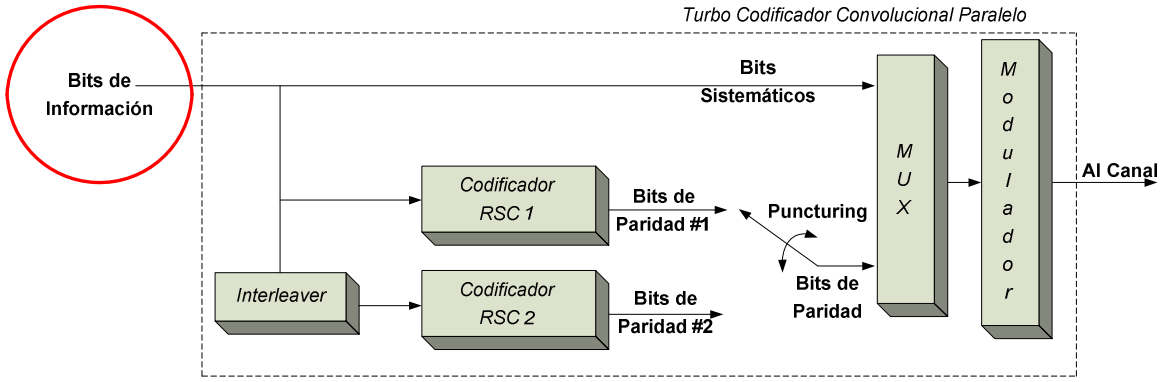


Figura 3.8. Ubicación en el diagrama de bloques de los Bits de Información.

[124] FUENTE: MOON, Todd K.; Error Correction Coding: Mathematical Methods and Algorithms; Pág: 586.

3.3.2.1.1 Descripción

Los bits de información son generados de forma aleatoria por medio de la función *randint* y representan a la trama de información sin cola. Los bits de información se almacenan en el vector ***u*** y su tamaño es igual a la longitud de la trama menos la memoria ***m*** del código (que coincide con el valor de la cola).

```
u =

     1     1     0
```

Bits de Información (Sin Cola)

Figura 3.9. Ejecución de Prueba para la Visualización de los Bits de Información Sin Cola.

Además, la información producida por dicho vector se guarda en el archivo *show0* con extensión *.mat*[125], debido a que esta información es necesaria para posteriormente ser utilizada en la turbo decodificación.

3.3.2.1.2 Implementación

Dentro de los parámetros que deben ser ingresados por los usuarios están el Tamaño de la Trama (***Long_Trama***), es decir la cantidad de bits que se desea procesar; y el Código Generador (***G***) que determina la cantidad de memoria que tiene el código.

```
% Tamaño de la Trama
Long_Trama = input(' Introduzca el tamaño de la trama: (= Información + Cola, Por
default: 400)   ');
if isempty(Long_Trama)
   Long_Trama = 400;     % Bits de información más bits de cola
end

% Código Generador
G = input(' Introduzca el código generador: ( Por default: G = [1 1 1; 1 0 1 ] )   ');
if isempty(G)
   G = [ 1 1 1; 1 0 1 ];
end

[n,K] = size(G); % Calculo tamaño Código Generador para obtener constraint length
m = K - 1; % Calculo la cantidad de memoria del código
```

Espacio de Código 3.1. Implementación del Ingreso del Tamaño de la Trama, Código Generador y Cálculo de la cantidad de Memoria del Código.

[125] ARCHIVOS .MAT: Los archivos .mat permiten guardar los datos en un formato permanente y portátil que puede volver a cargarse con facilidad luego en MATLAB.

Los bits de información al igual que todos los demás bloques del Turbo Codificador son implementados en la sub-función *Turbo_Codificador*.

```
u = randint(1, Long_Trama-m);  % Bits de información
Bits_Informacion_Sin_Cola = u
```

Espacio de Código 3.2. Implementación de los Bits de Información.

3.3.2.1.3 Interfaz Gráfica de Usuario

El desarrollo de la interfaz gráfica se hizo en Matlab R2008a, dada la familiaridad con las herramientas encontradas en el directorio *matlab\uitools* que facilitan la creación de interfaces de usuario de manera ágil y sencilla utilizando la herramienta *GUIDE*. *GUIDE* permite realizar conjuntos de ventanas, con botones, menúes, ventanas, etc; generando dos archivos con el mismo nombre para cada ventana creada: un archivo con extensión *.m* y un archivo con extensión *.fig*.

Estos archivos contienen toda la información básica de la interfaz; el archivo *.fig* contiene las características visuales de los elementos de cada ventana y el archivo *.m* contiene la parte estructural de la misma, es decir, las funciones que ejecutan cada uno de los elementos que la conforman.

Es a partir de esa función principal para cada ventana, que se hace un llamado a otras sub-funciones encargadas de realizar cálculos, operaciones o construir gráficos. En el caso de esta interfaz, realizan este llamado a los archivos *.m* que conforman las sub-funciones del modelo desarrollado, para efectuar los cálculos principales de las simulaciones.

En la interfaz se tienen ocho diferentes ventanas en las que el usuario puede trabajar con las diversas opciones planteadas para las simulaciones. En ellas se puede observar las propiedades y el desempeño de los Turbo Códigos por medio de opciones que permiten variar los parámetros propios del esquema de codificación, y los parámetros propios de la simulación, que se relacionan con la validez de los resultados. Dicha interfaz gráfica del prototipo se sintetiza en el diagrama de bloques de la Figura 3.10:

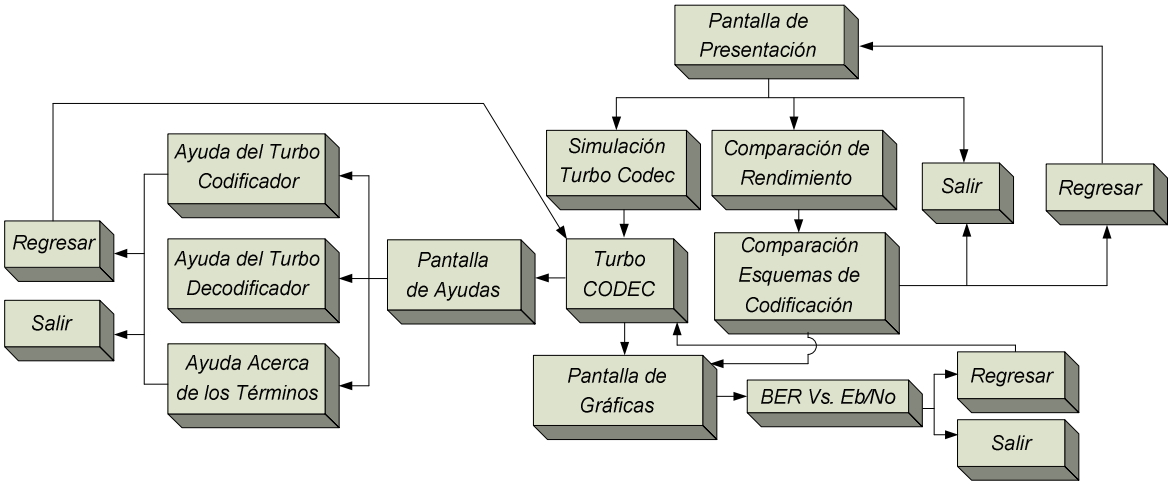


Figura 3.10. Diagrama de Bloques de la Interfaz Gráfica.

La primera ventana es una presentación que brinda ligeros datos del prototipo y de sus autores, permite al usuario elegir si se requiere la Simulación del Esquema clásico de un Turbo Codec, la Comparación de Rendimiento entre diferentes polinomios generadores y esquemas de codificación para control de errores o salir del programa, su código fuente así como el del resto de ventanas está disponible en el Anexo B. La Figura 3.11 muestra dicha ventana:

Figura 3.11. Ventana de Presentación.

El Diagrama de flujo que se muestra en la Figura 3.12 explica el funcionamiento de la ventana.

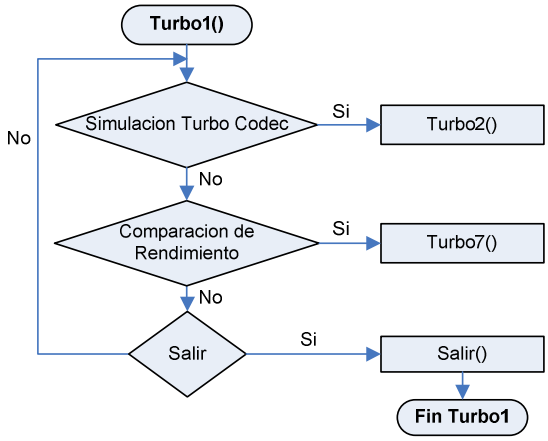


Figura 3.12. Diagrama de Flujo de la Ventana de Presentación.

Los bits de información se implementan en la segunda ventana del interfaz gráfico de usuario correspondiente al espacio *Tamaño de la Trama*, pero en este caso se ha implementado también la opción que permite ingresar la trama bit a bit en formato binario. El espacio de código que hace posible escoger entre la una u otra opción es:

```
% Tamaño de la Trama
format long
if (get(handles.trama1,'Value') == get(handles.trama1,'Max'))
    Long_Trama = str2double(get(handles.trama2, 'String'));
else
    w = str2double(get(handles.trama4, 'String'));
    aa=num2str(w);
    Long_Trama =length (aa)+ m;
end
if isnan(Long_Trama)
   errordlg('La Entrada debe ser un Número','Error');
end
```

Espacio de Código 3.3. Bits de Información en Formato Binario.

- La Figura 3.13, presenta un mensaje de error, que se muestra cuando no se han ingresado números o los campos están vacíos en la ventana de simulación del Turbo Codec, por lo que el usuario deberá ingresar un dato correcto o llenar los campos vacíos.

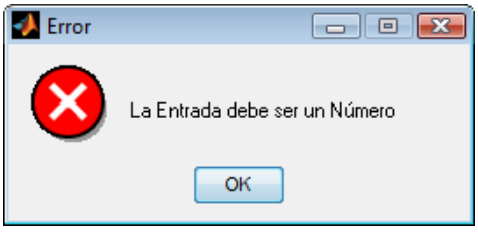


Figura 3.13. Mensaje de Error en el Ingreso de Datos.

La creación del *radio button* y el cuadro de texto para el ingreso del tamaño de la trama está definido por las siguientes líneas de programa:

```
% --- Executes on button press in trama1.
function trama1_Callback(hObject, eventdata, handles)

function trama2_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function trama2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Espacio de Código 3.4. Bits de Información en Función del Tamaño de la Trama.

Se debe recordar que el *Tamaño de la Trama* debe ser ingresado como un número entero positivo. La Figura 3.14, muestra los resultados del espacio de código implementado.

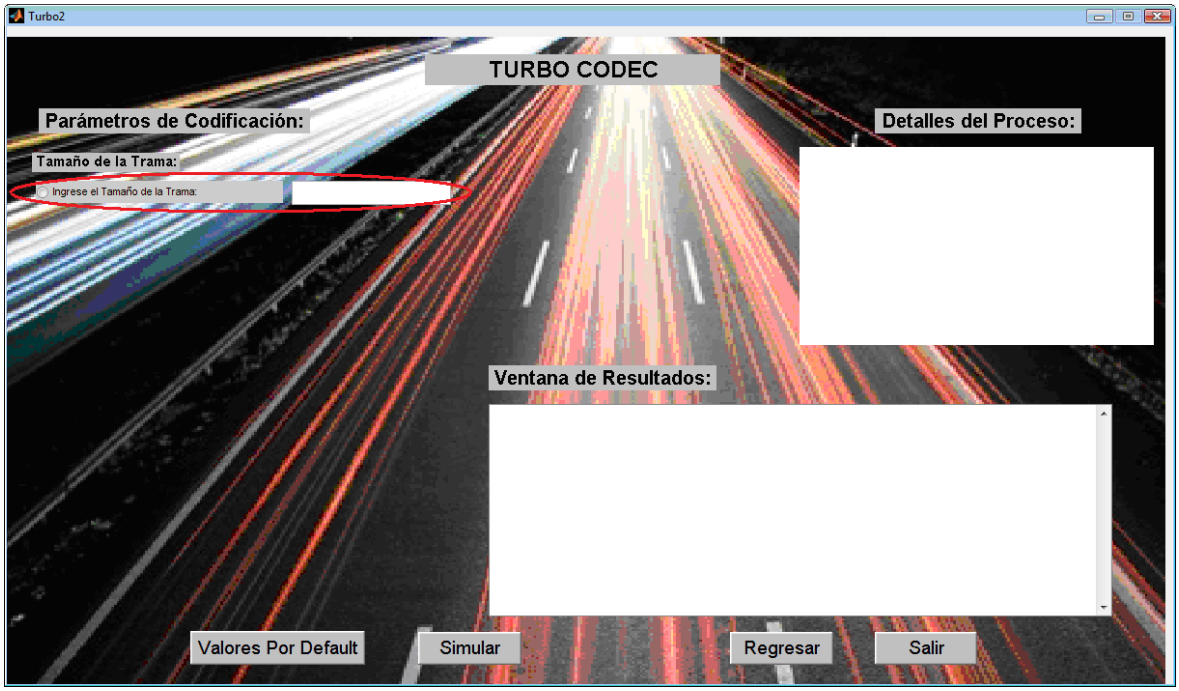


Figura 3.14. Implementación del Ingreso del Tamaño de la Trama en el Turbo Codec.

La creación del *radio button* y el cuadro de texto para el ingreso de la trama bit a bit está definida por las siguientes líneas de programa:

```
% --- Executes on button press in trama3.
function trama3_Callback(hObject, eventdata, handles)

function trama4_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function trama4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Espacio de Código 3.5. Bits de Información Ingresando la Trama Bit a Bit en Formato Binario.

La Figura 3.15 muestra la implementación gráfica en *Turbo2,* del Espacio de Código 3.5.

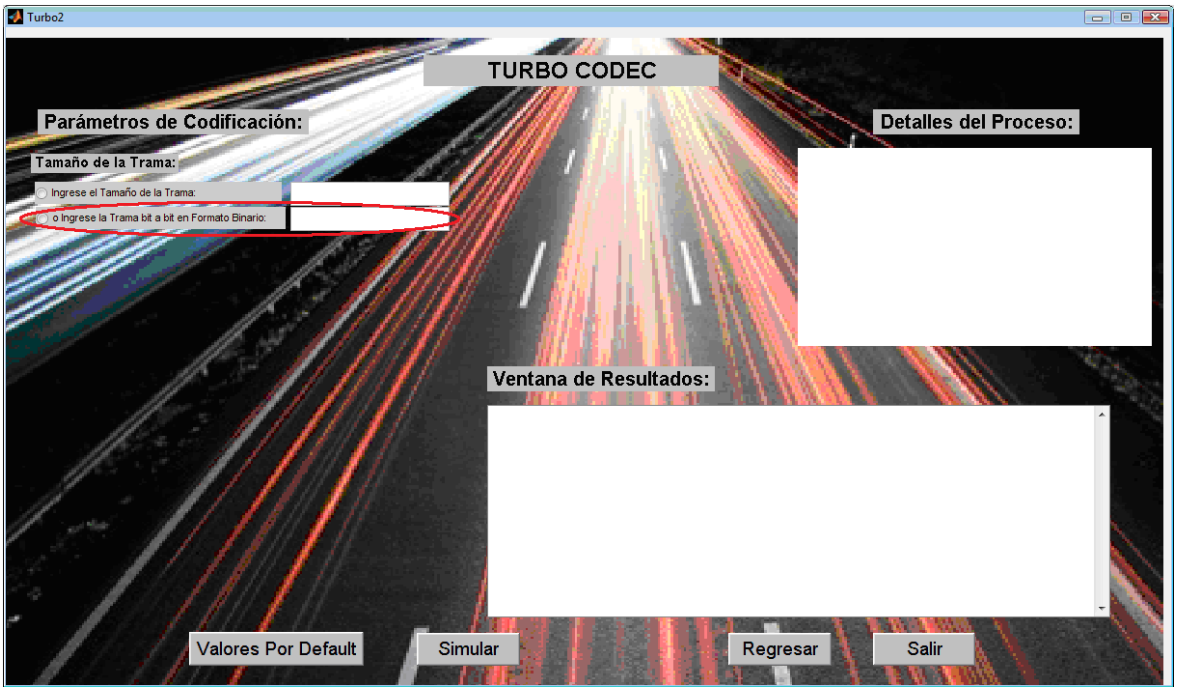


Figura 3.15. Implementación del Ingreso de la Trama Bit a Bit en Formato Binario en el Turbo Codec.

La ventana de Simulación del Turbo Codec está acompañada en la parte inferior de botones que permiten ejecutar diferentes procesos y órdenes.

El botón *Valores por Default* permite cargar al modelo de simulación con parámetros predefinidos. Estos valores predefinidos son valores que le permiten al Turbo Codec tener un buen rendimiento y que se han obtenido del proceso de pruebas y ensayos, para el caso del Turbo Codificador, estos se pueden observar en la Tabla 3.1.

Parámetro	Valor por Default
Tamaño de la Trama	1024
Perforación de Código	Unpunctured
Código Generador	[111, 101]

Tabla 3.1. Resumen de Parámetros de Simulación con Valores por Default.

Una vez que se han escogido los parámetros para el proceso de codificación o se ha seleccionado la opción de valores por *Default*, el botón *Simular* permite ejecutar los procesos del Turbo Codec. Mientras se está ejecutando el proceso de simulación, un *waitbar* (barra de espera) permite visualizar la evolución del proceso, y se inhabilitan los demás botones de control.

- La Figura 3.16, muestra un mensaje de espera, mediante una barra durante el procesamiento de datos; que se presenta cuando se está ejecutando el Turbo Codec.

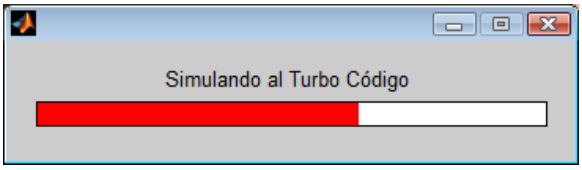


Figura 3.16. Barra de Espera para Procesamiento de Datos.

El Espacio de Código 3.6 es el que hace posible esto y se encuentra en la segunda ventana del interfaz gráfico de usuario:

```
% Barra de espera mientras el proceso se ejecuta
handle=waitbar(0,'Simulando al Turbo Código');

% Desactivación de los componentes y botones mientras se simula el turbo codec
set(handles.detalle,'String',n1);
set(handles.Default,'Enable','off');
set(handles.Salir,'Enable','off');
set(handles.Regresar,'Enable','off');
set(handles.Graficar,'Enable','off');
set(handles.detalle,'string',[]);
```

Espacio de Código 3.6. Waitbar.

Los botones *Regresar* y *Salir*, permiten retroceder a la ventana de presentación o salir definitivamente de la ejecución del interfaz. La ventana de Simulación del Turbo Codec, además muestra espacios en los que se visualizarán los *Detalles*

del Proceso y la *Ventana de Resultados*, creados en el *GUIDE* de Matlab mediante un *edit text* y un *listbox*, respectivamente.

En *Detalles del Proceso*, se presenta un resumen de los parámetros que han sido escogidos por el usuario: Algoritmo de Decodificación, Tamaño de la Trama, Código Generador, Perforación del Código, Número de Iteraciones, y la E_b/N_o. Mientras que los valores obtenidos de la simulación se presentan en la *Ventana de Resultados*, con la finalidad de que el usuario pueda observar de manera cuantitativa cuales son los valores de cada uno de los bloques que conforman al Turbo Codec y el BER obtenido luego de este proceso con la finalidad de observar el rendimiento del código evaluado.

3.3.2.2 Bits Sistemáticos

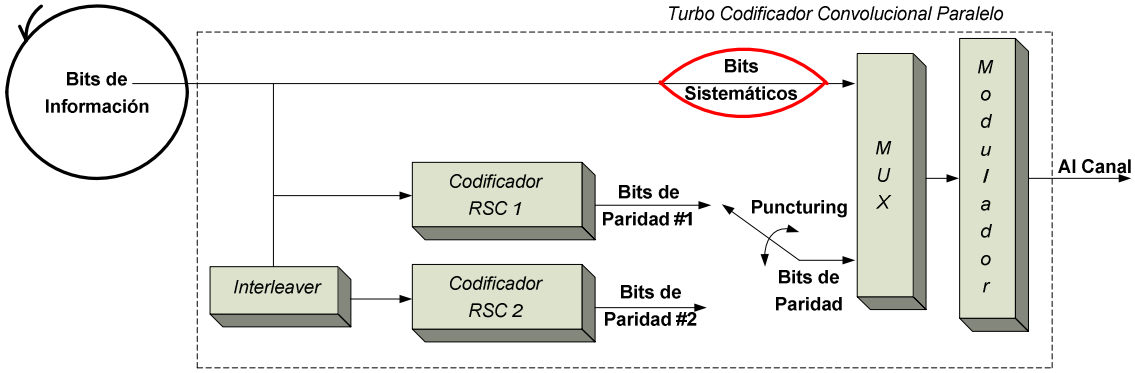


Figura 3.17. Ubicación en el diagrama de bloques de los Bits Sistemáticos.

3.3.2.2.1 Descripción

[126]Un código convolucional es *Sistemático* si la secuencia de información no es modificada, en la secuencia codificada producida por el codificador. Mientras que un codificador convolucional es *Recursivo* si existe retroalimentación de los registros de memoria del codificador hacia la entrada del mismo. [127]El codificador convolucional sistemático recursivo (RSC) se obtiene a partir del codificador convolucional no sistemático no recursivo (NSNR, convencional) por la

[126] FUENTE: INGVARSSON, Per Ola; SVENELL, Henrik; Master Thesis: Error Performance of Turbo Codes; Pág: 12.
[127] FUENTE: HUANG, Fu-hua; Master's Thesis: Evaluation of Soft Output Decoding for Turbo Codes; Págs: 24-26.

realimentación de una de estas salidas codificadas hacia su entrada. La Figura 3.18 muestra un codificador convolucional convencional.

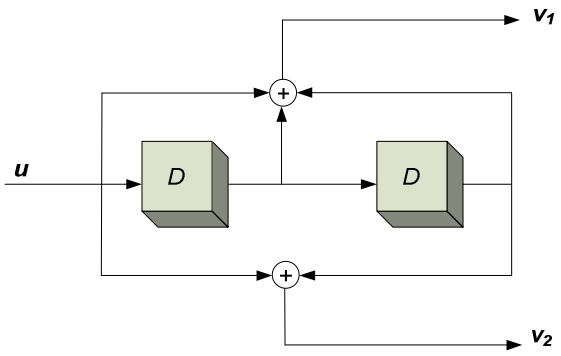


Figura 3.18. Codificador Convolucional Convencional (No Sistemático y No Recursivo NSNR) empleado por el Ejemplo para un polinomio G = [111,101]. [128]

El codificador convolucional convencional (no sistemático y no recursivo NSNR) es representado por los polinomios generadores g_0 y g_1 y puede ser equivalentemente representado en una forma más compacta como $G = [g_0, g_1]$. El codificador RSC de este codificador convolucional convencional es representado como $G = [1, g_1/g_0]$ donde la primera salida (representada por g_0) es realimentada (*feedback*[129]) a la entrada del mismo, el 1 denota la salida sistemática y g_1 denota la salida de *feedforward*[130]. La Figura 3.19 muestra el Codificador RSC.

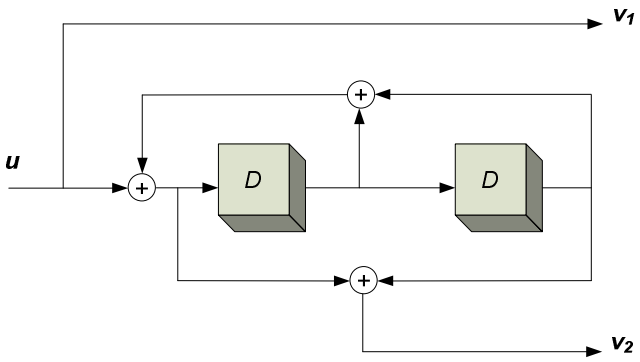


Figura 3.19. Codificador RSC empleado por el Ejemplo para un polinomio G = [111,101]. [131]

[128] FUENTE: HUANG, Fu-hua; Master's Thesis: Evaluation of Soft Output Decoding for Turbo Codes; Pág: 25.
[129] FEEDBACK: o realimentación, es un proceso por el que una cierta proporción de la señal de salida de un sistema se redirige de nuevo a la entrada. De uso frecuente para el control dinámico del sistema.
[130] FEEDFORWARD: o proalimentación, es un tipo de sistema que reacciona a los cambios en su entorno, para mantener algún estado concreto del sistema. Un sistema que exhibe este tipo de comportamiento responde a las alteraciones de manera predefinida, en contraste con los sistemas realimentados.
[131] FUENTE: HUANG, Fu-hua; Master's Thesis: Evaluation of Soft Output Decoding for Turbo Codes; Pág: 26.

Buenos códigos pueden ser obtenidos mediante la realimentación del codificador RSC a un polinomio generador primitivo, ya que el polinomio primitivo genera secuencias de longitud máxima, que añade aleatoriedad al Turbo Código. [132]Los códigos sistemáticos son preferidos sobre los códigos no sistemáticos porque permiten una rápida búsqueda, además requieren menos hardware para la codificación. Otra propiedad importante de los códigos sistemáticos es que no son catastróficos, lo que significa que los errores no se propagan catastróficamente. Todas estas propiedades hacen a los códigos sistemáticos muy deseables.

Para el ejemplo de la simulación, la sub-función *Turbo_Codificador* toma como entrada los bits de información ***u*** y configura el codificador RSC constituido por los polinomios generadores g_0 = [1 1 1] y g_1 = [1 0 1]. Los Bits Sistemáticos son el resultado de agregar cola a los bits de información, almacenándose dichos bits en el vector ***v1***. El tamaño de la cola se calcula como el *constraint length* del código generador menos 1.

$$cola = K - 1 \qquad \textbf{(Ecuación 3.2)}$$

Donde K = es el *constraint length* del código generador g

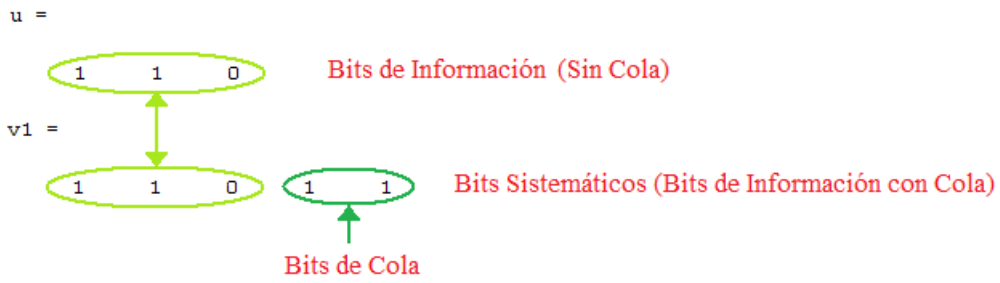


Figura 3.20. Ejecución de Prueba para la Visualización de los Bits Sistemáticos.

3.3.2.2.2 Implementación

Para la implementación de los bits sistemáticos, se debe realizar la configuración del Trellis por medio de la función *poly2trellis(K, F(2,1), F(1,1))* en modo *feedback*, para hacer esto se toma como entrada el *constraint length K* del código

[132] FUENTE: LANGTON, Charan; Intuitive Guide to Principles of Communications: Coding and Decoding with Convolutional Codes; Pág: 5.

generador, y los polinomios g_0 para la realimentación (representado en la sintaxis por *F(1,1)*) y g_1 para el *feedforward* (que en la sintaxis de la función está representado por *F(2,1)*).

```
cad_G = num2str(G); % Conversión código generador a cadena de caracteres
F = str2num(dec2base(bin2dec(cad_G),8)); % Conversión cadena de caracteres binaria a
                                         % octal
t = poly2trellis(K, F(2,1), F(1,1)); % Definición del Trellis
```

Espacio de Código 3.7. Configuración del Trellis en Modalidad Feedback.

Una vez configurado el Trellis, se codifica el mensaje de entrada ***u*** por medio de un codificador RSC mediante la función *convenc* (ya descrita en el apartado 2.4.2.2.2); como resultado de este proceso se devuelve la secuencia de bits de información codificados y se los carga en el vector ***v2_sin_cola*** y el estado final (vector ***estado_final***) del codificador después de procesar el mensaje***,*** dicho estado final será utilizado para realizar el cálculo de la cola.

```
% Codificador RSC 1
[v2_sin_cola  estado_final] = convenc(u, t); % Codifico a los Bits de Información y
                                             % Termino el Trellis del primer
                                             % codificador, la realimentacion debe
                                             % incluir a todos los estados.
```

Espacio de Código 3.8. Codificación RSC de los Bits de Información Sin Cola.

[133]Al igual que en los codificadores convolucionales, el turbo codificador se debe reinicializar para procesar una palabra mensaje diferente. La presencia del interleaver en la estructura del Turbo Codificador, hace que no sea práctico colocar bits de relleno al final de la palabra mensaje, pues existiría un retraso adicional no justificado en el interleaver (se estarían reordenado “ceros”). En su lugar lo que se hace es luego que los bits de información han sido codificados, y por ende el interleave queda vacío, hay que centrarse en los codificadores constitutivos (*RSC* 1 y *RSC* 2) y limpiarlos independientemente por cualquier método conocido, para el presente caso se usó la *Terminación Trellis* para cumplir con este cometido.

[133] FUENTE: CAMPANELLA, Humberto; NARDUCCI, Margarita; Técnicas y Tecnologías de Comunicaciones Móviles 3G; Pág: 61.

La *Terminación Trellis* reinicia los codificadores constitutivos en forma simétrica, toma los bits que quedan en los registros, luego que se ha terminado la palabra de mensaje, e introduce “automáticamente” *m* ceros al codificador para depurar a los codificadores.

Los vectores que van a contener la cola como la cola codificada deben ser inicializados a cero y tomando en cuenta la cantidad de memoria ***m*** del código, se determina la longitud de la cola y se hace el cálculo de la misma. La generación de estos bits redundantes (cola), se la realiza utilizando convoluciones módulo-2; la convolución en módulo-2 representa una operación XOR. Adicionalmente a los bits del vector ***cola*** calculados se les codifica nuevamente en modo *feedback* por medio de *convenc,* tomando en cuenta el estado final del codificador que ya se había calculado y se los contiene en el vector ***cola_codificada***.

```
cola = zeros(1,m); % Inicializa la cola
cola_codificada = zeros(1,m); % Inicializa la cola codificada

% Conversión de octal a cadena de caracteres binaria en las conexiones de realimentacion
del codificador
F_dec = oct2dec(F);
F_bin = dec2bin(F_dec, m);

% Generación de la cola y cola codificada
for n = 1:m
    estado_final_binario = dec2bin(estado_final, m);
    j = 0;
    for i = 1:m
        if estado_final_binario(i) == '1' & F_bin(i+1) == '1' % Si F_bin(i+1) == '1',
                                                              % ese registro esta
                                                              % realimentado;
                                                              % F_bin(1) es la conexion
                                                              % de la entrada al
                                                              % codificador
           j = j + 1;
        end
    end
    cola(n) = mod(j,2); % Generación de la cola, si el estado tiene cantidad de '1' par,
                        % entonces a la suma modulo-2(compuerta XOR) debe entrar un '0'.
    estado_anterior = estado_final; % Ingreso al codificador la cola
   [cola_codificada(n) estado_final] = convenc(cola(n),t, estado_anterior); % Generación
                                                                            % de la cola
                                                                            % codificada
end
```

Espacio de Código 3.9. Implementación de la Cola.

Los bits de información inicial ***u*** se convierten en bits sistemáticos si se les agrega una cola (la misma que ya fue calculada en el Espacio de Código 3.9), esto se logra concatenando los vectores ***u*** y ***cola*** en un nuevo vector ***v1***.

```
% Genero los bits sistematicos, agregando cola a los bits de información
v1 = [u, cola];
Bits_Sistematicos = v1
```

Espacio de Código 3.10. Implementación de los Bits Sistemáticos.

3.3.2.3 Bits de Paridad # 1

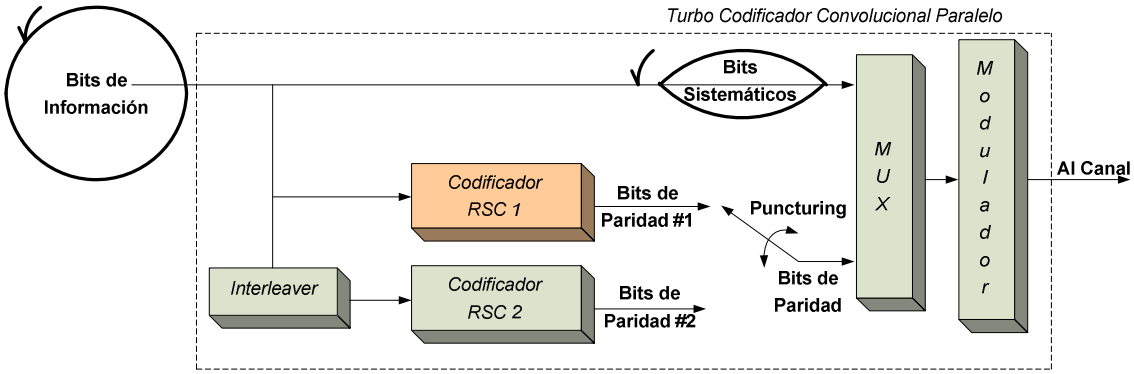


Figura 3.21. Ubicación en el diagrama de bloques de los Bits de Paridad # 1.

3.3.2.3.1 Descripción

Los bits de paridad # 1 son almacenados en el vector ***v2***, y son bits que resultan de una de las salidas del codificador RSC 1.

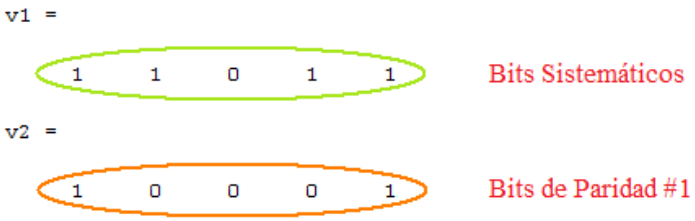


Figura 3.22. Ejecución de Prueba para la Visualización de los Bits de Paridad #1.

3.3.2.3.2 Implementación

A los bits de información codificados RSC y almacenados en ***v2_sin_cola*** se le agrega la cola codificada que se encuentra en el vector ***cola_codificada*** y mediante la concatenación de estos dos vectores se obtiene la secuencia final del vector ***v2***, que corresponde a los bits de paridad #1.

```
% Genero los bit de paridad 1, agregando a los bits del primer codificador la cola
codificada
v2 = [ v2_sin_cola, cola_codificada];
Bits_Paridad_1 = v2
```

Espacio de Código 3.11. Implementación de los Bits de Paridad # 1.

3.3.2.3.3 Interfaz Gráfica de Usuario

Para el caso del *Código Generador* se presentan dos opciones posibles para la simulación, el escoger entre polinomios generadores preasignados para varias tecnologías como: ADSL [1101, 1111][134], GSM [10011, 11011][135], D-AMPS [1101,1001][136], UMTS [1011, 1101][137], CDMA2000 [1011, 1111][138], WIMAX [1111001, 1011011][139] y uno por DEFAULT [111, 101][140]; o ingresar el código generador en formato octal. La primera opción se muestra en la Figura 3.23:

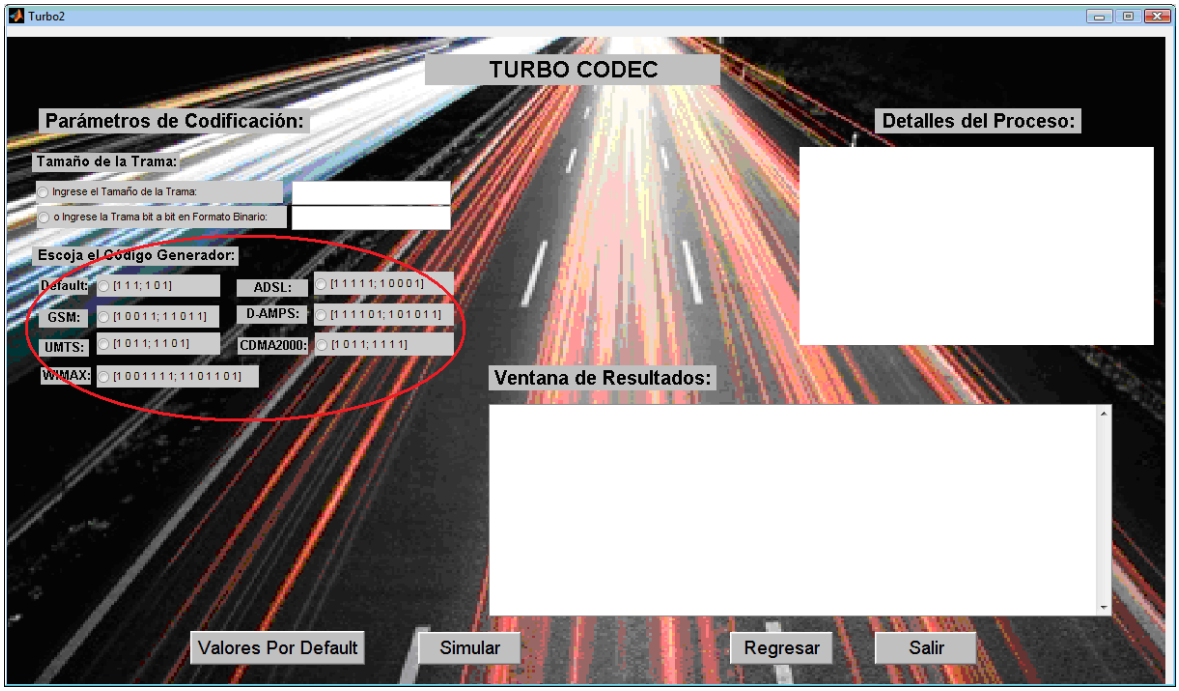


Figura 3.23. Implementación del Ingreso del Código Generador en el Turbo Codec.

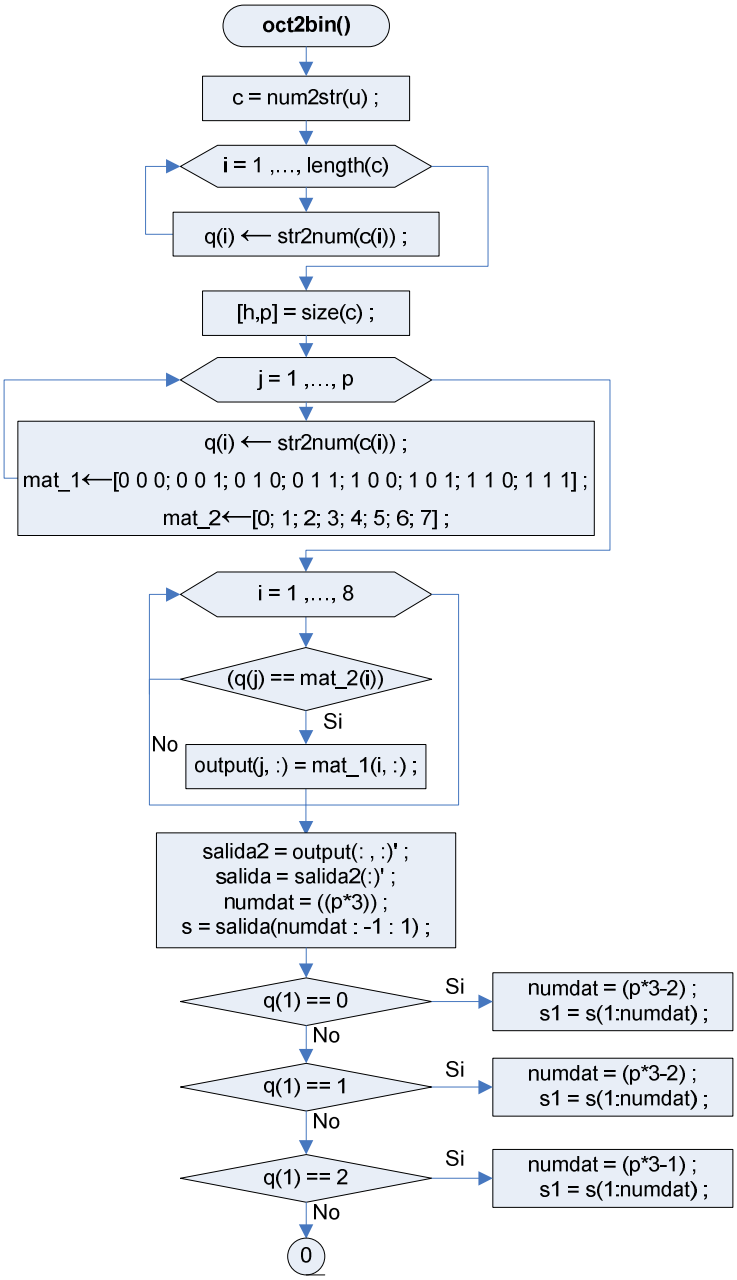
---

[134] FUENTE: KEATTISAK, Sripimanwat; Turbo Code Applications: A Journey From a Paper to Realization; Pág: 70.
[135] FUENTE: EBERSPÄCHER, Jörg; VÖGEL, Hans-Jörg; BETTSTETTER, Christian; HARTMANN, Christian; GSM-Architecture, Protocols and Services; Pág: 105.
[136] FUENTE: HARTE, Lawrence J.; SMITH, Adrian D.; JACOBS, Charles A.; Is-136 Tdma Technology, Economics and Services; Pág: 85.
[137] FUENTE: European Telecommunications Standards Institute (ETSI); Universal Mobile Telecommunications System (UMTS): Multiplexing and Channel Coding (FDD) – Release 4; Pág: 15.
[138] FUENTE: Third Generation Partnership Project 2 (3GPP2); Physical Layer Standard for cdma2000 Spread Spectrum Systems - Release C; Pág: 88.
[139] FUENTE: MAN CHEUK, Ng; MURALIDARAN, Vijayaraghavan; NIRAV, Dave; ARVIND; From WiFi to WiMAX: Techniques for High-Level IP Reuse across Different OFDM Protocols; Pág: 4.
[140] FUENTE: HUANG, Fu-hua; Master's Thesis: Evaluation of Soft Output Decoding for Turbo Codes; Pág: 26.

Una sub-función creada para la manipulación de los datos en la ventana de simulación del Turbo Codec es *oct2bin*, la misma que permite ingresar al usuario en forma octal (modo más común) el polinomio generador. Esta función convierte un número de sistema octal a sistema binario y devuelve un vector de salida ordenado, desde el bit menos significativo hacia el más significativo.

3.3.2.3.4 Diagrama de Flujo

El diagrama de flujo de la función *oct2bin* se muestra en la Figura 3.24:



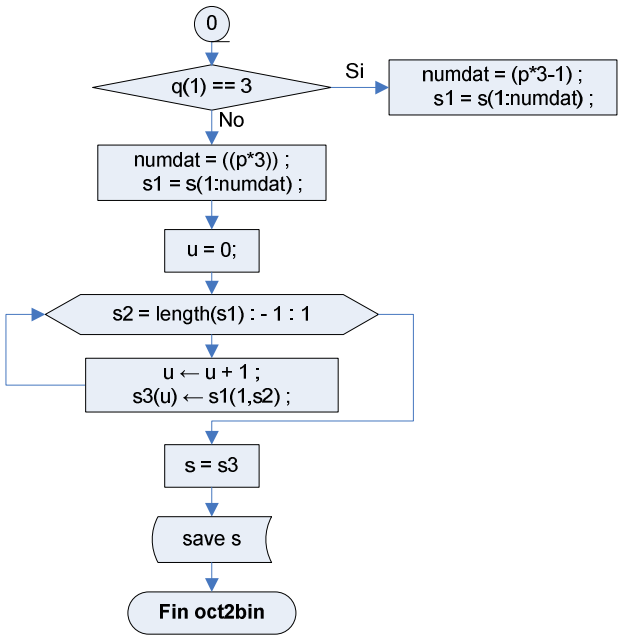


Figura 3.24. Diagrama de Flujo de la Función oct2bin.

3.3.2.3.5 Implementación

```
function oct2bin = oct2bin (u)
c = num2str(u);
for i = 1:length(c)
    q(i) = str2num(c(i));
end

% Esta función convierte un número de sistema octal a sistema binario
[h,p] = size(c);
for j = 1:p
    mat_1 = [0 0 0;0 0 1;0 1 0;0 1 1;1 0 0;1 0 1;1 1 0;1 1 1];
    mat_2=[0;1;2;3;4;5;6;7];
    for i = 1:8
        if(q(j)==mat_2(i))
          output(j,:)=mat_1(i,:);
        end
    end
end

% Coloca al vector desde el menos significativo al más significativo
salida2 = output(:,:)';
salida = salida2(:)';
numdat=((p*3));
s = salida(numdat:-1:1);
if  q(1)== 0
    numdat=(p*3-2);
    s1 = s(1:numdat);
elseif q(1)== 1
    numdat=(p*3-2);
    s1 = s(1:numdat);
elseif q(1)== 2
    numdat=(p*3-1);
    s1 = s(1:numdat);
elseif q(1)== 3
    numdat=(p*3-1);
    s1 = s(1:numdat);
else
    numdat=((p*3));
    s1 = s(1:numdat);
end
```

```
u = 0;
for s2 = length(s1):-1:1
    u = u+1;
    s3(u) = s1(1,s2);
end
s = s3;
% Graba los valores para poder mostrarlos
save s
```

Espacio de Código 3.12. Función oct2bin.

La Figura 3.25, muestra la implementación del la opción que permite ingresar los polinomios generadores en formato octal:

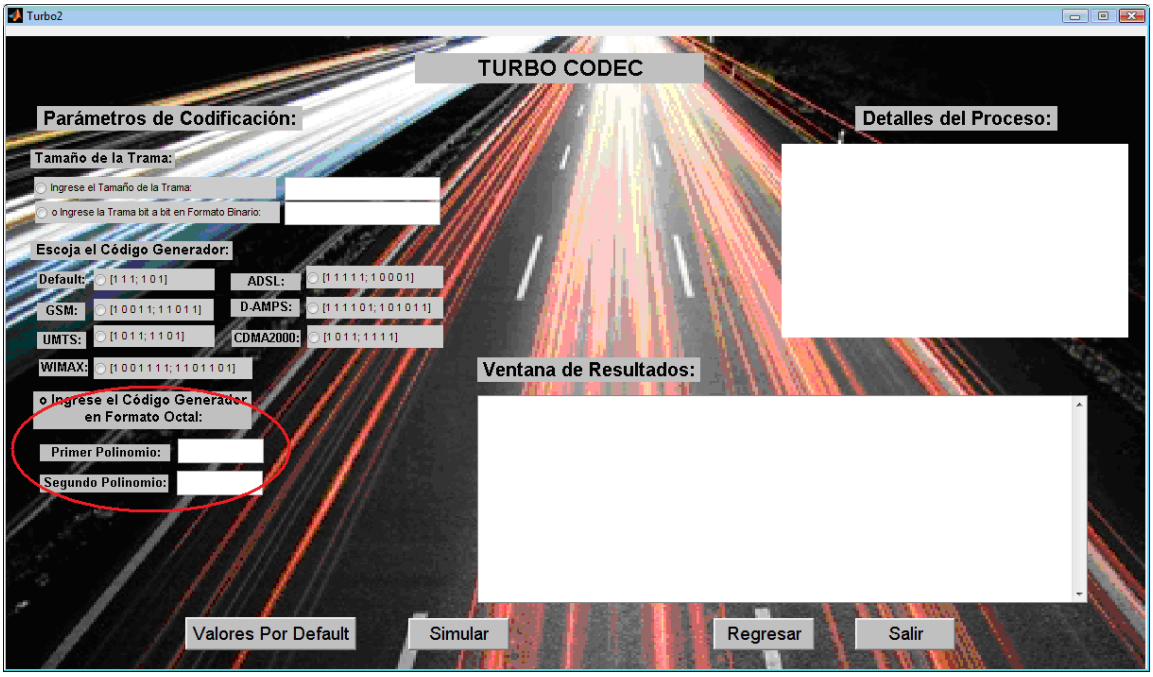


Figura 3.25. Implementación del Ingreso del Código Generador en Formato Octal en el Turbo Codec

3.3.2.4 Interleaver

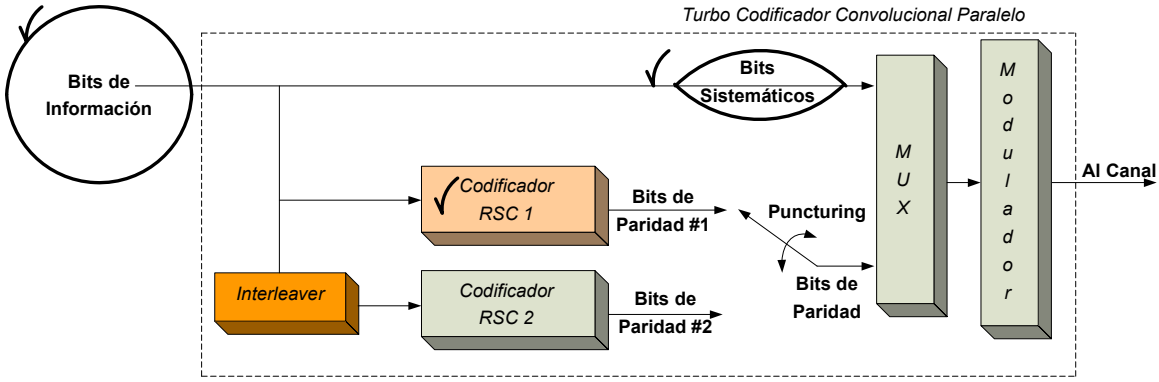


Figura 3.26. Ubicación en el diagrama de bloques del Interleaver.

3.3.2.4.1 Descripción

[141]El interleaver es un dispositivo que permuta la secuencia de entrada en forma determinística, es decir, cambia el orden temporal de los bits pero sin modificar su contenido. Una de las principales razones por las cuales el uso del interleaver es necesario radica en que, en medios de comunicación como el inalámbrico, el ruido no afecta por igual toda la cadena de información debido a que el modelo estadístico del ruido no sigue un comportamiento gaussiano. La función del interleaver es reordenar los bits de tal forma que el efecto ruido, y la posterior reorganización de la información, la probabilidad de error tenga un patrón más uniforme para todos los bits transmitidos.

[142]Los interleavers pueden dividirse en dos grupos:

- *Interlerleavers Convolucionales*: que son capaces de trabajar con los bits de información de manera secuencial, y utilizados en una clase de Turbo Codificación conocida como *Stream Oriented Turbo Codes,* generalmente empleada en aplicaciones de conmutación de circuitos como la Red de Telefonía Pública Conmutada - PSTN.
- *Interleavers de Bloque*: que trabajan con los bits de información por *bloques* y no de manera secuencial.

A continuación se exponen algunos de los interleavers de bloque más utilizados en la Turbo Codificación:

3.3.2.4.2 Interleaver "Fila-Columna"

Consiste en una memoria en la que la información es escrita a lo largo de las filas y leída a lo largo de las columnas, por lo que este es el interleaver más simple que se conoce. La Tabla 3.2 ilustra un ejemplo de un interleaver de este tipo.

---

[141] FUENTE: CAMPANELLA, Humberto; NARDUCCI, Margarita; Técnicas y Tecnologías de Comunicaciones Móviles 3G; Págs: 60-61.
[142] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Págs: 73-77.

X_1	X_2	X_3
X_4	X_5	X_6
X_7	X_8	X_9
X_{10}	X_{11}	X_{12}
X_{13}	X_{14}	X_{15}
X_{16}	X_{17}	X_{18}
X_{19}	X_{20}	X_{21}

Tabla 3.2. Ejemplo de la Memoria de un Interleaver "Renglón-Columna".

La salida de este interleaver estará dada por: X_1, X_4, X_7, X_{10}, X_{13}, X_{16}, X_{19}, X_2, X_5…

3.3.2.4.3 Interleaver "Espiral"

Consiste en una memoria en la que la información es escrita como en la Tabla 3.2 pero leída en forma diagonal. La salida de este interleaver estará dada por: X_{19}, X_{17}, X_{15}, X_{10}, X_8, X_6, X_1, X_{20}, X_{18}, X_{13}, ...

3.3.2.4.4 Interleaver "Impar-Par"

Un interleaver "Impar-par" es un interleaver "Fila-Columna" cuya memoria tiene un número impar de filas y un número impar de columnas. Para la secuencia de información que se está utilizando la memoria del interleaver quedaría escrita de acuerdo a la Tabla 3.3:

X_1	X_2	X_3	X_4	X_5
X_6	X_7	X_8	X_9	X_{10}
X_{11}	X_{12}	X_{13}	X_{14}	X_{15}

Tabla 3.3. Ejemplo de la Memoria de un Interleaver "Impar-Par" 3 x 5.

Suponiendo que la secuencia de salida de este interleaver es codificada por el RSC 2 y que se almacenan los bits codificados que estén en las posiciones pares, entonces resulta la Tabla 3.4. Donde la primera fila corresponde al

reordenamiento de las columnas de la Tabla 3.3 y la segunda fila (representada por la variable z) toma en cuenta los bits codificados en posiciones pares.

X_1	X_6	X_{11}	X_2	X_7	X_{12}	X_3	X_8	X_{13}	X_4	X_9	X_{14}	X_5	X_{10}	X_{15}
-	Z_6	-	Z_2	-	Z_{12}	-	Z_8	-	Z_4	-	Z_{14}	-	Z_{10}	-

Tabla 3.4. Bits codificados en las Posiciones Pares.

En las Tablas 3.3 y 3.4, los 15 bits de información tienen un bit codificado asociado a cada uno de ellos, lo cual se traduce en que la potencia de codificación está uniformemente distribuida. Las dos salidas $\mathbf{v_1}$ y $\mathbf{v_2}$-$\mathbf{v_3}$ del Turbo Codificador considerado estarán dadas por la Tabla 3.5:

v_1	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}	X_{12}	X_{13}	X_{14}	X_{15}
v_2 - v_3	Y_1	Z_6	Y_3	Z_2	Y_5	Z_{12}	Y_7	Z_8	Y_9	Z_4	Y_{11}	Z_{14}	Y_{13}	Z_{10}	Y_{15}

Tabla 3.5. Salidas del Turbo Codificador utilizando un Interleaver “Impar-Par” 3 x 5.

3.3.2.4.5 Interleaver “Pseudo-Aleatorio”

Un número pseudo-aleatorio es un número generado en un proceso que parece producir números al azar, pero no lo hace realmente. Las secuencias de números pseudo-aleatorios no muestran ningún patrón o regularidad aparente desde un punto de vista estadístico, a pesar de haber sido generadas por un algoritmo completamente determinista, en el que las mismas condiciones iniciales producen siempre el mismo resultado. Los mecanismos de generación de números aleatorios que se utilizan en la mayoría de los sistemas informáticos son en realidad procesos pseudo-aleatorios. Un interleaver pseudo-aleatorio, toma la secuencia de información y la reordena en una manera pseudo-aleatoria, repetible y ajustada, la misma que es escrita en su memoria y posteriormente es leída de la memoria columna por columna.

Los interleavers son diseñados en base a las exigencias del sistema, para tamaños pequeños de bloque y una relación E_b/N_0 baja, un interleaver “Impar-par”

ofrece un mucho mejor desempeño que un interleaver “Pseudo-aleatorio” y viceversa (para una relación E_b/N_0 alta).

[143]Cuando el Turbo Codificador tiene dentro de sus salidas una salida sistemática, es decir una salida exactamente igual a los bits de entrada, se recomienda utilizar un interleaver pseudo-aleatorio para que el codificador aparezca como aleatorio ante el canal, por esta razón para la implementación en Matlab se ha escogido al interleaver pseudo-aleatorio. [144]Matlab R2008a posibilita la implementación de un interleaver pseudo-aleatorio mediante la función ***randintrlv***, esta elige una tabla de permutación aleatoria, utilizando la entrada inicial de estado que se proporciona; la permutación de la secuencia se almacena en una variable. Para el ejemplo seleccionado la sintaxis de la función estaría dada por *randintrlv(v1,104)*, donde la secuencia a desordenar con el objeto de evitar el desvanecimiento profundo en la transmisión por el canal, es el vector almacenado en ***v1*** que contiene a los bits sistemáticos**,** y el estado inicial es ***104***, siendo este último una secuencia única con la que se desordena el vector, de tal manera que se pueda recuperar el mensaje original en caso de tener que hacerlo; la nueva secuencia con interleaving de entrada al segundo codificador se almacena en el vector ***v1_int***. Esta función *randintrlv* usa el algoritmo Mersenne Twister elaborado por Nishimura y Matsumoto mediante la función *rand*. Adicionalmente, el Espacio de Código 3.13 realiza el mapeo de dicho interleaver, es decir encuentra las posiciones en las que se localizaban inicialmente los bits sistemáticos**.**

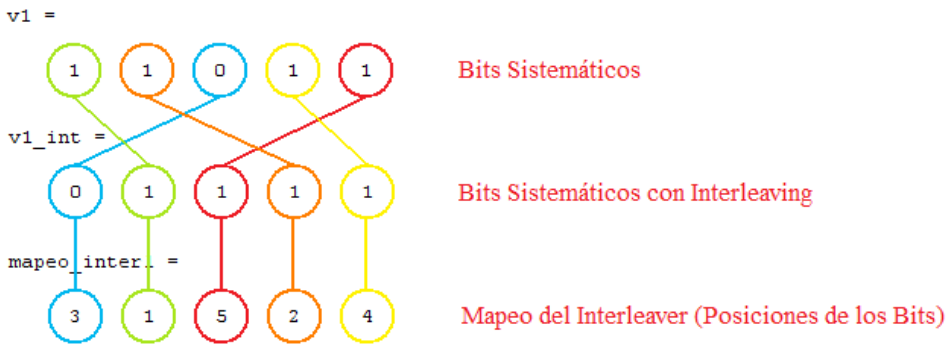


Figura 3.27. Ejecución de Prueba para la Visualización de los Bits Sistemáticos con Interleaving.

[143] FUENTE: CAMPANELLA, Humberto; NARDUCCI, Margarita; Técnicas y Tecnologías de Comunicaciones Móviles 3G; Pág: 60.
[144] FUENTE: THE MATHWORKS; Communications Toolbox™ 4: User’s Guide; Sección: 7, Pág: 6.

3.3.2.4.6 Implementación

```
% Realizo el interleaving de entrada al segundo codificador
v1_int(1,:) = randintrlv(v1,104);
Bits_Sistematicos_Con_Interleaving = v1_int
mapeo_trama = (1:Long_Trama);
mapeo_interl = randintrlv(mapeo_trama,104);
Mapeo_Interleaver = mapeo_interl
```

Espacio de Código 3.13. Implementación del Interleaver y Mapeo de Interleaver.

3.3.2.5 Bits de Paridad # 2

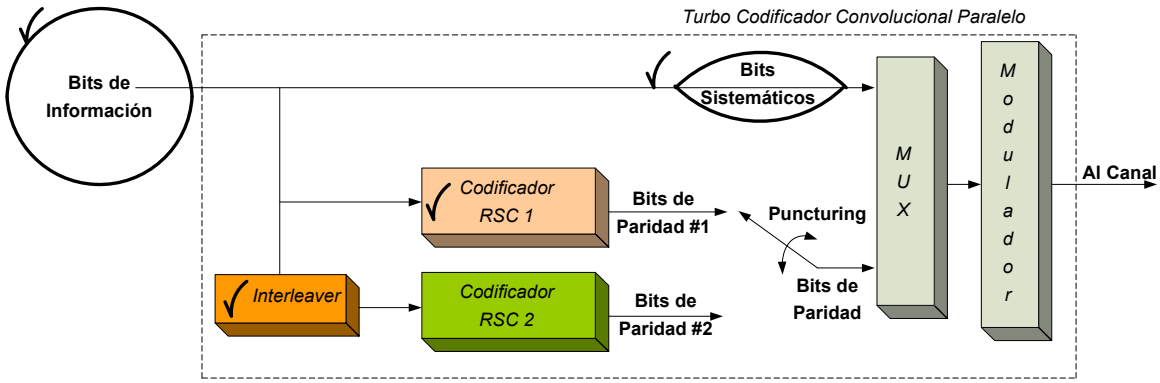


Figura 3.28. Ubicación en el diagrama de bloques de los Bits de Paridad # 2.

3.3.2.5.1 Descripción

Los bits de paridad # 2 son almacenados en el vector ***v3***, y resultan de pasar a los bits sistemáticos con interleaving (almacenados en el vector ***v1_int***) por el codificador RSC 2.

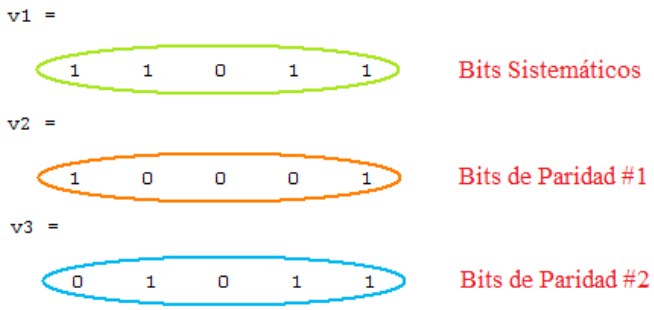


Figura 3.29. Ejecución de Prueba para la Visualización de los Bits de Paridad #2.

3.3.2.5.2 Implementación

Se utiliza la función *convenc* nuevamente, para la implementación del codificador RSC 2, con la misma configuración del Trellis.

```
% Codificador RSC 2
v3 = convenc(v1_int, t); % Genero los bit de paridad 2
Bits_Paridad_2 = v3
```

Espacio de Código 3.14. Implementación de los Bits de Paridad # 2.

3.3.2.6 Puncturing

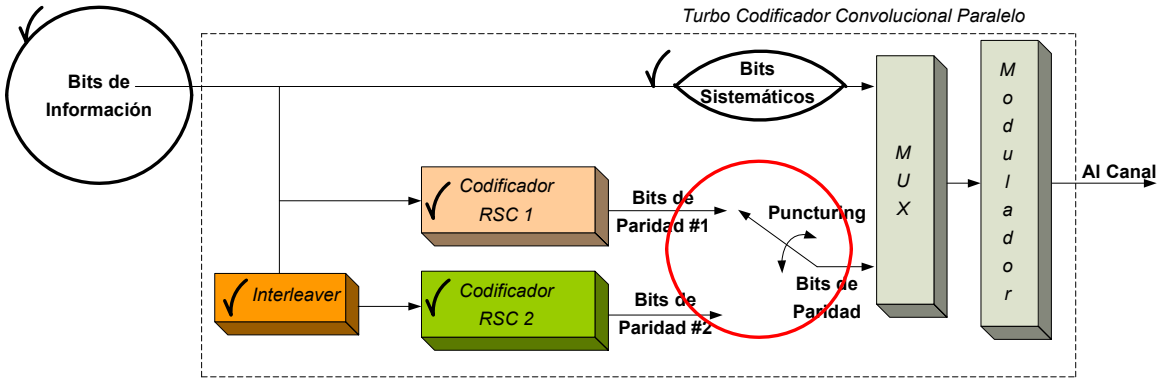


Figura 3.30. Ubicación en el diagrama de bloques del Puncturing.

3.3.2.6.1 Descripción

[145]El *Puncturer* es un elemento del Turbo Codificador utilizado para lograr tasas de codificación más altas. Generalmente, este elemento consiste en un switch de multiplexado que reduce la cantidad de bits a ser transmitidos descartando algunos de los bits de paridad producidos por los codificadores. En el lado del receptor, el Turbo Decodificador asumirá que los bits descartados eran todos 1´s o todos 0´s. La Figura 3.30 muestra un Turbo Codificador que utiliza dos RSC´s y en el caso de utilizar un *puncturer*, la secuencia de salida estaría dada por: $v = \left(v_1^{(1)}, v_2^{(1)}, v_1^{(2)}, v_3^{(2)}, v_1^{(3)}, v_2^{(3)}, v_1^{(4)}, v_3^{(4)}, ...\right)$. Para obtener tasas de codificación más bajas, lo que se hace es incrementar el número de interleavers y de codificadores en el Turbo Codificador.

[145] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Págs: 68-69.

3.3.2.6.2 Implementación

El usuario debe ingresar la tasa de codificación mediante el Perforado del Código; si se escoge el punctured (se almacena un 0 en la variable ***puncture***) produciendo una tasa de código de 1/2, que corresponde a un bit sistemático y alterna entre un bit de paridad #1 (bits impares) y un bit de paridad #2 (bits pares) mientras que si se usa unpunctured (se almacena un 1 en la variable ***puncture***) produciendo una tasa de código de 1/3, que tiene a un bit sistemático y los dos de paridad. La tasa de código por tanto estará determinada por la Ecuación 3.3:

$$\text{Tasa} = \frac{\text{Número de Bits Ingresados}}{\text{Número de Bits Generados}} \qquad \textbf{(Ecuación 3.3)}$$

```
% Perforado del Código
puncture = input(' Escoja punctured / unpunctured (0/1): Por default: 0     ');
if isempty(puncture)
   puncture = 0;
end
```

Espacio de Código 3.15. Implementación del Ingreso del Perforado del Código.

Para facilitar el proceso de puncturing, se ha concatenado los vectores almacenados en ***v1***, ***v2*** y ***v3*** en una sola matriz ***v***, donde la primera fila corresponden a los bits sistemáticos, la segunda a los bits de paridad # 1 y la tercera fila a los bits de paridad # 2. La visualización de los resultados del puncturing o unpuncturing se ve reflejado en el número de bits que son multiplexados.

```
% Creo una matriz donde: La primera fila corresponde a los bits sistemáticos,
% La segunda fila corresponde a los bits de paridad 1,
% La tercera fila corresponde a los bits de paridad 2.
v = [v1; v2; v3];
```

Espacio de Código 3.16. Implementación de la Matriz para Realizar el Puncturing.

3.3.2.6.3 Interfaz Gráfica de Usuario

El interfaz gráfico de usuario muestra la leyenda *Perforación del Código,* que determina si el código es punctured o unpunctured, como muestra la Figura 3.31.

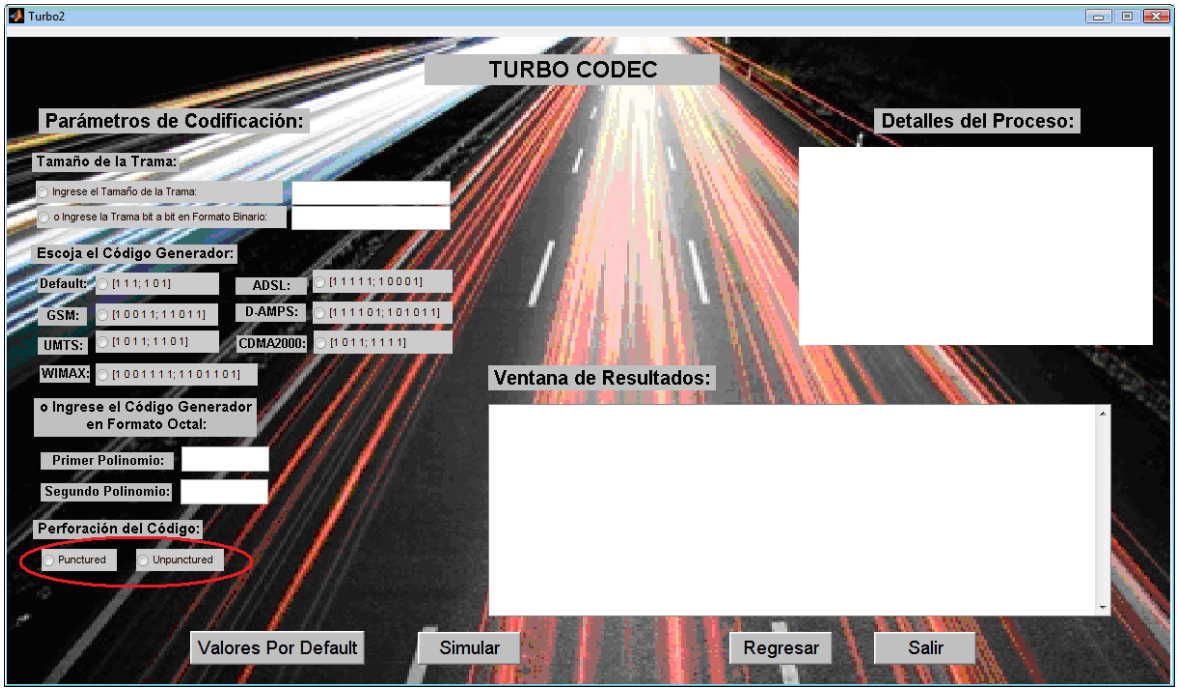


Figura 3.31. Implementación de la Perforación del Código en el Turbo Codec.

El Espacio de Código 3.17 implementa el perforado del código, en la segunda ventana del interfaz gráfico de usuario:

```
% puncture = 0, puncturing con tasa de 1/2;
% puncture = 1, no puncturing
if (get(handles.nopuncture,'Value') == get(handles.nopuncture,'Max'))
   puncture=1;
   punc_tails = ('Unpunctured, Tasa de Código = 1/3 ');
else
   puncture=0;
   punc_tails = ('Punctured, Tasa de Código = 1/2 ');

end
```

Espacio de Código 3.17. Implementación del Puncturing en el Interfaz Gráfico de Usuario.

3.3.2.7 Multiplexor

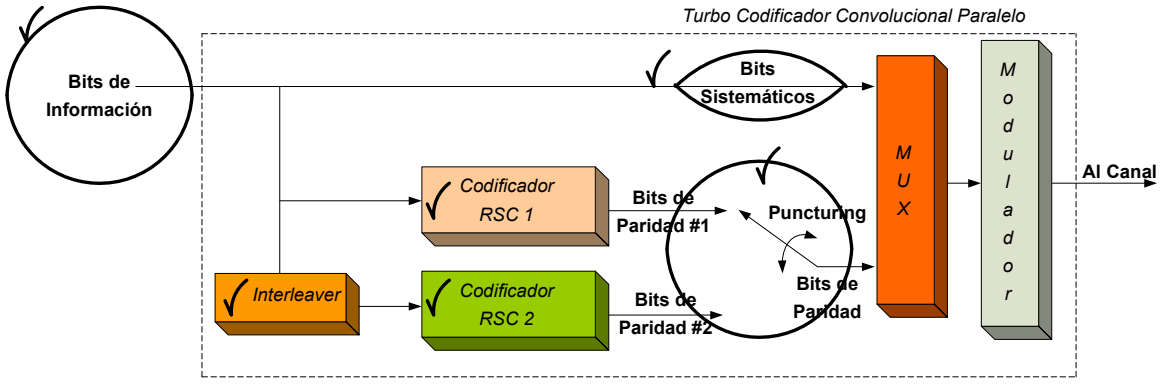


Figura 3.32. Ubicación en el diagrama de bloques del Multiplexor.

3.3.2.7.1 Descripción

Los datos almacenados en las tres filas de la matriz ***v*** correspondientes a los bits sistemáticos, bits de paridad #1 y bits de paridad #2, respectivamente, se ordenan en el vector ***mux*** que corresponde a la salida multiplexada de acuerdo a la tasa de código. Este vector de bits multiplexados se establece tomando en cuenta la disposición de las columnas de la matriz ***v***.

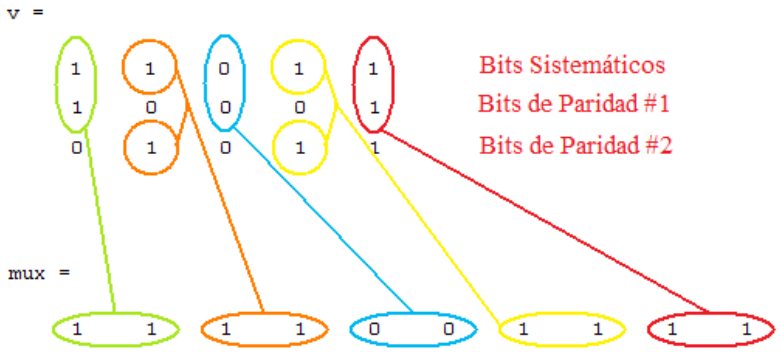


Figura 3.33. Ejecución de Prueba para la Visualización de los Bits Multiplexados con Tasa 1/2.

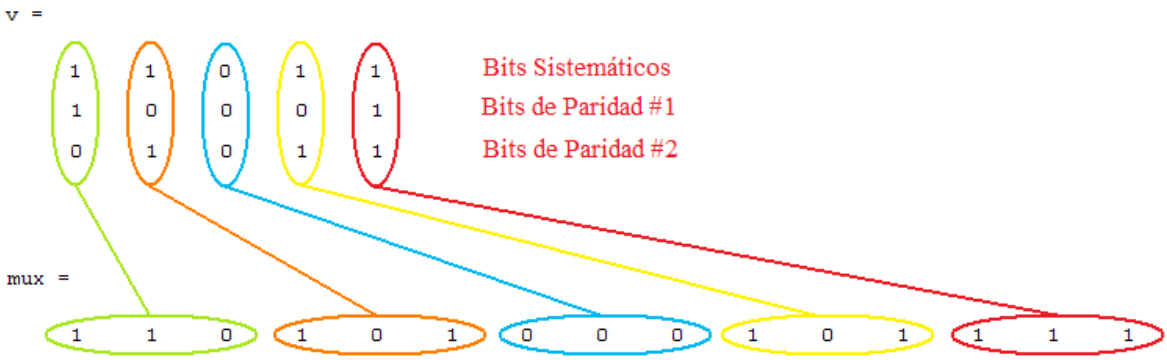


Figura 3.34. Ejecución de Prueba para la Visualización de los Bits Multiplexados con Tasa 1/3.

3.3.2.7.2 Implementación

```
% Multiplexa de paralelo a serie para conseguir un vector de salida
% puncture = 0: punctured, tasa = 1/2;
% puncture = 1; unpunctured, tasa = 1/3;
[n,K] = size(G); % Calculo nuevamente el tamaño del Código Generador
if puncture == 0    % Punctured
   for i = 1:Long_Trama
       mux(1,n*(i-1)+1) = v(1,i);
       if rem(i,2)
          mux(1,n*i) = v(2,i); % Bits impares del RSC1
       else
          mux(1,n*i) = v(3,i); % Bits pares del RSC2
       end
   end
```

```
    Bits_Multiplexados = mux

else                % Unpunctured
    for i = 1:Long_Trama
        for j = 1:3
            mux(1,3*(i-1)+j) = v(j,i);
        end
    end
    Bits_Multiplexados = mux
end
```

Espacio de Código 3.18. Implementación del Multiplexor.

3.3.2.8 Modulador

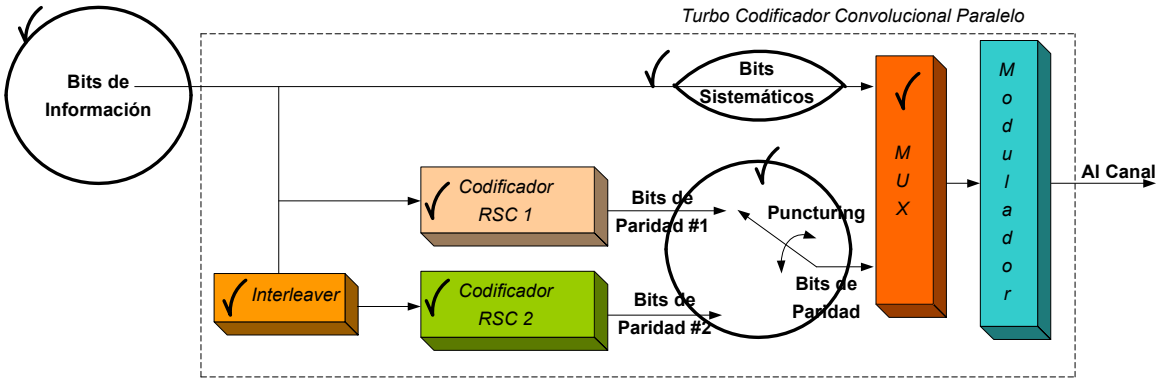


Figura 3.35. Ubicación en el diagrama de bloques del Modulador.

3.3.2.8.1 Descripción

Finalmente se realiza una Modulación BPSK (antípoda[146]) de los bits multiplexados, entre los valores de +1 y -1, asignándose un +1 para el 1 lógico y un -1 para el 0 lógico. Estos valores modulados implementados en Matlab son almacenados en el vector ***modula***, y posteriormente serán enviados a través del canal AWGN.

```
mux =
     1     1     1     1     0     0     1     1     1     1

modula =
     1     1     1     1    -1    -1     1     1     1     1
```

Figura 3.36. Ejecución de Prueba para la Visualización de los Bits Modulados con Tasa 1/2.

[146] ANTÍPODA: La relación de fase entre los elementos para BPSK (180° fuera de fase), es el formato al que se le menciona como *modulación antípoda* y ocurre solo cuando se permite dos niveles de señales binarias y cuando una señal es la negativa exacta de la otra. Debido a que ningún otro esquema de señalización de bit por bit es mejor, el rendimiento de la modulación antípoda se utiliza como referencia de comparación.

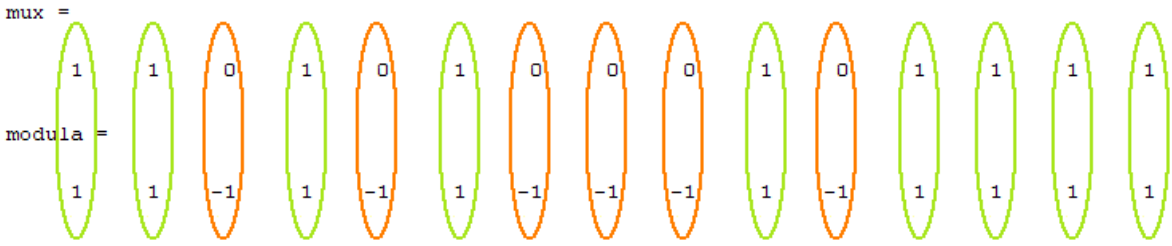


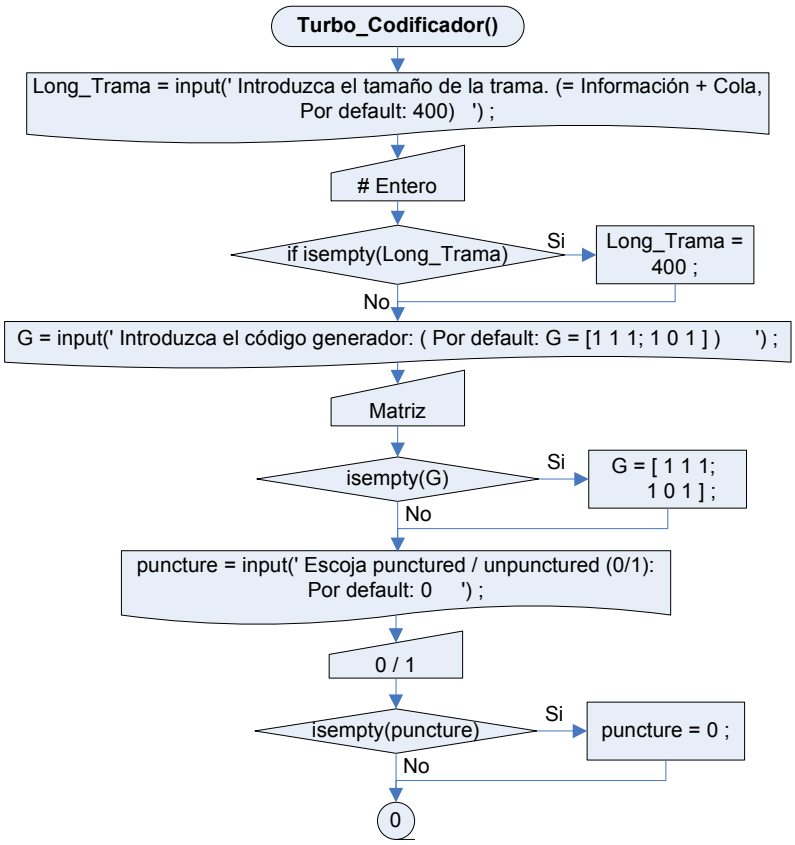
Figura 3.37. Ejecución de Prueba para la Visualización de los Bits Modulados con Tasa 1/3.

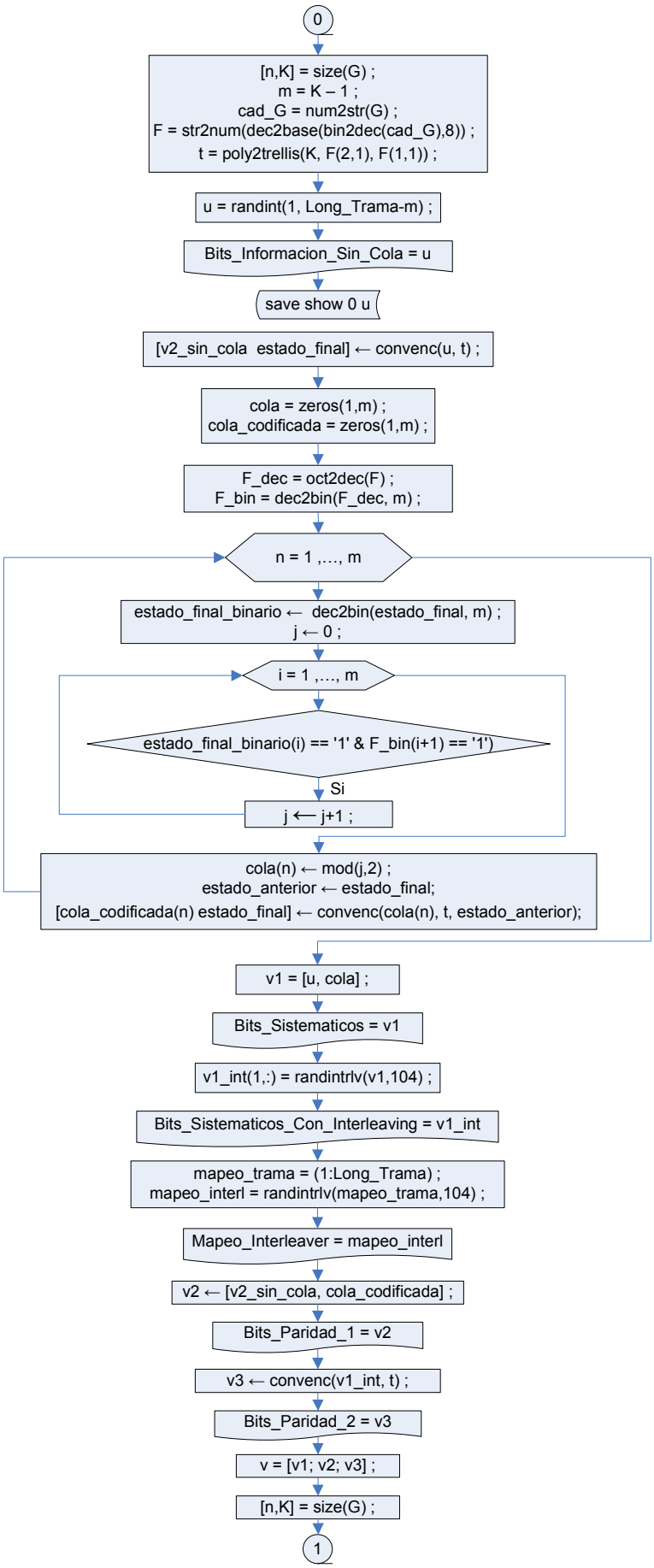
3.3.2.8.2 Implementación

```
% Modulación BPSK: +1/-1
M = 2; % Número de niveles o estados de la señal modulada
modula = real(modulate(modem.pskmod(M,pi),mux)); % Bits Modulados (BPSK)
Bits_Modulados = modula
```

Espacio de Código 3.19. Implementación del Modulador.

3.3.2.8.3 Diagrama de Flujo de la Función Turbo Codificador




0
[n,K] = size(G) ;
m = K – 1 ;
cad_G = num2str(G) ;
F = str2num(dec2base(bin2dec(cad_G),8)) ;
t = poly2trellis(K, F(2,1), F(1,1)) ;
u = randint(1, Long_Trama-m) ;
Bits_Informacion_Sin_Cola = u
save show 0 u
[v2_sin_cola estado_final] ← convenc(u, t) ;
cola = zeros(1,m) ;
cola_codificada = zeros(1,m) ;
F_dec = oct2dec(F) ;
F_bin = dec2bin(F_dec, m) ;
n = 1 ,..., m
estado_final_binario ← dec2bin(estado_final, m) ;
j ← 0 ;
i = 1 ,..., m
estado_final_binario(i) == '1' & F_bin(i+1) == '1'
Si
j ⟵ j+1 ;
cola(n) ← mod(j,2) ;
estado_anterior ← estado_final;
[cola_codificada(n) estado_final] ← convenc(cola(n), t, estado_anterior);
v1 = [u, cola] ;
Bits_Sistematicos = v1
v1_int(1,:) = randintrlv(v1,104) ;
Bits_Sistematicos_Con_Interleaving = v1_int
mapeo_trama = (1:Long_Trama) ;
mapeo_interl = randintrlv(mapeo_trama,104) ;
Mapeo_Interleaver = mapeo_interl
v2 ← [v2_sin_cola, cola_codificada] ;
Bits_Paridad_1 = v2
v3 ← convenc(v1_int, t) ;
Bits_Paridad_2 = v3
v = [v1; v2; v3] ;
[n,K] = size(G) ;
1

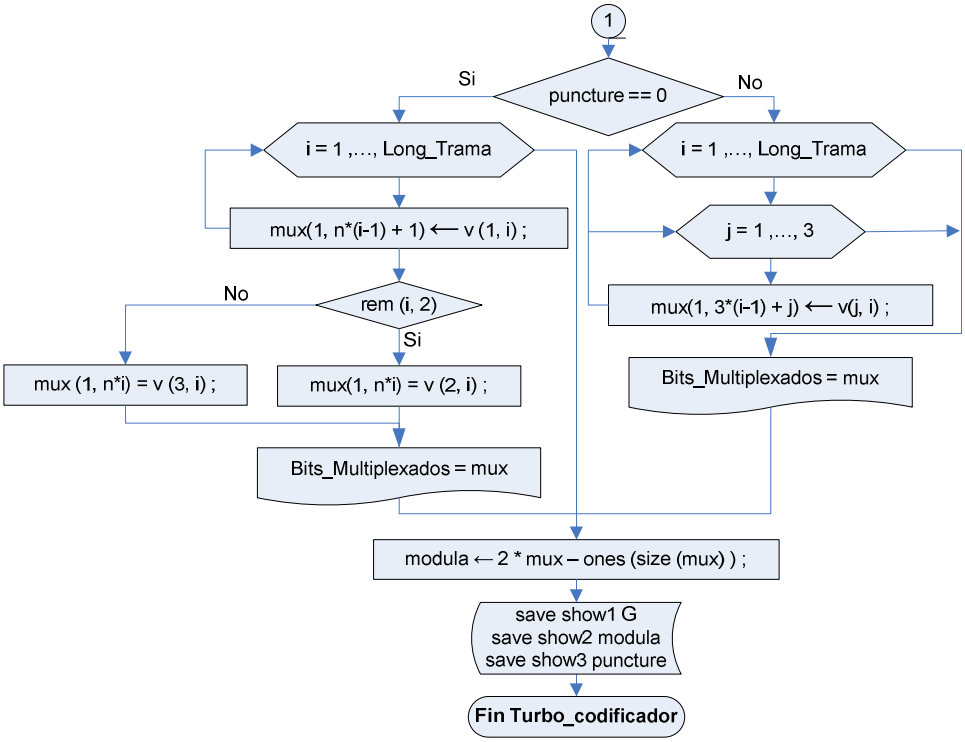


Figura 3.38. Diagrama de Flujo de la Función Turbo_Codificador.

3.3.2 DISEÑO DEL CANAL AWGN

La Figura 3.39 muestra la ubicación en el diagrama de bloques del Turbo Codec, del Canal AWGN.

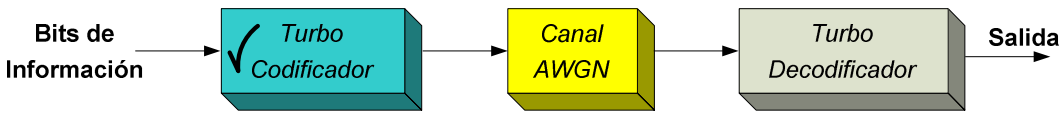


Figura 3.39. Ubicación en el diagrama de bloques del Canal AWGN.

3.3.2.1 Canal AWGN

3.3.2.1.1 Descripción

Los bits que fueron modulados, para motivo de análisis pasan por un canal AWGN, que simula las posibles alteraciones de los valores al pasar a través de un medio, y en este caso se almacenan en el vector ***r***.

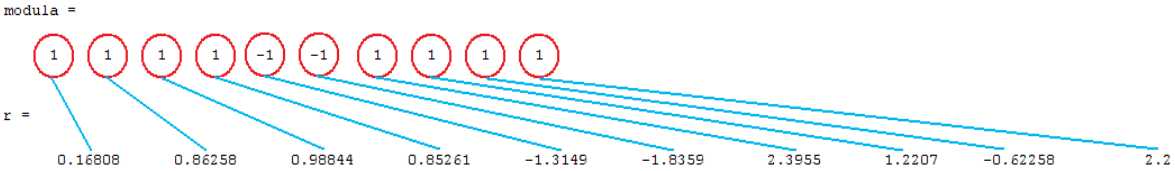


Figura 3.40. Ejecución de Prueba para la Visualización del Canal AWGN con Tasa 1/2.

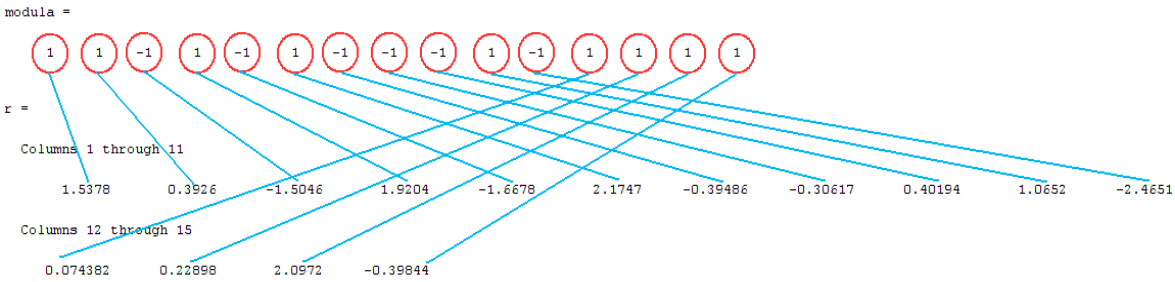


Figura 3.41. Ejecución de Prueba para la Visualización del Canal AWGN con Tasa 1/3.

Uno de lo datos que debe ser ingresado por el usuario es el valor de la relación de E_b/N_o en dB, y que es uno de los parámetros que se utiliza para introducir ruido AWGN en el canal. Este valor debe ser un entero positivo que generalmente está fluctuando entre 1 y 4.

```
% Valor de la relación Eb/No en dB
d = input(' Introduzca el valor de la relación Eb/No en dB : Por default: [2.0]    ');
if isempty(d)
   d = [2.0];
end
```

Espacio de Código 3.20. Implementación del Ingreso de la Eb/No.

3.3.2.1.2 Implementación

El Canal AWGN al igual que todos los bloques que corresponden al Turbo Decodificador son implementados en la sub-función *Turbo_Decodificador* a excepción de los algoritmos de decodificación para los cuales se ha creado una sub-función propia para cada caso. En particular el Canal AWGN implementado ya fue descrito en el Espacio de Código 1.6.

3.3.2.1.3 Interfaz Gráfica de Usuario

La Figura 3.42 implementa en la interfaz gráfica de usuario el ingreso del parámetro E_b/N_o en dB.

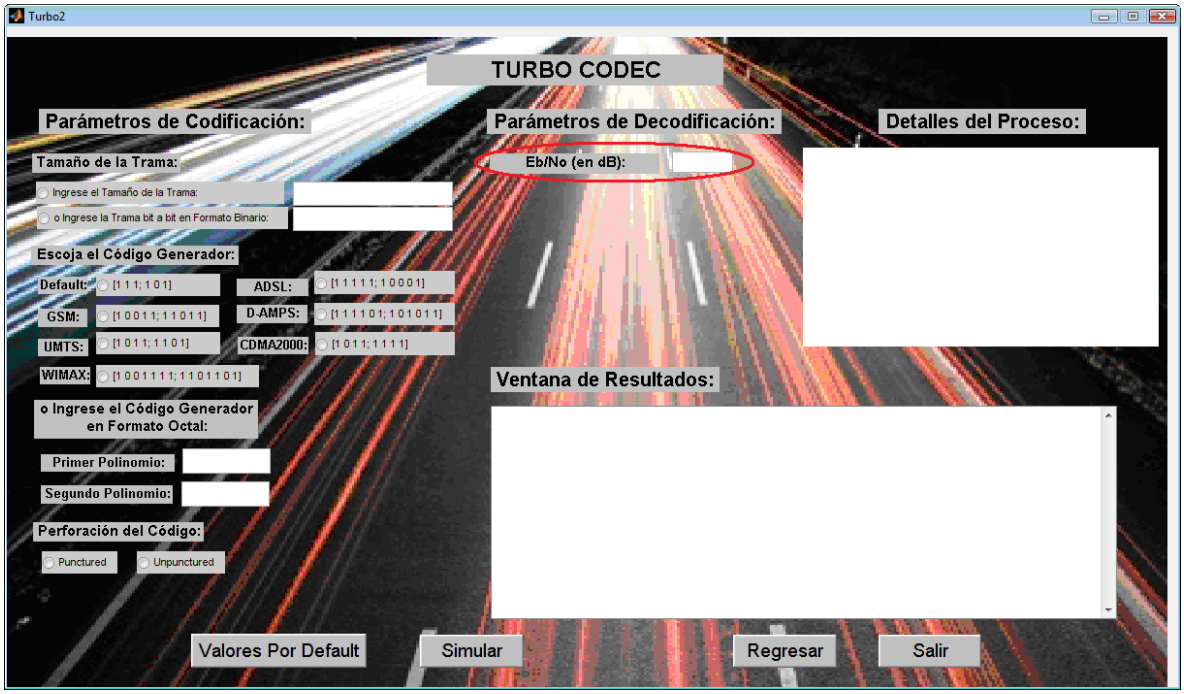


Figura 3.42. Implementación del Ingreso de Eb/No en dB en el Turbo Codec.

3.3.3 DISEÑO DEL TURBO DECODIFICADOR

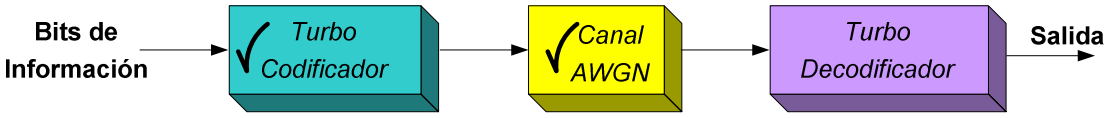


Figura 3.43. Ubicación en el diagrama de bloques del Turbo Decodificador.

La Figura 3.43 muestra la ubicación en el diagrama de bloques del Turbo Codec, del Turbo Decodificador. La Turbo Decodificación se realiza de forma iterativa, es decir, la salida suave del proceso es realimentada y utilizada como entrada en la siguiente iteración de decodificación. Las iteraciones, aunque aumentan la latencia del sistema, reducen considerablemente la potencia requerida para alcanzar un mejor desempeño (muy cercano a la solución óptima con una complejidad en la implementación no prohibitiva) y aumentar la eficiencia de los receptores.

La Figura 3.44 muestra un diagrama de bloques que ejemplifica el proceso de Turbo Decodificación implementado, [147]se hace uso de esta estructura en el

[147] FUENTE: BARBULESCU, Adrian S.; PIETROBON, Steven S.; TURBO CODES: A Tutorial on a New Class of Powerful Error Correcting Coding Schemes, Part II: Decoder Design and Performance; Pág: 4.

decodificador, ya que de esta manera la información extrínseca generada por las salidas suaves del *Decodificador 1*, no depende de las actuales entradas al decodificador, haciéndolo estadísticamente independiente, lo que reduce la probabilidad de que los bits sean corrompidos por el ruido del canal aumentando así el rendimiento del Turbo Codec.

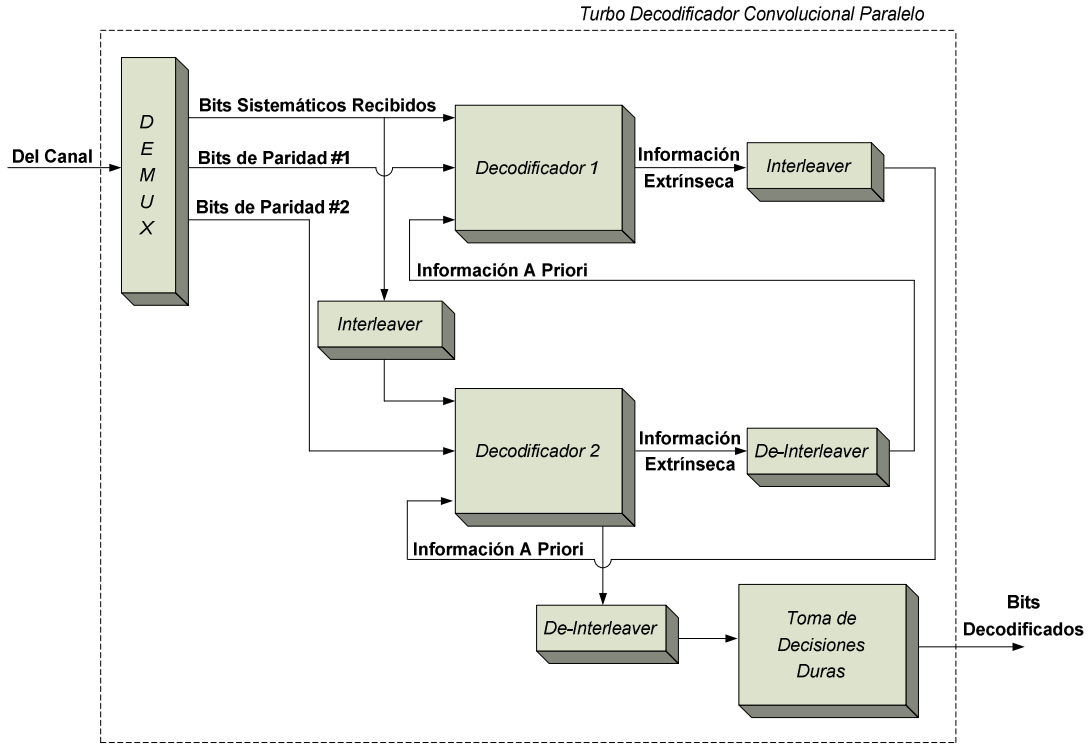


Figura 3.44. Diagrama de Bloques del Turbo Decodificador. [148]

Para el proceso de decodificación son ingresados dos parámetros por parte del usuario, los cuales son:

- *El Algoritmo de Decodificación:* el mismo que permite escoger entre dos tipos de algoritmos: MAP y SOVA.
- *El Número de Iteraciones para cada Trama:* que hace referencia al número de iteraciones que se lleva a cabo por cada trama que se está procesando.

El Espacio de Código 3.21 implementa en Matlab las opciones que permiten al usuario ingresar por teclado estos valores.

[148] FUENTE: MOON, Todd K.; Error Correction Coding: Mathematical Methods and Algorithms; Pág: 604.

```
% Permite escoger el Algoritmo de Decodificación
dec_alg = input(' Introduzca el algoritmo de decodificación. (0 = Log-MAP, 1 = SOVA) Por
default: 0    ');
if isempty(dec_alg)
   dec_alg = 0;
end

% Número de iteraciones
num_iter = input(' Introduzca el número de iteraciones para cada trama: Por default: 2
');
if isempty(num_iter)
   num_iter = 2;
end
```

Espacio de Código 3.21. Implementación del Ingreso del Algoritmo de Decodificación y Número de Iteraciones.

Para el caso del interfaz gráfico de usuario, los valores predefinidos por *default* en el Turbo Decodificador, se pueden observar en la Tabla 3.6:

Parámetro	Valor por Default
Algoritmo de Decodificación	MAP
Número de Iteraciones por Trama	2
E_b/N_o (en dB)	2

Tabla 3.6. Resumen de Parámetros de Simulación con Valores por Default.

3.3.3.1 Demultiplexor

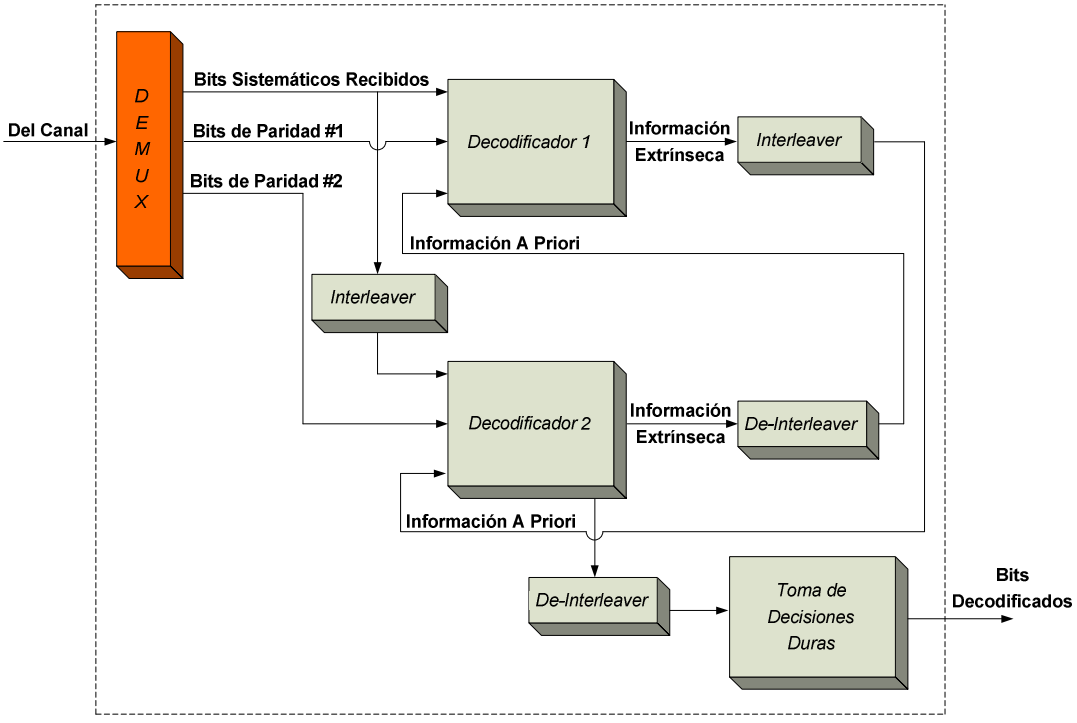


Figura 3.45. Ubicación en el diagrama de bloques del Demultiplexor.

3.3.3.1.1 Descripción

Se realiza la demultiplexación de serie a paralelo en el receptor final, para obtener el valor de cada palabra codificada. Se extraen los bits de paridad y los bits sistemáticos que pasaron a través de un medio (canal AWGN), para los dos decodificadores utilizados y para el segundo decodificador se realiza el interleaving de los bits sistemáticos recibidos. Los bits sistemáticos se guardan en el vector ***bits_sist*** mientras que los bits de paridad #1 en el vector ***bits_pari_1*** y los bits de paridad #2 en el vector ***bits_pari_2***.

El orden en el cual se colocan los bits en el vector ***bits_sist***; considerando que para una tasa de código de 1/2 se tomaba en cuenta un solo bit de paridad de forma alternada, se hallarán saltando un bit en el vector ***r***.

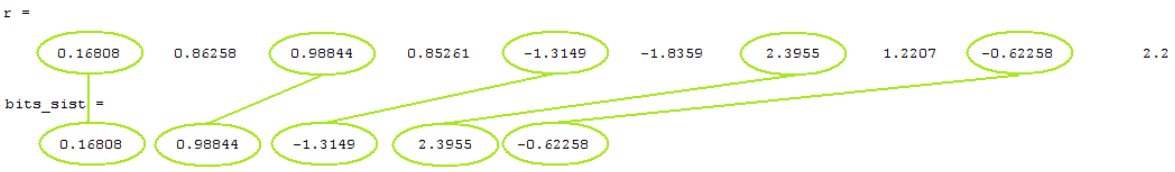


Figura 3.46. Ejecución de Prueba para la Visualización de los Bits Sistemáticos con Tasa 1/2.

Mientras que si se tiene una tasa de código de 1/3 se toma en cuenta los dos bits de paridad, por lo que los bits sistemáticos se hallaran saltando dos bits en el vector ***r*** como se ilustra en la Figura 3.47.

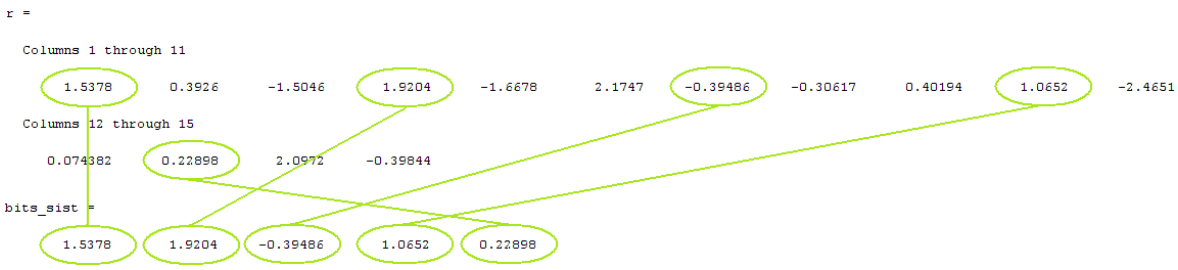


Figura 3.47. Ejecución de Prueba para la Visualización de los Bits Sistemáticos con Tasa 1/3.

Se debe tomar en cuenta que si se ha producido un proceso de puncturing en el Turbo Codificador, para lograr una tasa de codificación más alta (tasa de código 1/2), el Turbo Decodificador deberá insertar ceros en las salidas suaves del canal

que correspondan a los bits descartados por el puncturer; esto sucede tanto con el vector ***bits_pari_1*** como con el vector ***bits_pari_2***, tal como se muestra en las Figuras 3.48 y 3.49:

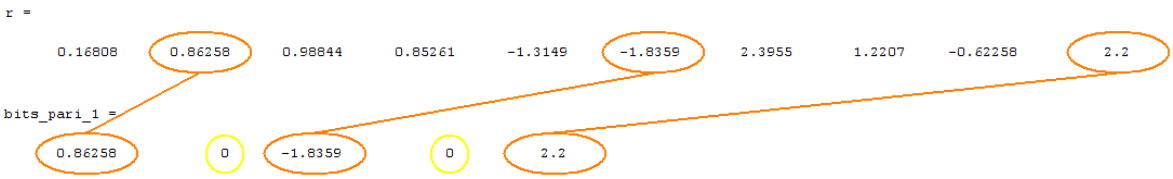


Figura 3.48. Ejecución de Prueba para la Visualización de los Bits de Paridad #1 con Tasa 1/2.

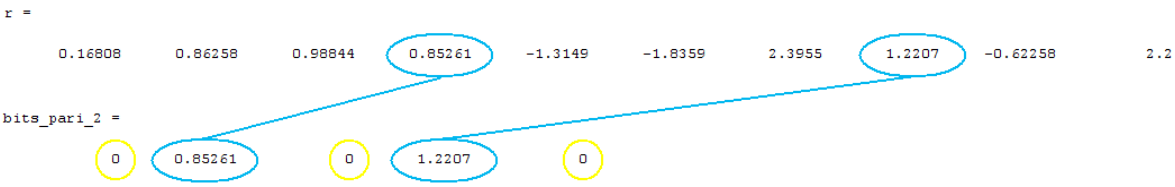


Figura 3.49. Ejecución de Prueba para la Visualización de los Bits de Paridad #2 con Tasa 1/2.

Para una tasa de código de 1/3 se toma en cuenta los dos bits de paridad, sin la necesidad de agregar ceros de relleno; dicho efecto se observa en las Figuras 3.50 y 3.51:

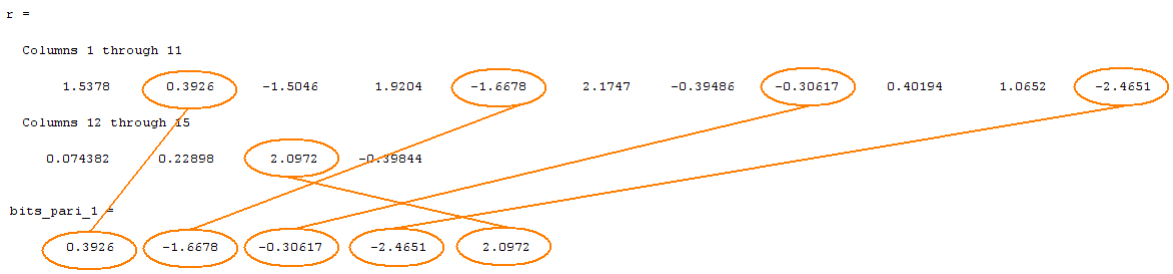


Figura 3.50. Ejecución de Prueba para la Visualización de los Bits de Paridad #1 con Tasa 1/3.

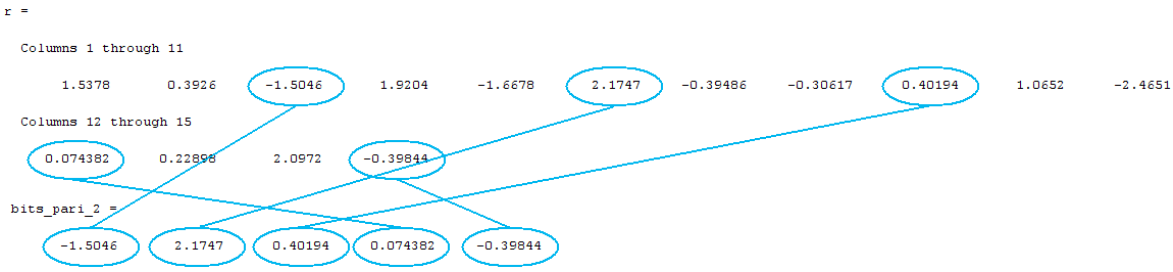


Figura 3.51. Ejecución de Prueba para la Visualización de los Bits de Paridad #2 con Tasa 1/3.

3.3.3.1.2 Implementación

El Demultiplexor de Bits Sistemáticos es implementado en el Espacio de Código 3.22:

```
% Demultiplexo los bits sistematicos y de paridad que han pasado a través del canal
if puncture == 1        % Unpunctured
   for i = 1:Long_Trama
       bits_sist(i) = r(3*(i-1)+1);
       for j = 1:2
           demux_total(j,2*i) = r(3*(i-1)+1+j);
       end
   end

else                     % Punctured
   for i = 1:Long_Trama
       bits_sist(i) = r(2*(i-1)+1);
       for j = 1:2
           demux_total(j,2*i) = 0;
       end
       if rem(i,2)>0
          demux_total(1,2*i) = r(2*i);
       else
          demux_total(2,2*i) = r(2*i);
       end
   end
end

% Para el decodificador uno: (extraigo bits sistemáticos)
Bits_sistematicos_demultiplexados = bits_sist
```

Espacio de Código 3.22. Implementación del Demultiplexor de Bits Sistemáticos.

Mientras que el demultiplexor de Bits de Paridad, es implementado en el Espacio de Código 3.23:

```
% Para el decodificador uno: (extraigo bits de paridad 1) y
% Para el decodificador dos: (extraigo bits de paridad 2)
for jj = 1:Long_Trama
    bits_pari_1 (1,jj) = demux_total(1,2*(jj-1)+2);
    bits_pari_2 (1,jj) = demux_total(2,2*(jj-1)+2);
end
Bits_paridad_1_demultiplexados = bits_pari_1
Bits_paridad_2_demultiplexados = bits_pari_2
```

Espacio de Código 3.23. Implementación del Demultiplexor de Bits de Paridad.

3.3.3.2 Interleaving Bits Sistemáticos

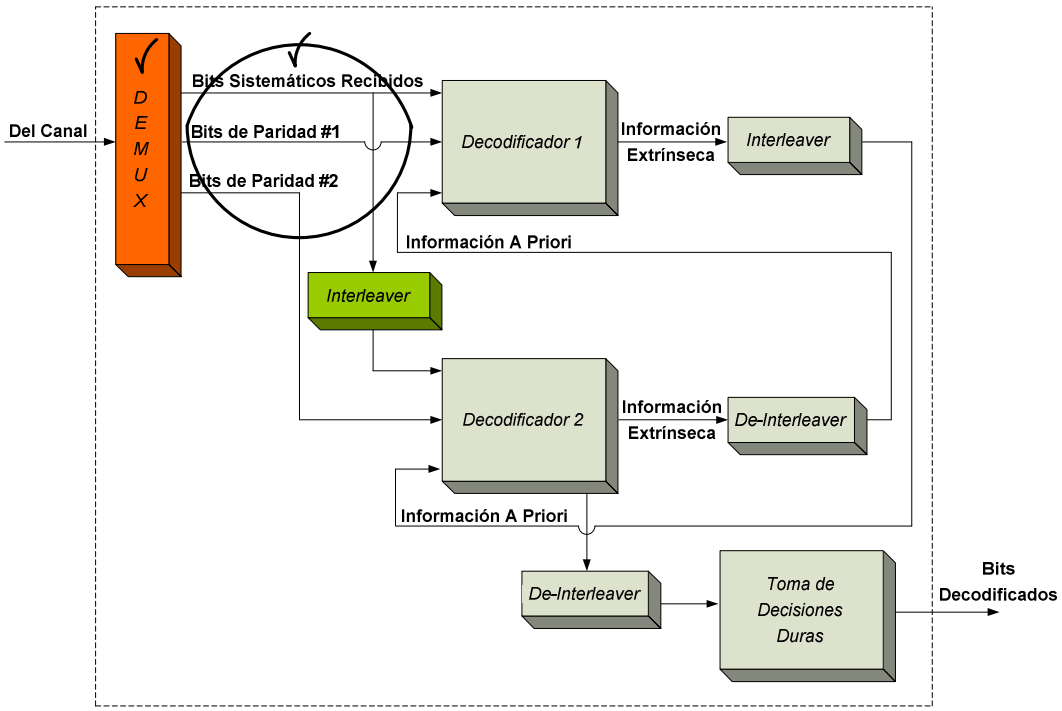


Figura 3.52. Ubicación en el diagrama de bloques del Interleaver de Bits Sistemáticos.

3.3.3.2.1 Descripción

[149]El principal papel del interleaver en un proceso de turbo decodificación consiste en tratar de que los decodificadores hagan estimados no correlacionados de los valores suaves del mismo bit de información, ya que entre menos correlacionados estén ambos estimados, mejor será la convergencia del algoritmo de decodificación iterativa.

De la misma manera que en el turbo codificador, se implementa un interleaver pseudo-aleatorio en el turbo decodificador, y a través de la función *randintrlv* se desordenará las posiciones en la que se encontraban inicialmente los bits sistemáticos demultiplexados almacenados en el vector ***bits_sist*,** siendo el estado inicial ***104*** y la nueva secuencia con interleaving son los bits almacenados en el vector ***bits_sist_int***. Se debe tomar en cuenta que el proceso de interleaving, tanto para la tasa de 1/2 como para la tasa de 1/3, es idéntico puesto que se usa el mismo estado inicial.

[149] FUENTE: BARBULESCU, Sorín A.; What a Wonderful Turbo World; Pág: 52.

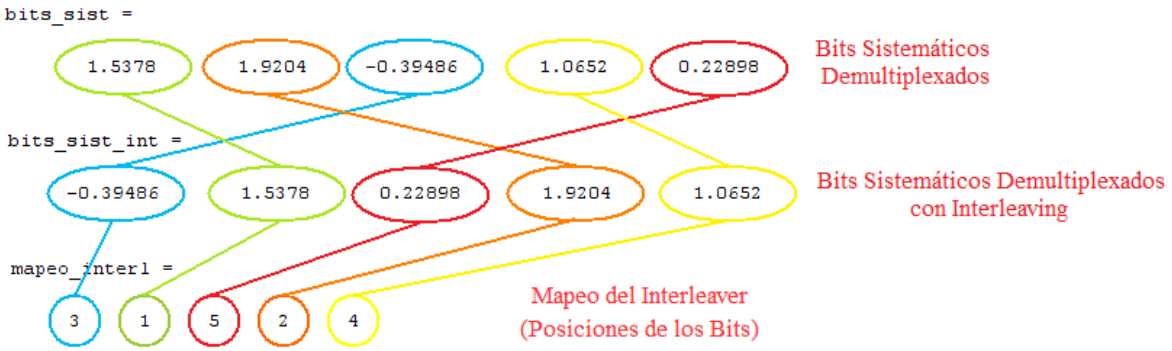


Figura 3.53. Ejecución de Prueba para la Visualización del Interleaving de los Bits Sistemáticos Demultiplexados con Tasa 1/3.

3.3.3.2.2 Implementación

El interleaving de los Bits Sistemáticos Demultiplexados, es implementado en el Espacio de Código 3.24.

```
% Para el decodificador dos: (realiza interleaving de bits sistemáticos)
bits_sist_int = randintrlv(bits_sist,104);
Bits_sistematicos_demultiplexados_con_interleaving = bits_sist_int
```

Espacio de Código 3.24. Implementación del Interleaver de Bits Sistemáticos.

Una vez mostrados los valores intermedios de la demultiplexación se vuelve a armar la matriz que contenga a los bits sistemáticos y bits de paridad, pero en este caso se agrega también a los bits sistemáticos con interleaving, estos datos se almacenan en ***demux_total***. El Espacio de Código 3.25 muestra dicha implementación.

```
% Creo una matriz cuya primera fila contenga los bits sistemáticos y bits de paridad
% 1 y la segunda fila los bits sistemáticos con interleaving y bits de paridad 2.
for j = 1:Long_Trama
    demux_total(1,2*(j-1)+1) = bits_sist(1,j);
    demux_total(2,2*(j-1)+1) = bits_sist_int(1,j);
end
```

Espacio de Código 3.25. Implementación de Matriz de Bits Demultiplexados.

3.3.3.2.3 Diagrama de Flujo del Demultiplexor

A continuación, la Figura 3.54 muestra la parte del diagrama de flujo de la función *Turbo_Decodificador* que implementa el demultiplexor:

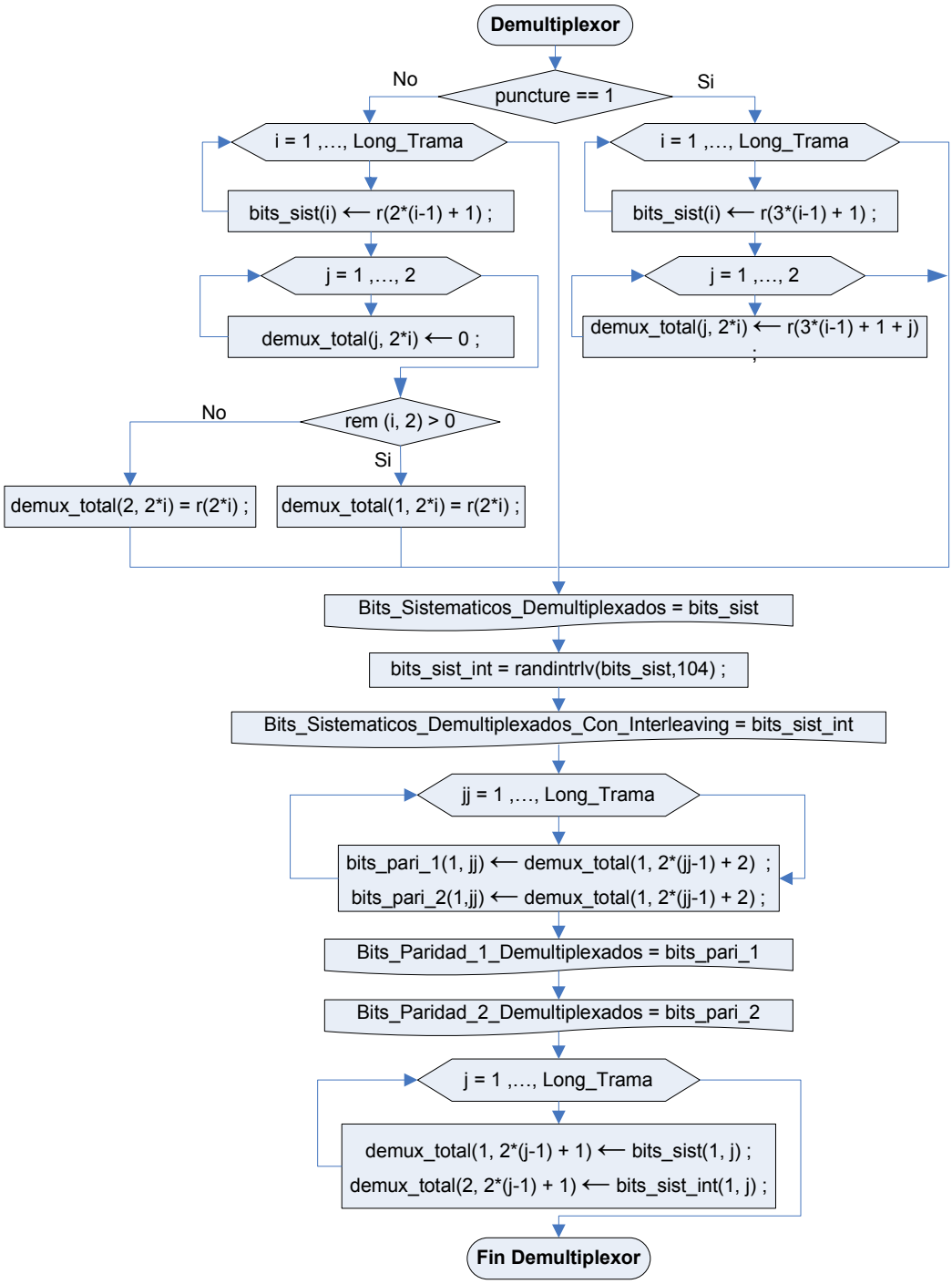


Figura 3.54. Diagrama de Flujo de la función Turbo_Decodificador que Implementa el Demultiplexor.

3.3.3.3 Información A-Priori Decodificador 1

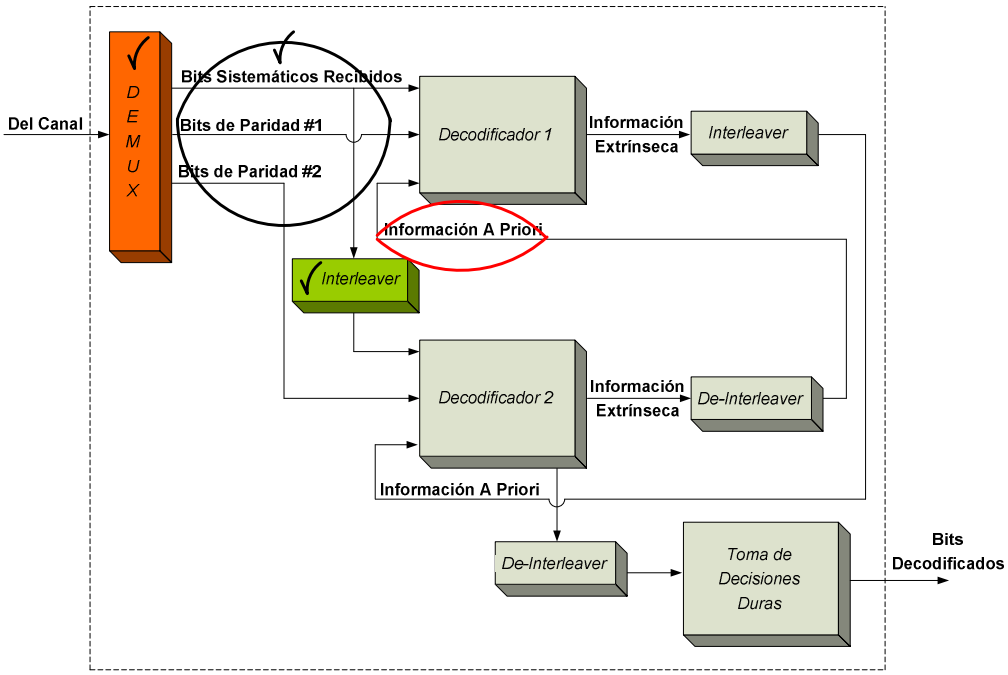


Figura 3.55. Ubicación en el diagrama de bloques de la Información A Priori del Decodificador 1.

3.3.3.3.1 Descripción

El número de iteraciones para cada trama, es otro de lo datos que debe ser ingresado por el usuario, siendo este uno de los parámetros que influye en la información a priori del decodificador 1. Como se muestra en la Figura 3.55, la información a priori del decodificador 1, es la misma que resulta del proceso de de-interleaving de la información extrínseca del decodificador 2. Para iniciar el proceso de turbo decodificación, este vector es inicializado a cero, ya que esta información, para la primera iteración, todavía no ha pasado por el decodificador 2. En el caso de tener dos o más iteraciones, el valor del vector ***Información_A_Priori_Dec_1***, va a ser diferente de cero.

```
Informacion_A_Priori_Dec_1 =

     0     0     0     0     0

Informacion_A_Priori_Dec_1 =

    4.308    -0.4933    -6.4746    -0.76175    -1.5934
```

Primera Iteración

Segunda Iteración

Figura 3.56. Ejecución de prueba para la visualización de la Información A-Priori del Decodificador 1, con dos iteraciones.

3.3.3.3.2 Implementación

```
% Inicializo la información extrínseca
    L_e(1:Long_Trama) = zeros(1,Long_Trama);
    for iter = 1:num_iter

% Decodificador N°1
        L_a = randdeintrlv(L_e,104);
        Informacion_A_Priori_Dec_1 = L_a
```

Espacio de Código 3.26. Implementación de la Información A Priori del Decodificador 1.

3.3.3.3.3 Interfaz Gráfica de Usuario

El número de iteraciones es implementado al igual que los otros parámetros para la decodificación en la segunda ventana del interfaz gráfico; se debe considerar que mientras el proceso de Turbo Decodificación está en ejecución, se decrementa con cada nueva iteración la BER de los bits decodificados. Sin embargo, la ganancia de codificación de una iteración a otra se decrementa con el número de iteraciones. Por esta razón, y por motivos de complejidad en la implementación del Turbo Decodificador, generalmente solo se efectúan de 6 a 8 iteraciones en un proceso de Turbo Decodificación.

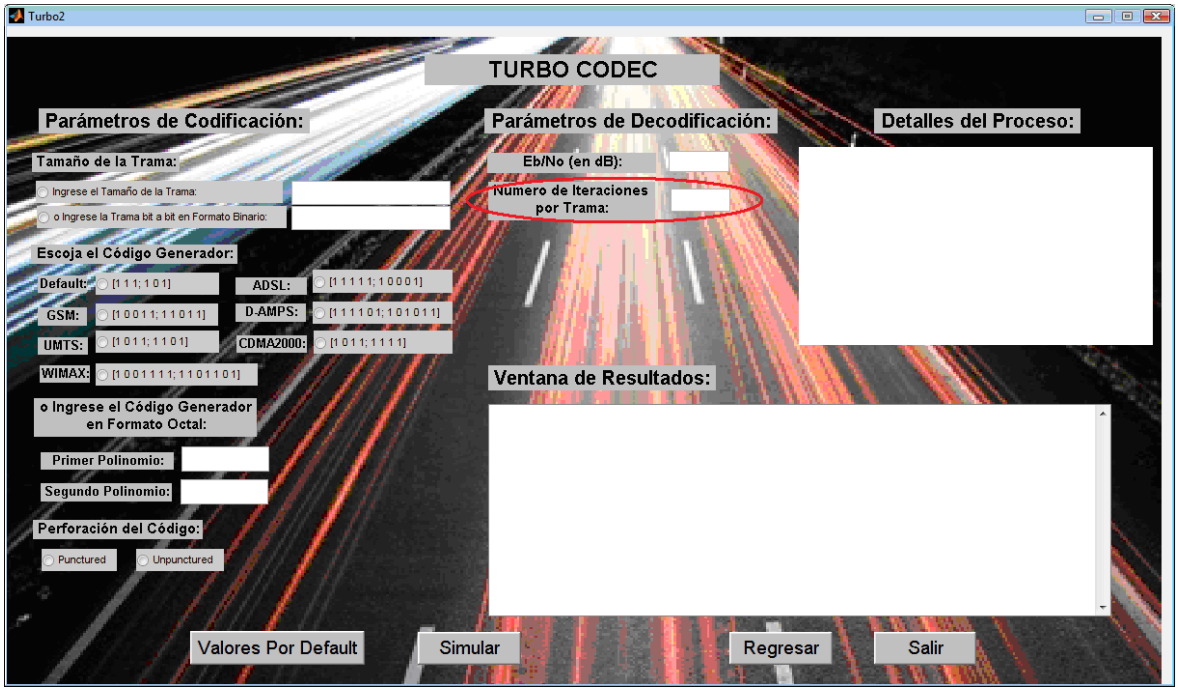


Figura 3.57. Implementación del Ingreso del Número de Iteraciones por Trama en el Turbo Codec.

3.3.3.4 Decodificador 1

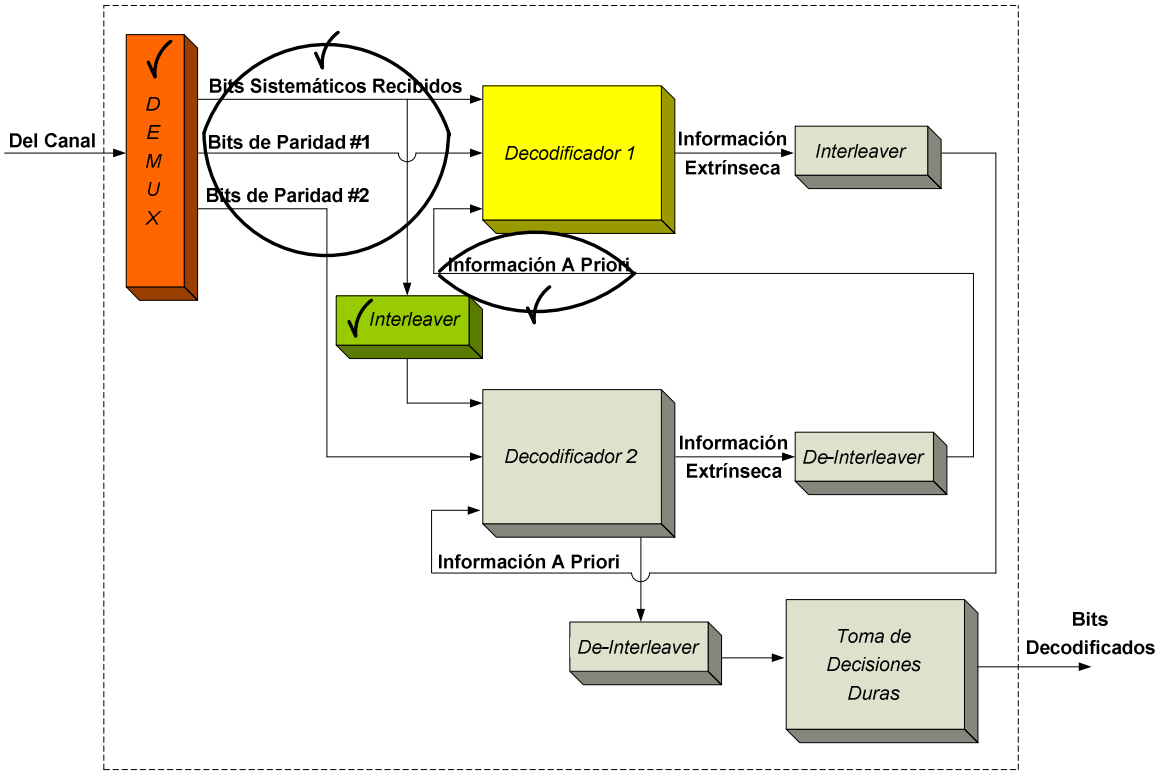


Figura 3.58. Ubicación en el diagrama de bloques del Decodificador 1.

Las entradas al Turbo Decodificador provenientes del canal deben ser suaves, es decir que si se utiliza un demodulador, este debe cuantizar la señal a más de dos estados, o en caso contrario no usar un demodulador, para la implementación del Turbo Decodificador del presente proyecto de titulación, se ha optado por la segunda opción. De este modo, ambos decodificadores del Turbo Decodificador deberán ser capaces tanto de generar como de recibir salidas y entradas suaves, respectivamente. Para que esto se cumpla, es necesaria la utilización de algoritmos de decodificación *SISO*[150].

Se debe tener en cuenta que estas entradas y salidas suaves se determinan en términos de las llamadas *log-likelihoods ratios*, que para el caso específico de los turbo decodificadores SISO, se organizan tal como se muestra en la Figura 3.59.

[150] SISO: (Soft-In-Soft-Out) el "Soft-In" se refiere al hecho de que los datos de entrada pueden tomar valores distintos de 0 o 1, a fin de indicar la confiabilidad. El "Soft-out" se refiere al hecho de que cada bit en la salida decodificada también adquiere un valor que indica la confiabilidad.

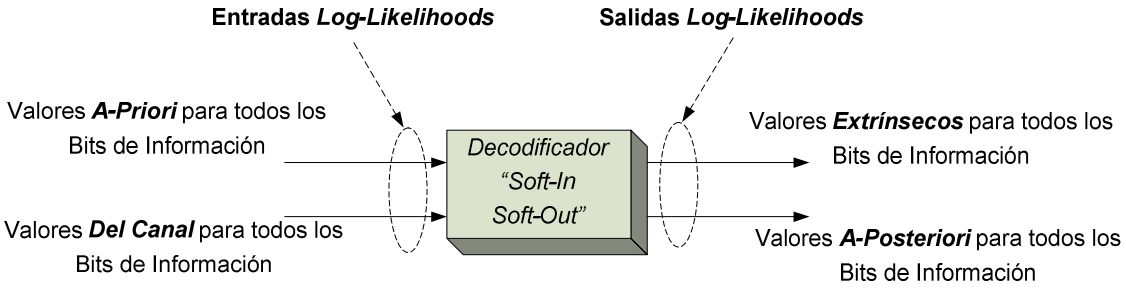


Figura 3.59. Decodificador SISO. [151]

Previo, a la descripción formal de los algoritmos SISO se dará una breve reseña del significado de las LLR, para comprender su significado y como se representan.

3.3.3.4.1 Relaciones Log-Likelihood (LLR´s, Log-Likelihood Ratios)

[152]Las entradas y salidas suaves de los decodificadores son representadas por las *relaciones Log-likelihood* donde las magnitudes y amplitudes dan información sobre el signo de cada bit y la probabilidad de una decisión correcta. La LLR para un bit decodificado u_{deco} está dada por la Ecuación 3.4:

$$L(u_{deco}) = ln\left(\frac{P(u_{deco}=+1)}{P(u_{deco}=-1)}\right) \longrightarrow L(u_{deco}) = ln\left(\frac{P(u_{deco}=+1)}{1-P(u_{deco}=+1)}\right) \qquad \textbf{(Ecuación 3.4)}$$

La ecuación anterior es ploteada por medio del Espacio de Código 3.27, como función de la probabilidad de uno de los eventos. En la Figura 3.60 se observa que cuando $L(u_{deco})$=1, entonces la probabilidad de que $u_{deco} = +1$ es igual a 0.73.

```
clear all
x = 0:0.01:1; % Generación de un vector entre 0 y 1
y = log(x./(1-x)); % Cálculo de las LLR
plot(x,y), grid; % Grafica LLR Vs Probabilidad
xlabel('Probabilidad'); % Título del Eje de las X
ylabel('LLR'); % Título del Eje de las Y
title('LLR Vs Probabilidad'); % Título del Gráfico
```

Espacio de Código 3.27. LLR vs Probabilidad.

[151] FUENTE: HAGENAUER, Joachim; OFFER, Elke; PAPKE, Lutz; Iterative Decoding of Binary Block and Convolutional Codes; Págs: 432.
[152] FUENTE: LANGTON, Charan; Intuitive Guide to Principles of Communications: Turbo Coding and MAP Decoding, Parte 1; Págs: 9-10.

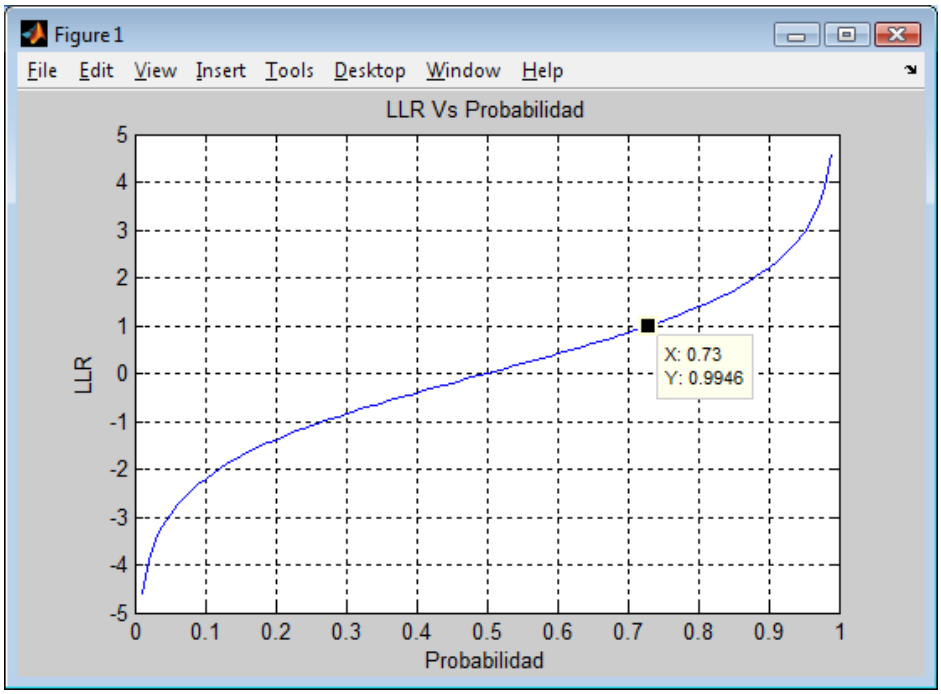


Figura 3.60. LLR vs Probabilidad.

De esta manera se puede decir que, $P(u_{deco} = +1)$ es la probabilidad de que el bit $u_{deco} = +1$ (1 binario), y $P(u_{deco} = -1)$ es la probabilidad de que el bit $u_{deco} = -1$ (0 binario). Una LLR positiva y alta indica alta probabilidad de que el bit decodificado sea un 1, contrariamente una LLR negativa y alta indica una gran probabilidad de que el bit decodificado sea un 0, es decir que la LLR es un buen indicador del signo del bit, mucho mejor que la distancia Euclideana.

3.3.3.4.2 Algoritmos de Decodificación SISO

[153]Los Turbo Códigos son capaces de lograr ganancias de codificación muy altas a tasas de error muy bajas, por el hecho de combinar códigos simples, de forma tal que cada uno de estos códigos puede decodificarse de manera independiente por medio de decodificadores SISO con algoritmos de decodificación de máxima probabilidad (*MLDA - Maximum Likelihood Decoding Algorithms*). Mediante los decodificadores SISO, la información puede ser pasada de un decodificador al

[153] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Págs: 78-82.

siguiente de modo iterativo. Existen dos categorías de algoritmos de decodificación SISO:

- Los algoritmos que minimizan la probabilidad de error de símbolo y;
- Los algoritmos de decodificación de máxima probabilidad, que minimizan la probabilidad de error de palabra o secuencia.

A la primera categoría pertenece el algoritmo *MAP* (*Maximum A Posteriori*) y a la segunda el algoritmo *SOVA* (*Soft Output Viterbi Algorithm*).

3.3.3.4.3 Algoritmo MAP (Maximum A Posteriori)

[154]Este algoritmo de decodificación minimiza el BER y fue propuesto en 1974 por Bahl, Cocke, Jelinek y Raviv. Aunque el algoritmo MAP supera al algoritmo de Viterbi en términos de BER, debido a su complejidad significativamente superior solo se lo utilizó en raras ocasiones, hasta que los turbo códigos fueron descubiertos. Se lo conoce también como *Algoritmo BCJR* debido a las iniciales de los apellidos de sus descubridores.

[155]El algoritmo MAP provee la secuencia de bits estimada y las probabilidades para cada bit de que haya sido decodificado correctamente. Siendo esto fundamental para el proceso de decodificación iterativa para Turbo Códigos. La gran complejidad del algoritmo MAP se debe a la enorme cantidad de multiplicaciones y adiciones que debe efectuar por cada estimado que produce la APP (probabilidad a-posteriori[156]) de cada bit de información decodificado. Por lo tanto, se han realizado varias investigaciones para reducir la complejidad de este algoritmo, y se ha logrado desarrollar dos algoritmos que ofrecen un desempeño casi idéntico al ofrecido por MAP con una complejidad menor: el algoritmo *Max-Log-MAP* y el algoritmo *Log-MAP*.

---

[154] FUENTE: HANZO, L.; LIEW, T.; YEAP, B.; Turbo Coding, Turbo Equalization and Space-Time Coding for Transmission over Wireless Channels; Pág: 13.
[155] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Págs: 88-92.
[156] PROBABILIDAD A-POSTERIORI: Probabilidad que resulta de revisar una probabilidad a priori, inicial o de partida, en función de la información deducida de las nuevas pruebas practicadas. La distinción entre probabilidad a priori y a posteriori es relativa ya que una probabilidad a posteriori vuelve a ser a priori en relación a un nuevo experimento.

[157]Para encontrar la secuencia de bits estimados, maximizando la probabilidad de que este estimado sea correcto, el algoritmo MAP necesita calcular:

- Las métricas de rama (también llamadas métricas de trayectoria) y;
- Las métricas de estado hacia atrás y hacia adelante (por esta razón, el algoritmo MAP también se lo conoce como *Forward-Backward Algorithm*) con la finalidad de estimar una salida suave en forma de un LLR.

A estos parámetros generalmente se los denomina funciones del Algoritmo MAP. [158]El decodificador MAP está constituido por cuatro funciones principales: *alfa, beta, gamma, lambda*. La organización de estas funciones y la forma en la que se ejecutan tomando en cuenta los parámetros que han sido recibidos desde el canal, se muestran en el diagrama de bloques de la Figura 3.61.

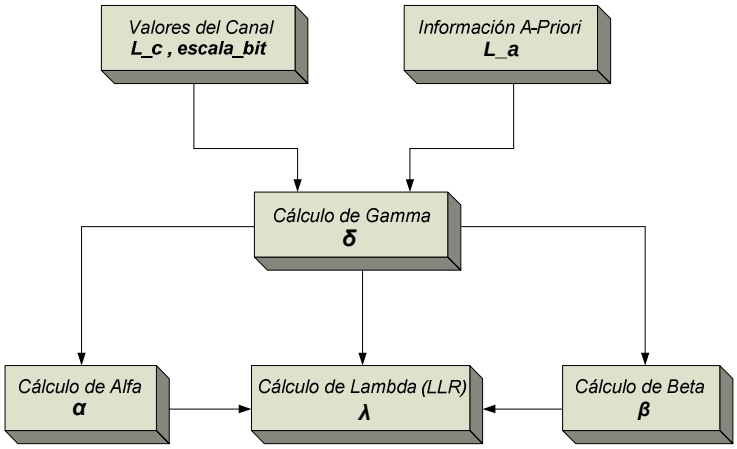


Figura 3.61. Diagrama de Bloques de las Operaciones en el Algoritmo MAP. [159]

La función *Gamma* realiza el cálculo de la métrica de rama, que consiste en estimar la probabilidad de que el dato recibido pertenezca a una de las transiciones del *Trellis*. [160]El cálculo de las métricas de rama se computa por medio de la Ecuación 3.5:

[157] FUENTE: LANGTON, Charan; Intuitive Guide to Principles of Communications: Turbo Coding and MAP Decoding, Parte 1; Pág: 13.
[158] FUENTE: WOODWARD, Jason P.; HANZO, Lajos; Comparative Study of Turbo Decoding Techniques: An Overview; Pág: 2211.
[159] FUENTE: WOODWARD, Jason P.; HANZO, Lajos; Comparative Study of Turbo Decoding Techniques: An Overview; Pág: 2212.
[160] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Págs: 94-95.

$$\delta_t^{k,s} = P(u_t = k)\exp(L_C(x_t k + y_t v_t^{k,s})) \qquad \textbf{(Ecuación 3.5)}$$

Donde u_t es el bit de información, x_t es el símbolo de información recibido, c_t es el bit codificado, y_t es el símbolo codificado recibido, k puede ser 0 o 1, para cada estado s y tiempo t y [161]el valor de confiabilidad L_c del canal está dado por la Ecuación 3.6:

$$L_c = 4(E_b/N_o)(R) \qquad \textbf{(Ecuación 3.6)}$$

Para cada x_t e y_t recibidas existen solamente cuatro posibles combinaciones de u_t y v_t, por lo que se requiere calcular únicamente cuatro métricas de rama diferentes. La ecuación de las métricas de estado hacia adelante necesita dos de las cuatro métricas de rama para cada cálculo nuevo de métricas de estado. [162]Considerando a la información a-priori (L_a), la ecuación de las métricas de rama se puede reescribir como la Ecuación 3.7:

$$\delta_t^{k,s} = \frac{1}{2}u_t L_a + \frac{L_c}{2}\sum_{t=1}^{n} x_t y_t \qquad \textbf{(Ecuación 3.7)}$$

Se debe tener en cuenta que entre las múltiples funciones desarrolladas por el grupo de investigadores del Virginia Tech, de cuyas ideas se ha valido este proyecto de titulación para la implementación del Turbo Codec, se encuentra una que simula al algoritmo Log-MAP. En primera instancia, al ser modificada esta sub-función hacia el algoritmo MAP, se notó que la carga computacional fue bastante grande, lo que se vio reflejado en un gran tiempo de espera para la decodificación de las tramas de entrada; esto principalmente se debe a la forma en que se hace el cálculo del Trellis. Por esta razón se descartó esta primera opción, desarrollándose así una nueva función que simule al algoritmo MAP, y que calcule el Trellis por medio de la función que proporciona Matlab *poly2trellis.* De esta manera se optimizan recursos y se reduce en forma drástica los tiempos de procesamiento haciendo viable la simulación de este algoritmo.

[161] FUENTE: LANGTON, Charan; Intuitive Guide to Principles of Communications: Turbo Coding and MAP Decoding, Parte 1; Pág: 20.
[162] FUENTE: WOODWARD, Jason P.; HANZO, Lajos; Comparative Study of Turbo Decoding Techniques: An Overview; Pág: 2211.

Para la implementación de este algoritmo en Matlab, se debe considerar que los valores contenidos en la matriz ***demux_total*** creada para efecto de la demultiplexación son escalados, antes de ingresar al decodificador 1; es decir que dicha matriz es multiplicada por un [163]factor de confiabilidad que se desprende del canal (calculado bajo el conocimiento del valor de la SNR y los valores recibidos), permitiendo que los valores que van a ser procesados por los algoritmos de decodificación (tanto MAP como SOVA), tengan un valor de guarda, el mismo que es extraído una vez que se termina el proceso, cuando se hace el cálculo de la información extrínseca. Estos valores escalados se almacenan en la matriz ***escala_bit***. Su implementación se ubica en el Espacio de Código 3.28.

```
% Escalo los bits demultiplexados recibidos
    L_c = 4*en*tasa;  % Valor de confiablidad del canal
    escala_bit = 0.5*L_c*demux_total;
```

Espacio de Código 3.28. Escala de los Bits Demultiplexados.

Previo al cálculo de las métricas de rama (Gamma), en la sub-función *map*, se ha configurado nuevamente el Trellis en modo *feedback* con el objeto de que los resultados de éste, sean usados en posteriores cálculos del algoritmo. Dichos resultados son procesados en una matriz ***matriz,*** antes de realizar cualquier cálculo, esto se lo hace para poder manipular de manera más sencilla los datos que se han obtenido hasta el momento en el proceso de turbo decodificación a través de la función *Turbo_Decodificador*. El Espacio de Código 3.29, muestra como la función *map* realiza este proceso.

```
[n,K] = size(G); % Cálculo tamaño Código Generador para obtener constraint length
cad_G = num2str(G); % Conversión código generador a cadena de caracteres
F = str2num(dec2base(bin2dec(cad_G),8)); % Conversión cadena de caracteres binaria a
octal
t = poly2Trellis(K, F(2,1), F(1,1)); % Definición del Trellis

% Matriz donde: 1era matriz es el estado anterior, 2da matriz es el estado actual
matriz = zeros(t.numStates, t.numStates, 2);
for columna = 1:2
```

[163] FUENTE: LANGTON, Charan; Intuitive Guide to Principles of Communications: Turbo Coding and MAP Decoding, Parte 1; Pág: 22.

```
    for i = 1:t.numStates
        if columna == 1
        matriz(i,t.nextStates(i,columna)+1,1) = -1; % La 1er columna son las ramas del 0
        else
        matriz(i,t.nextStates(i,columna)+1,1) = +1; % La 2da columna son las ramas del 1
        end
    end
end

for k = 1:t.numStates
    for j = 1:t.numStates % Ramas de j a k
        if matriz(j,k,1) ~= 0
           % Almacena en matriz la salida codificada a partir del estado actual y el
           % bit entrante al codificador, se hace la conversión a formato polar
           matriz(j,k,2) = 2*(t.outputs(j, ((matriz(j,k,1) + 1)/2) + 1)) - 1;
        end
    end
end

% Algoritmo de Decodificacion MAP
Long_Trama = length(L_a);
escala_bit = escala_bit';

% Demultiplexo bit sistematico escalado (o bit sistematico con interleaving) y bit
% de paridad (1 ó 2) escalado
for i = 1:Long_Trama
    % Bit sistematico recibido
    sist(i,1) = escala_bit(2*(i-1)+1,1);
    % Bit de paridad recibido
    pari(i,1) = escala_bit(2*(i-1)+2,1);
end

% Traspongo la informacion a priori para poder procesarla
L_a = L_a';
```

Espacio de Código 3.29. Configuración del Trellis y Procesamiento de los Datos Recibidos.

Mientras que el Espacio de Código 3.30, muestra como la función *map* calcula los valores correspondientes a las métricas de rama (gammas).

```
% Cálculo de los GAMMAS
Gamma = zeros(Long_Trama,t.numStates,t.numStates);
Gamma(:,:,:) = - Inf;
for i = 1:Long_Trama
        for k = 1:t.numStates
            for j = 1:t.numStates
                if matriz(j,k,1) ~= 0;
                   Gamma(i,j,k) = matriz(j,k,1).*L_a(i)/2 +
L_c/2*(sist(i).*matriz(j,k,1)+pari(i).*matriz(j,k,2));
                end
            end
        end
end
```

Espacio de Código 3.30. Cálculo de las Métricas de Rama (Gammas).

Alfa realiza la estimación hacia adelante de la métrica de estados, es decir, calcula la probabilidad de la relación del estado actual del Trellis con el estado

anterior. [164]Un esquema de las *métricas de estado hacia adelante* α_t^s , se muestra en la Figura 3.62. Estas métricas se basan en métricas de estado previas $\alpha_{t-1}^{s_1}$ y en métricas de rama previas $\delta_{t-1}^{k,s}$.

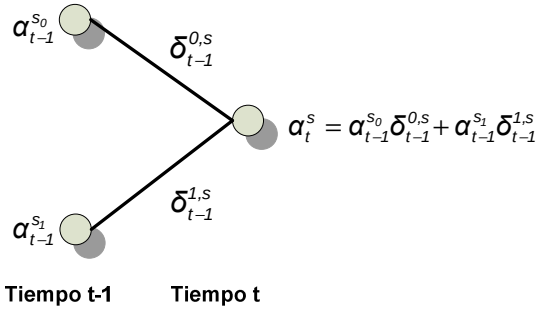


Figura 3.62. Cálculo de las Métricas de Estado hacia Adelante.[165]

El cálculo de un nuevo estado está dado por la siguiente Ecuación 3.8:

$$\alpha_t^s = \alpha_{t-1}^{s_0}\delta_{t-1}^{0,s} + \alpha_{t-1}^{s_1}\delta_{t-1}^{1,s} \qquad \textbf{(Ecuación 3.8)}$$

Donde s_0 (ó s_1) es el estado a partir del cual, dado un bit de entrada cero (ó uno), el estado del codificador cambia al estado *s*.

Se requiere calcular 2^m métricas de estado nuevas; ya que existen 2^m estados, (*m* es la memoria del codificador). El codificador parte del estado cero con cada bloque nuevo, por lo que las métricas de estado hacia adelante son inicializadas a cero, exceptuando a la métrica de estado cero. [166]Se debe tener en cuenta que estos números son mayores que 1, lo que significa, que no son probabilidades, por eso se llaman métricas. Esto también significa que desde una típica trama de Turbo código están miles de secciones de Trellis y que la multiplicación de estos números pueden dar lugar a desbordamiento numérico. Por esta razón, tanto *Alfa* como *Beta* se normalizan. La inicialización y cálculo de las métricas de estado hacia delante se implementan en el Espacio de Código 3.31.

---

[164] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Págs: 93-94.
[165] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Pág: 94.
[166] FUENTE: LANGTON, Charan; Intuitive Guide to Principles of Communications: Coding and Decoding with Convolutional Codes; Pág: 14.

```
% Cálculo de los ALFAS
Alfa = zeros(Long_Trama,t.numStates); %Inicializo Alfa
Alfa(1,:) = -Inf;
Alfa(1,1) = 0;

for i = 2:Long_Trama
    for k = 1:t.numStates
        for j = 1:t.numStates
            A(j) = Alfa(i-1,j) + Gamma(i-1,j,k);
        end
        if A(1) == -Inf & A(2) == -Inf
           prueba = -Inf;
        else
           prueba = max(A(1),A(2)) + log(1+exp(-1*abs(A(1)-A(2))));
        end

        for j = 3:t.numStates
            if A(j) ~= -Inf
               prueba = max(prueba,A(j)) + log(1+exp(-1*abs(prueba-A(j))));
            end
        end
        Alfa(i,k) = prueba;
    end
end
```

Espacio de Código 3.31. Cálculo de las Métricas de Estado hacia Adelante (Alfas).

Beta realiza la estimación hacia atrás de la métrica de estados, calculando las relaciones entre los estados futuros y los actuales del Trellis. [167]Es decir que las métricas de estado nuevo también deben ser calculadas yendo hacia atrás en el tiempo (comenzando al final del bloque), como se muestra en la Figura 3.63. Estas métricas se representan por β_t^s y su cálculo es mediante la Ecuación 3.9:

$$\beta_t^s = \beta_{t+1}^{s_0}\delta_t^{0,s} + \beta_{t+1}^{s_1}\delta_t^{1,s} \qquad \textbf{(Ecuación 3.9)}$$

Donde s_0 (ó s_1) es el siguiente estado del codificador si el bit de entrada es cero (ó uno) y el estado del codificador es s.

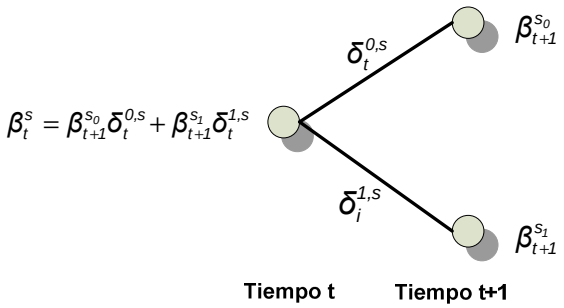


Figura 3.63. Cálculo de las Métricas de Estado hacia Atrás. [168]

[167] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Pág: 95.
[168] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Pág: 95.

Si el Trellis no es terminado en un estado conocido (generalmente el estado cero), las *métricas de estado hacia atrás* β_t^s se inicializan al mismo valor.

Por tanto, la inicialización de las métricas hacia atrás, que están determinadas por la variable *Beta,* se ponen a cero o se inicializan al mismo valor con las instrucciones del Espacio de Código 3.32.

```
% Cálculo de los BETAS
Beta = zeros(Long_Trama+1,t.numStates); %Inicializo Beta
if decodificador == 1 % Para 1er Decodificador, último Beta es 0
   Beta(Long_Trama+1,:) = -Inf;
   Beta(Long_Trama+1,1) = 0;
else  % Para 2do Decodificador, todos los últimos Betas son 0
   Beta(Long_Trama+1,:) = 0;
end

for i = Long_Trama:-1:2
    for j = 1:t.numStates % Estado actual
        for k = 1:t.numStates % Estado próximo
            B(k) = Beta(i+1,k) + Gamma(i,j,k);
        end
        if B(1) == -Inf & B(2) == -Inf;
           prueba = -Inf;
        else
           prueba = max(B(1),B(2)) + log(1+exp(-1*abs(B(1)-B(2))));
        end
        for k = 3:t.numStates
            if B(k) ~= -Inf;
               prueba = max(prueba,B(k)) + log(1+exp(-1*abs(prueba-B(k))));
            end
        end
        Beta(i,j) = prueba;
    end
end
```

Espacio de Código 3.32. Cálculo de las Métricas de Estado hacia Atrás (Betas).

Lambda calcula las *LLR´s*, que consiste en la relación entre la probabilidad de que el dato recibido sea un uno y la probabilidad de que sea un cero, para, con base en ella, calcular la información extrínseca del decodificador. [169]La salida producida por el algoritmo MAP está dada por la siguiente Ecuación 3.10:

$$\lambda_t = \frac{\sum_s \alpha_t^s \delta_t^{1,s} \beta_{t+1}^{s_1}}{\sum_s \alpha_t^s \delta_t^{0,s} \beta_{t+1}^{s_0}} \qquad \textbf{(Ecuación 3.10)}$$

[169] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Pág: 96.

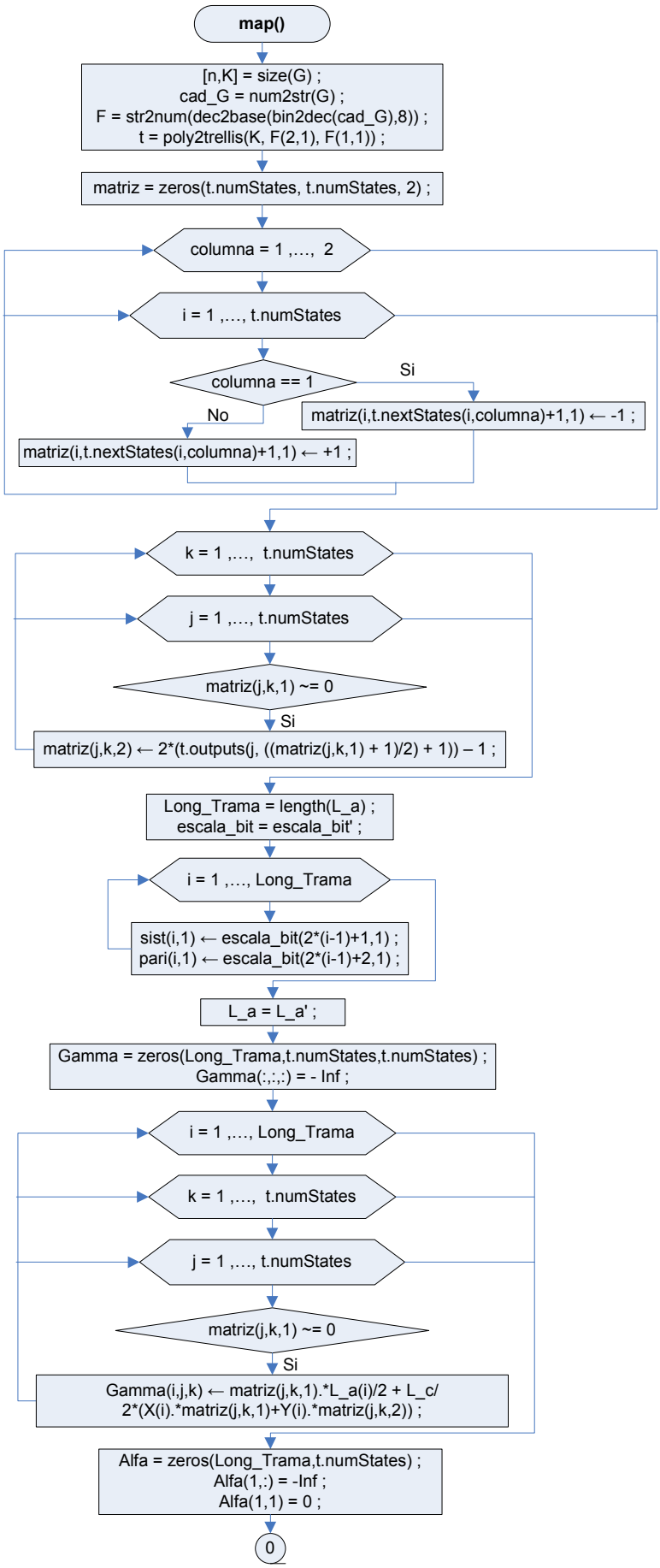
El Espacio de Código 3.33, implementa en Matlab el cálculo de los valores de *Lambda*.

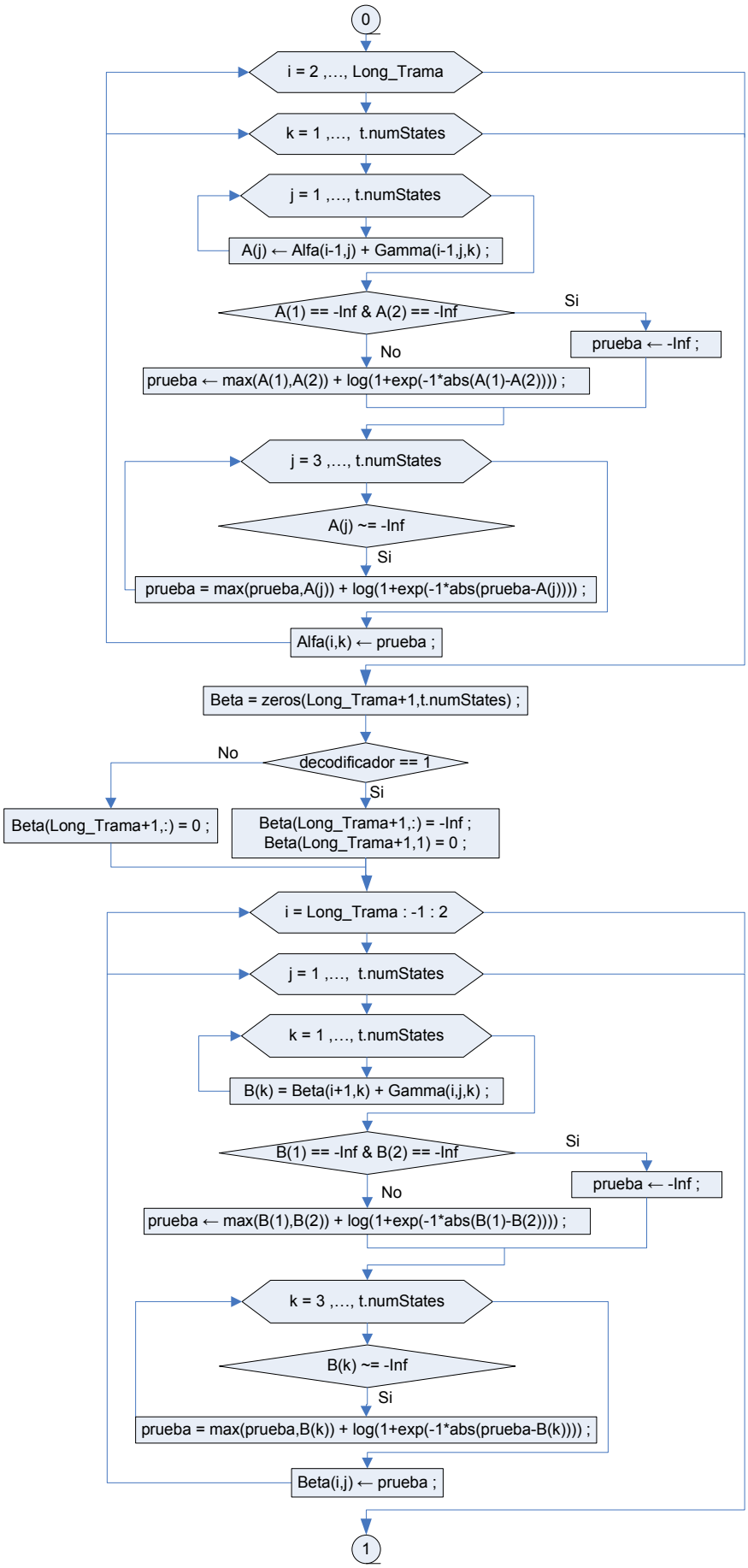
```
% Cálculo de los LAMBDA (LLR)
Lambda = zeros(Long_Trama,1);
for i = 1:Long_Trama
    for k = 1:t.numStates;
        for j = 1:t.numStates;
            if matriz(j,k,1) == 1;
               numerador(k) = Alfa(i,j) + Gamma(i,j,k) + Beta(i+1,k);
            elseif matriz(j,k,1) == -1;
                   denominador(k) = Alfa(i,j) + Gamma(i,j,k) + Beta(i+1,k);
            end
        end
    end

    % LLR en todos los elementos del Trellis del numerador y denominador
    if numerador(1) == -Inf & numerador(2) == -Inf
       numerador_auxiliar = -Inf;
    else
       numerador_auxiliar = max(numerador(1),numerador(2)) + log(1+exp(-
1*abs(numerador(1)-numerador(2))));
    end
    if denominador(1) == -Inf & denominador(2) == -Inf
       denominador_auxiliar = -Inf;
    else
       denominador_auxiliar = max(denominador(1),denominador(2)) + log(1+exp(-
1*abs(denominador(1)-denominador(2))));
    end
    for k = 3:t.numStates
        if numerador(k) ~= -Inf % Para evitar el NaN (Not-a-Number)
           numerador_auxiliar = max(numerador_auxiliar,numerador(k)) + log(1+exp(-
1*abs(numerador_auxiliar-numerador(k))));
        end
        if denominador(k) ~= -Inf % Para evitar el NaN (Not-a-Number)
           denominador_auxiliar = max(denominador_auxiliar,denominador(k)) + log(1+exp(-
1*abs(denominador_auxiliar-denominador(k))));
        end
    end
    Lambda(i)= numerador_auxiliar - denominador_auxiliar;
end
L_total = Lambda';
```

Espacio de Código 3.33. Cálculo de las LLR (Lambda).

3.3.3.4.4 Diagrama de Flujo de la Sub-función MAP

El diagrama de flujo de la sub-función ***map*** se ilustra por medio de la Figura 3.64 que se muestra a continuación:


map()
[n,K] = size(G) ;
cad_G = num2str(G) ;
F = str2num(dec2base(bin2dec(cad_G),8)) ;
t = poly2trellis(K, F(2,1), F(1,1)) ;
matriz = zeros(t.numStates, t.numStates, 2) ;
columna = 1 ,…, 2
i = 1 ,…, t.numStates
columna == 1
Si
No
matriz(i,t.nextStates(i,columna)+1,1) ← -1 ;
matriz(i,t.nextStates(i,columna)+1,1) ← +1 ;
k = 1 ,…, t.numStates
j = 1 ,…, t.numStates
matriz(j,k,1) ~= 0
Si
matriz(j,k,2) ← 2*(t.outputs(j, ((matriz(j,k,1) + 1)/2) + 1)) – 1 ;
Long_Trama = length(L_a) ;
escala_bit = escala_bit' ;
i = 1 ,…, Long_Trama
sist(i,1) ← escala_bit(2*(i-1)+1,1) ;
pari(i,1) ← escala_bit(2*(i-1)+2,1) ;
L_a = L_a' ;
Gamma = zeros(Long_Trama,t.numStates,t.numStates) ;
Gamma(:,:,:) = - Inf ;
i = 1 ,…, Long_Trama
k = 1 ,…, t.numStates
j = 1 ,…, t.numStates
matriz(j,k,1) ~= 0
Si
Gamma(i,j,k) ← matriz(j,k,1).*L_a(i)/2 + L_c/
2*(X(i).*matriz(j,k,1)+Y(i).*matriz(j,k,2)) ;
Alfa = zeros(Long_Trama,t.numStates) ;
Alfa(1,:) = -Inf ;
Alfa(1,1) = 0 ;
0


0
i = 2 ,..., Long_Trama
k = 1 ,..., t.numStates
j = 1 ,..., t.numStates
A(j) ← Alfa(i-1,j) + Gamma(i-1,j,k) ;
A(1) == -Inf & A(2) == -Inf
Si
prueba ← -Inf ;
No
prueba ← max(A(1),A(2)) + log(1+exp(-1*abs(A(1)-A(2)))) ;
j = 3 ,..., t.numStates
A(j) ~= -Inf
Si
prueba = max(prueba,A(j)) + log(1+exp(-1*abs(prueba-A(j)))) ;
Alfa(i,k) ← prueba ;
Beta = zeros(Long_Trama+1,t.numStates) ;
No
decodificador == 1
Si
Beta(Long_Trama+1,:) = 0 ;
Beta(Long_Trama+1,:) = -Inf ;
Beta(Long_Trama+1,1) = 0 ;
i = Long_Trama : -1 : 2
j = 1 ,..., t.numStates
k = 1 ,..., t.numStates
B(k) = Beta(i+1,k) + Gamma(i,j,k) ;
B(1) == -Inf & B(2) == -Inf
Si
prueba ← -Inf ;
No
prueba ← max(B(1),B(2)) + log(1+exp(-1*abs(B(1)-B(2)))) ;
k = 3 ,..., t.numStates
B(k) ~= -Inf
Si
prueba = max(prueba,B(k)) + log(1+exp(-1*abs(prueba-B(k)))) ;
Beta(i,j) ← prueba ;
1

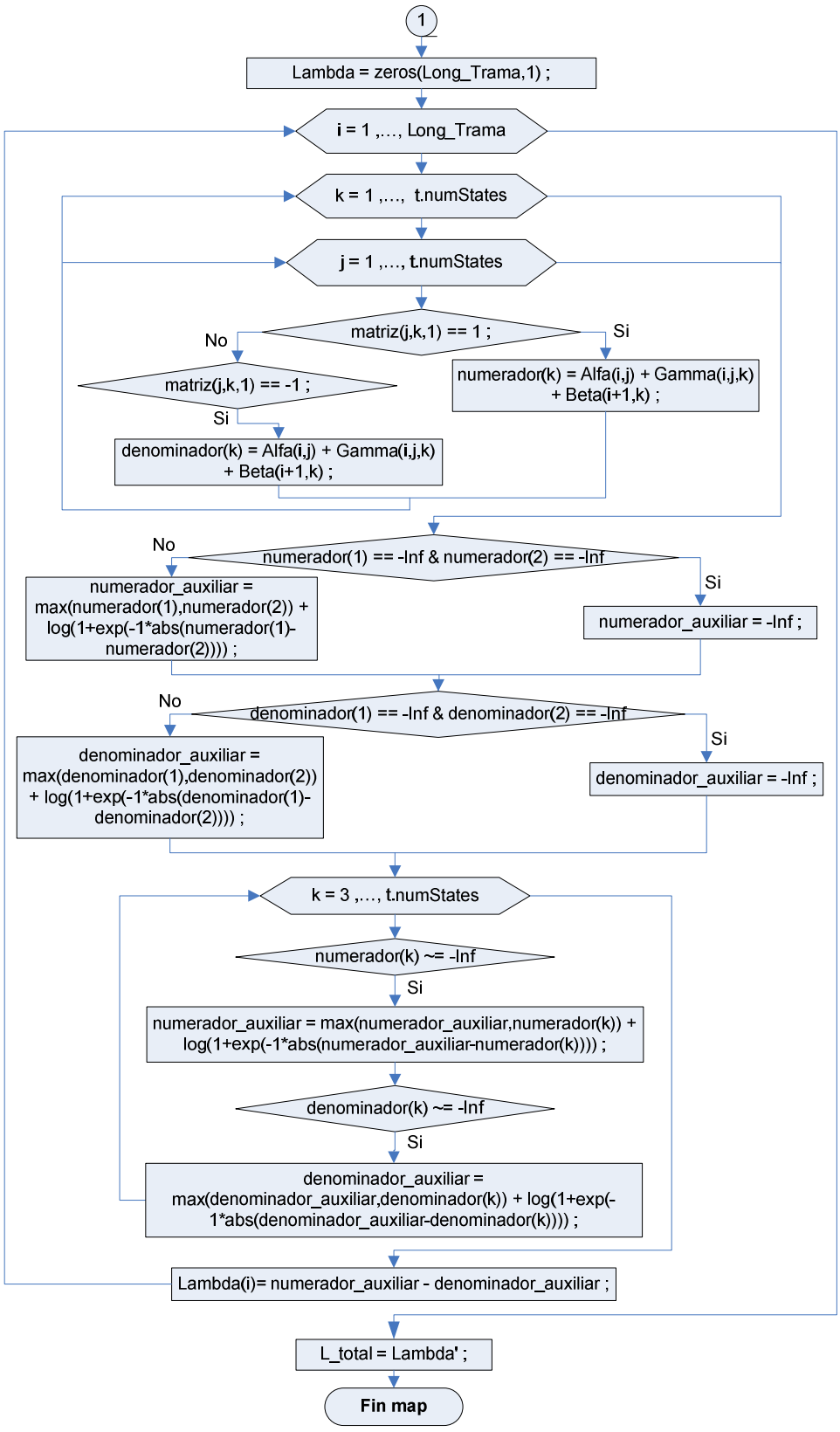


Figura 3.64. Diagrama de Flujo de la Función map.

3.3.3.4.5 Interfaz Gráfica de Usuario

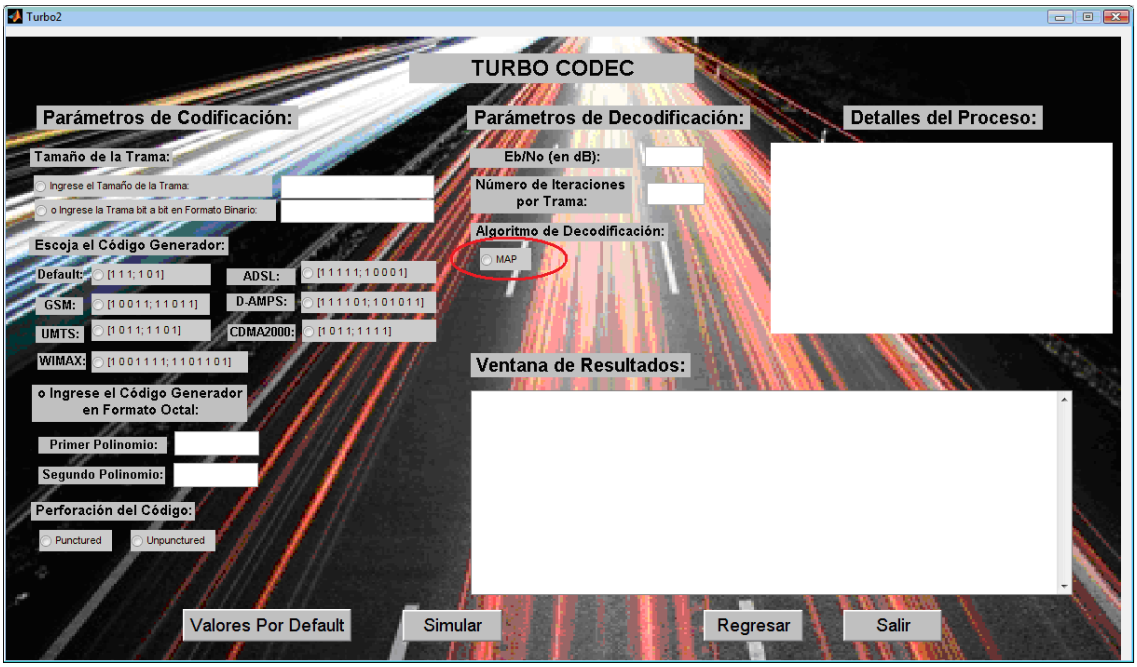


Figura 3.65. Implementación del Algoritmo de Decodificación MAP en el Turbo Codec.

3.3.3.4.6 Algoritmo LOG-MAP

[170]En el año de 1995, fue formulado el algoritmo Log-MAP por los investigadores P. Robertson, E. Villebrun y P. Hoeher. Este algoritmo toma las multiplicaciones comprendidas del algoritmo MAP y las convierte en sumas, reduciéndose de esta manera la complejidad de decodificado, ya que un sumador es más fácil de implementar que un multiplicador y consume menos recursos computacionales.

3.3.3.4.7 Algoritmo MAX-LOG-MAP

[171]El algoritmo Max-Log-MAP fue propuesto por W. Koch y A. Baier en 1990. Es un algoritmo computacionalmente más eficiente, aunque subóptimo derivado del algoritmo MAP. La decodificación utilizando el criterio MAP requiere un gran número de operaciones, entre cálculos exponenciales y multiplicaciones, la reescritura del algoritmo de decodificación en el dominio logarítmico simplifica el procesamiento (logaritmos de los APP, conocidos como *Log-APP).* La aplicación

[170] FUENTE: BARBULESCU, Sorín A.; What a Wonderful Turbo World; Pág: 60.
[171] FUENTE: GLAVIEUX, Alain; Channel Coding in Communication Networks: From Theory to Turbocodes; Págs: 284-286.

del algoritmo Max-Log-MAP realiza una decodificación de Viterbi doble, en la dirección hacia adelante y hacia atrás. Por esta razón, también se llama *Algoritmo Doble de Viterbi*.

[172]La diferencia entre los algoritmos Max-Log-MAP y Log-MAP es la forma en que efectúan la operación suma en el dominio logarítmico, el algoritmo Max-Log-MAP efectúa la operación suma como $\ln\left(e^{x}+e^{y}\right)\approx \max(x,y)$, mientras que Log-MAP efectúa dicha operación como $\ln\left(e^{x}+e^{y}\right)=\max(x,y)+f_{C}\left(\left|y-x\right|\right)$.

3.3.3.4.8 Algoritmo SOVA

[173]Para la turbo decodificación una alternativa a los algoritmos de decodificación tipo MAP (MAP, Log-MAP o Max-Log-MAP) es el algoritmo SOVA descubierto en 1989 por los científicos J. Hagenauer y P. Hoeher que produce probabilidades a-posteriori de las transiciones de estado, o equivalentemente la confiabilidad de los estimados de bit. Este se modifica con respecto al algoritmo convencional de Viterbi en dos aspectos:

- En el algoritmo SOVA las métricas de trayectoria usadas, son modificadas para tomar en cuenta la información a-priori cuando se selecciona la trayectoria (camino) de máxima probabilidad (ML) a través del Trellis.
- El algoritmo SOVA provee una salida suave en forma del LLR a-posteriori para cada bit decodificado, ya que a pesar de que el algoritmo de Viterbi convencional puede aceptar entradas suaves, únicamente es capaz de producir salidas duras y por tanto es inadecuado en Turbo Decodificación.

El investigador Wu Yufei, como parte del MPRG de la Universidad de Virginia Tech, ha creado una sub-función en Matlab *sova0.m* para la simulación del algoritmo SOVA**,** la cual se ha modificado en este proyecto de titulación para poder representar la turbo decodificación usando el ya mencionado algoritmo. La función planteada por Yufei, hace el llamado a varias sub-funciones para hacer el cálculo del Trellis y el ordenamiento de los bits así como su codificación, pero

[172] FUENTE: MOON, Todd ; Error Correction Coding: Mathematical Methods and Algorithms; Págs: 608-610.
[173] FUENTE: MOON, Todd ; Error Correction Coding: Mathematical Methods and Algorithms; Pág: 610.

debido a que este trabajo no se ha valido de estas sub-funciones se han realizado varios cambios que se detallan a continuación para poder adaptarla a nuestro sistema ya planteado. El cálculo del Trellis como ya se mencionó en al apartado 3.3.3.4.3 con la función expuesta por Yufei (para implementar el algoritmo *Log-MAP*) presenta varias complicaciones (debido al retardo) por lo que se decidió configurar el Trellis por medio de la función *poly2trellis*, de esta manera se pueden utilizar los parámetros entregados por dicha función para encontrar las respectivas trayectorias. Se ha aprovechado esta ventaja para poder realizar de manera más óptima el cálculo de los trazos (transiciones) tanto hacia atrás como hacia adelante, requeridos por el algoritmo; así como también poder hacer el cálculo de las salidas suaves antes y después de la divergencia de una manera más sencilla. Yufei, no considera un valor de confiabilidad del canal L_c en el cálculo de las métricas, lo que se modificó, ya que el sistema planteado así lo requiere. Se tuvo que modificar las tasas recibidas, ya que la función originalmente planteada solo permite procesar tasas en función de *k* y *n,* e inicialmente tiene como parámetro de entrada un *LLR*, que para el propósito planteado no es válido. Además se eliminó el llamado a todas las otras sub-funciones empaquetando todos los procesos en una misma función llamada ***sova.m***.

[174]La primera modificación del algoritmo SOVA con respecto al algoritmo convencional de Viterbi, se logra considerando que para cada estado en cada etapa del Trellis la métrica $M_{(A_s,B_s)}$, es calculada para las dos trayectorias (caminos) *A* o *B* que se unen en el estado *s* que se puede obtener a partir de la Ecuación 3.11:

$$M_{(A_s,B_s)} = M_{(A_{s-1},B_{s-1})} + \frac{1}{2}L_a + \frac{L_c}{2}\sum_{t=1}^{n} x_t y_t \qquad \textbf{(Ecuación 3.11)}$$

El camino con la métrica más grande es seleccionado como el sobreviviente, y para este estado (en esta etapa del Trellis) es almacenado un indicador para el estado previo a lo largo del camino sobreviviente, tal cual como se lo realizaba en

[174] FUENTE: WOODWARD, Jason P.; HANZO, Lajos; Comparative Study of Turbo Decoding Techniques: An Overview; Págs: 2216-2217.

el algoritmo clásico de Viterbi. Sin embargo, con el fin de permitir que la confiabilidad de que los bits decodificados sean calculados, la información L_c es almacenada para conseguir la salida suave.

El algoritmo SOVA (que tiene en cuenta a la información a-priori), una vez que calcula las métricas para ambos caminos convergentes, descarta el camino con la métrica más baja y hace el cálculo de la diferencia entre las métricas que está dado por la Ecuación 3.12.

$$\Delta^{s}_{A,B} = M_{(A_s)} - M_{(B_s)} \geq 0 \qquad \textbf{(Ecuación 3.12)}$$

El proceso, por el cual se busca todas las trayectorias posibles considerando a la diferencia de métricas se le denomina comúnmente como *transiciones hacia adelante* del algoritmo SOVA.

Previo al cálculo de las métricas de trayectoria, al igual que para el algoritmo MAP, en la sub-función *sova.m*, se ha configurado nuevamente el Trellis en modo *feedback* y sus resultados son procesados en varias matrices. Así la ***matriz0***, determina los valores correspondientes a las próximas salidas del Trellis; ***matriz1***, guarda los valores de los próximos estados del Trellis; mientras que ***matriz2*** y ***matriz3***, almacenan las salidas y estados previos del Trellis, respectivamente. Esta manipulación de la información obtenida por medio de la función *poly2trellis* se la realiza con el objeto de operar de manera más sencilla los datos que se han obtenido hasta el momento en el proceso de turbo decodificación y usarlos en posteriores cálculos del algoritmo. El Espacio de Código 3.34, muestra como la función *sova* realiza este proceso.

```
Long_Trama = length(L_a); % Calculo tamaño de la trama con los datos generados en
decodificador
[n,K] = size(G); % Cálculo tamaño Código Generador para obtener constraint length
cad_G = num2str(G); % Conversión código generador a cadena de caracteres
F = str2num(dec2base(bin2dec(cad_G),8)); % Conversión cadena de caracteres binaria a
octal
t = poly2trellis(K, F(2,1), F(1,1)); % Definición del Trellis
num_estado = t.numStates; % Número de estados

% Creación de matrices para poder procesar la información
% Próximas Salidas del Trellis
matriz0 = zeros(t.numStates, 4);
for columna = 1:4
    for i = 1:num_estado
```

```
        matriz0(i,1) = -1;
        matriz0(i,3) = 1;
        if t.outputs(i,1) == 0
        matriz0(i,2) = -1; % En la 2da columna las ramas del 0
        else
        matriz0(i,4) = -1; % En la 4ta columna las ramas del 0
        end
        if t.outputs(i,1) == 1
        matriz0(i,2) = +1; % En la 2da columna las ramas del 1
        else
        matriz0(i,4) = +1; % En la 4ta columna las ramas del 1
        end
    end
end

% Próximos Estados del Trellis
matriz1 = t.nextStates + 1;

% Busca cuales dos estados anteriores pueden llegar al estado actual
for b=0:1
   for estado = 1:num_estado
       % Previas Salidas del Trellis
       matriz2(matriz1(estado,b+1), b*2+1:b*2+2) = matriz0(estado, b*2+1:b*2+2);
       % Previos Estados del Trellis
       matriz3(matriz1(estado,b+1), b+1) = estado;
   end
end
```

Espacio de Código 3.34. Configuración del Trellis y Procesamiento de los Datos Recibidos.

El Espacio de Código 3.35, permite inicializar las trayectorias de las métricas para posteriormente hacer el cálculo de las mismas.

```
% Inicializa la trayectoria de las métricas hacia -Inft
Inft = 1e10; % Definición de Infinito para evitar NaN
for t = 1:Long_Trama+1
   for estado = 1:num_estado
       trayect_metrica(estado,t) = -Inft;
   end
end
```

Espacio de Código 3.35. Inicialización de las Trayectorias de las Métricas.

Mientras que el Espacio de Código 3.36, muestra como la función *sova* calcula los valores correspondientes a las métricas de trayectoria y las transiciones hacia adelante.

```
% Transiciones Hacia Adelante: para calcular todas las trayectorias de las métricas
trayect_metrica(1,1) = 0;
for t = 1:Long_Trama
    sist_pari = escala_bit(2*t-1:2*t); % Demultiplexo bit sistematico escalado (o bit
sistematico con interleaving)
                                       % y bit de paridad (1 ó 2) escalado
   for estado = 1:num_estado
       salida_0 = matriz2(estado,1:2);
       salida_1 = matriz2(estado,3:4);
       s0 = matriz3(estado,1);
       s1 = matriz3(estado,2);
       Ma = trayect_metrica(s0,t) - L_a(t)/2  +  L_c/2*sist_pari*salida_0';
       Mb = trayect_metrica(s1,t) + L_a(t)/2  +  L_c/2*sist_pari*salida_1';
```

```
        if Ma < Mb
          trayect_metrica(estado,t+1) = Mb;
          dif_trayect(estado,t+1) = Mb - Ma;
          prev_bit(estado,t+1) = 1;
        else
          trayect_metrica(estado,t+1) = Ma;
          dif_trayect(estado,t+1) = Ma - Mb;
          prev_bit(estado, t+1) = 0;
        end
    end
end
```

Espacio de Código 3.36. Cálculo de las Métricas y Transiciones Hacia Adelante.

[175]De esta manera, la métrica en el algoritmo SOVA es actualizada como en el algoritmo clásico de Viterbi, con el término adicional ***L_a*** incluído, de modo que la información a-priori válida sea tomada en cuenta. Nótese que este es un equivalente del cálculo de las métricas hacia delante del algoritmo MAP.

Mientras que el proceso, por el cual se busca el camino más probable con sus respectivas estimaciones de bit, se le denomina como *transiciones hacia atrás* del algoritmo SOVA y se lo ha implementado en Matlab por medio del Espacio de Código 3.37.

```
% Transiciones Hacia Atrás:
if decodificador == 1
   estado_ML(Long_Trama+1) = 1; % Decodificador #1: Transiciones Hacia Atrás desde todos
los estados cero
else
   estado_ML(Long_Trama+1) =
find(trayect_metrica(:,Long_Trama+1)==max(trayect_metrica(:,Long_Trama+1))); %
Decodificador #2: Transiciones Hacia Atrás desde el estado más probable
end

for t = Long_Trama:-1:1
    estimados(t) = prev_bit(estado_ML(t+1),t+1);
    estado_ML(t) = matriz3(estado_ML(t+1), estimados(t)+1); % Transiciones Hacia Atrás
para conseguir los bits estimados, y el camino más probable
end
```

Espacio de Código 3.37. Cálculo de las Transiciones Hacia Atrás.

[176]Ahora, la segunda modificación del algoritmo de Viterbi, es necesaria para calcular las salidas suaves. Para esto se debe considerar que, la diferencia $\Delta^{s}_{A,B} = M_{(A_s)} - M_{(B_s)} \geq 0$ entre las métricas de los caminos sobrevivientes y descartados están almacenados, junto con un vector binario que contiene delta +1

[175] FUENTE: WOODWARD, Jason P.; HANZO, Lajos; Comparative Study of Turbo Decoding Techniques: An Overview; Pág: 2217.
[176] FUENTE: WOODWARD, Jason P.; HANZO, Lajos; Comparative Study of Turbo Decoding Techniques: An Overview; Pág: 2216-2217.

bits (conocido también como *ventana*), el cual indica si el camino descartado habría dado la misma serie de bits como lo hace el camino sobreviviente. Esta serie de bits se la denomina secuencia de actualización.

[177]Cuando el algoritmo SOVA ha identificado el camino de Máxima Probabilidad (ML – *Maximum Likelihood*), la secuencia de actualización almacenada y las diferencias de las métricas a lo largo de este camino son usadas en la Ecuación 3.13:

$$LLR = u_t min\Delta^s_{A,B} \qquad \textbf{(Ecuación 3.13)}$$

Todo esto con el objeto de calcular los valores de las salidas suaves (*LLR*). En el Espacio de Código 3.38, se muestra como la función *sova* realiza el cálculo de las salidas suaves.

```
% Cálculo de las Salidas Suaves:
ventana = length(Long_Trama)+1; % Tamaño de la ventana SOVA
% Toma la decisión después de la espera "ventana".
for t = 1:Long_Trama
    suaves = Inft;
    for i = 0:ventana
        if t+i < Long_Trama+1
           bit = 1-estimados(t+i);
           estado_transicion = matriz3(estado_ML(t+i+1), bit+1);
           for j = i-1:-1:0 % Secuencia de Actualización
               bit = prev_bit(estado_transicion,t+j+1);
               estado_transicion = matriz3(estado_transicion, bit+1);
           end
           if bit ~= estimados(t)
              suaves = min(suaves,dif_trayect(estado_ML(t+i+1), t+i+1));
           end
        end
    end
    L_total(t) = (2*estimados(t)-1)*suaves; % Consigue la Soft Output (Salida Suavizada)
end
```

Espacio de Código 3.38. Cálculo de las Salidas Suaves.

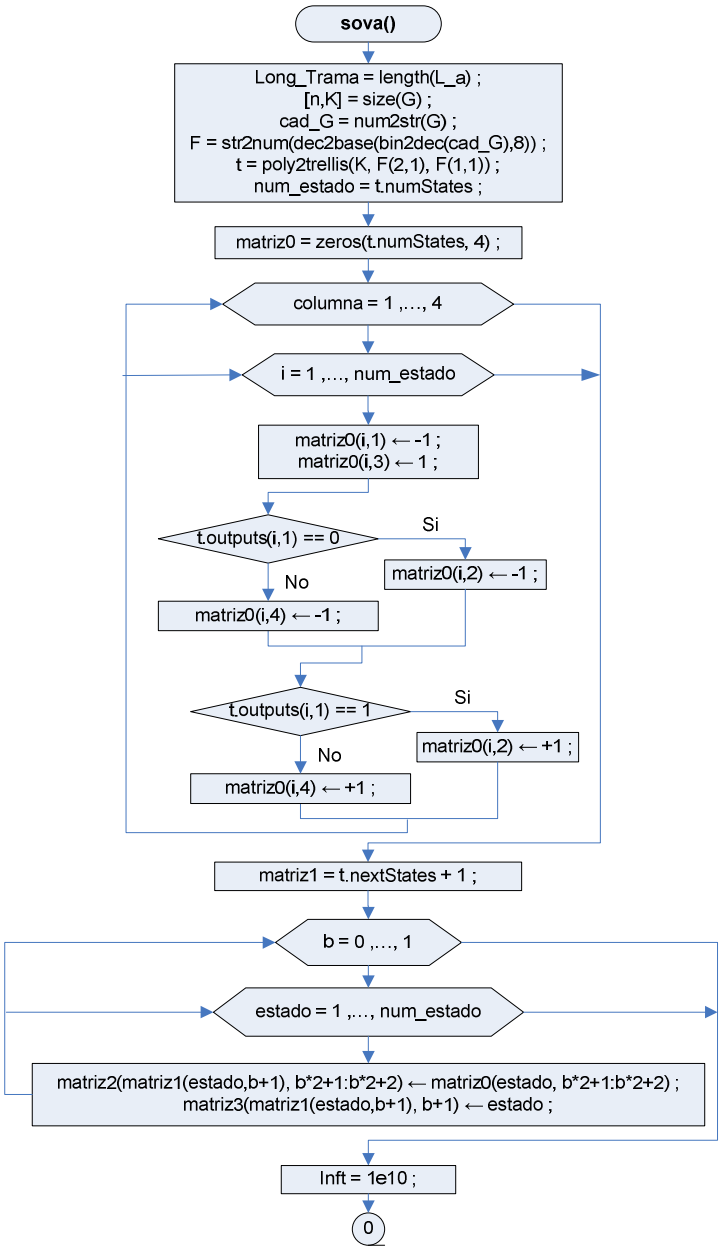
[178]En general la complejidad relativa de los algoritmos de decodificación SISO depende del *constraint length* de los códigos convolucionales utilizados en el proceso de Turbo Codificación. El peor desempeño tiene el algoritmo SOVA, a continuación el algoritmo Max-Log-MAP, luego el algoritmo Log-MAP, y finalmente el algoritmo MAP, siendo este último el algoritmo óptimo para el decodificador de

[177] FUENTE: WOODWARD, Jason P.; HANZO, Lajos; Comparative Study of Turbo Decoding Techniques: An Overview; Pág: 2217.
[178] FUENTE: FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G; Pág: 93.

Turbo Códigos pero también el más complejo. En canales AWGN el algoritmo Max-Log-MAP provee aproximadamente 0.5 dB más de ganancia de codificación que la que provee el algoritmo SOVA, y el algoritmo Log-MAP provee un aproximado de 0.5 dB más de ganancia de codificación que la que provee el algoritmo Max-Log-MAP.

3.3.3.4.9 Diagrama de Flujo de la Sub-función SOVA

El diagrama de flujo de la sub-función ***sova*** se ilustra por medio de la Figura 3.66 que se muestra a continuación:



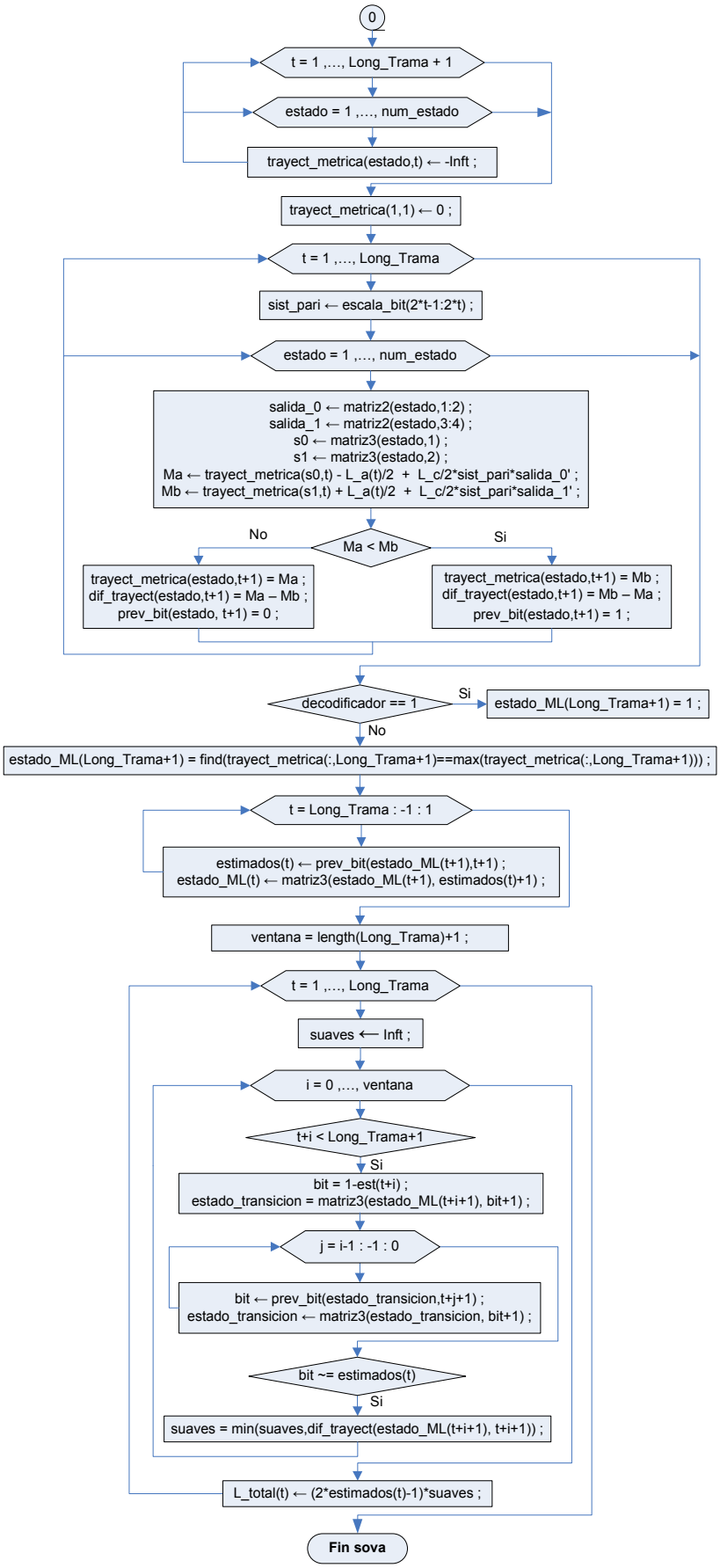


Figura 3.66. Diagrama de Flujo de la Función sova.

3.3.3.4.10 Interfaz Gráfica de Usuario

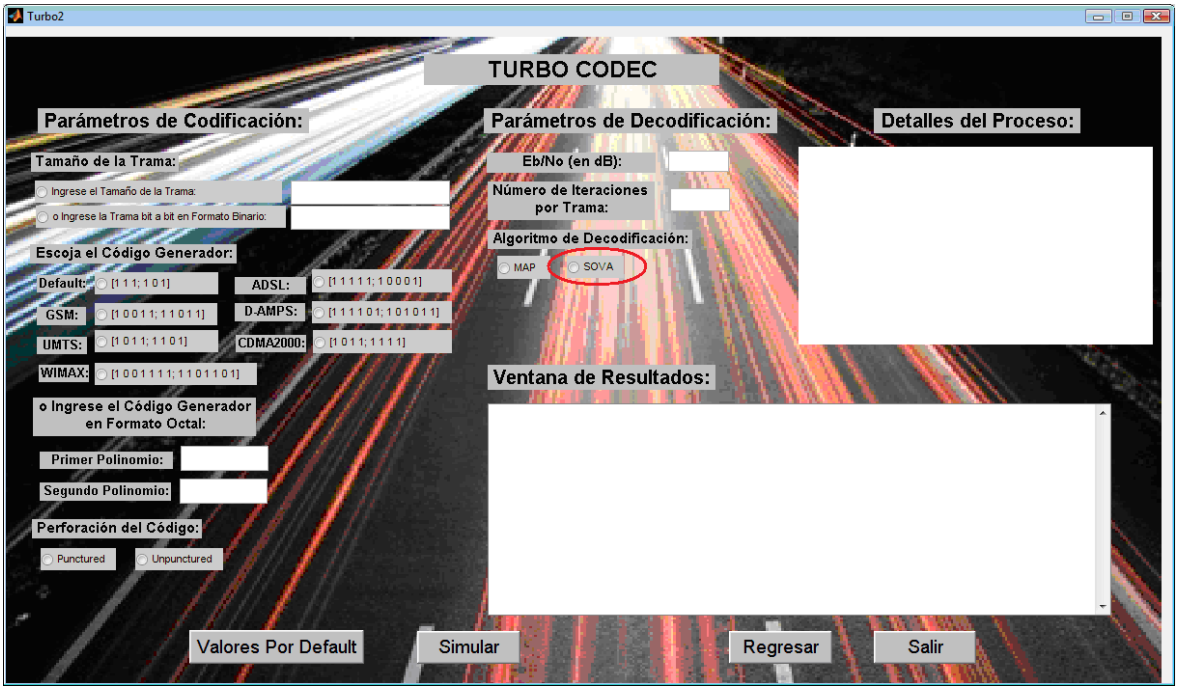


Figura 3.67. Implementación del Algoritmo de Decodificación SOVA en el Turbo Codec.

3.3.3.4.11 Implementación

El Espacio de Código 3.39 permite escoger el tipo de algoritmo de decodificación y determina los parámetros necesarios para el Decodificador 1. Es importante tener en cuenta que la complejidad total de un Turbo Decodificador dependerá de que tan eficientemente esté implementado el algoritmo de decodificación, ya que este genera las salidas suaves para los bits decodificados.

```
% Decodificador N°1
        if dec_alg == 0
           L_total = map(escala_bit(1,:), G, L_a, L_c, 1); % Información completa
        else
           L_total = sova(escala_bit(1,:), G, L_a, L_c, 1);  % Información completa
        end
```

Espacio de Código 3.39. Implementación del Decodificador 1.

3.3.3.5 Información Extrínseca Decodificador 1

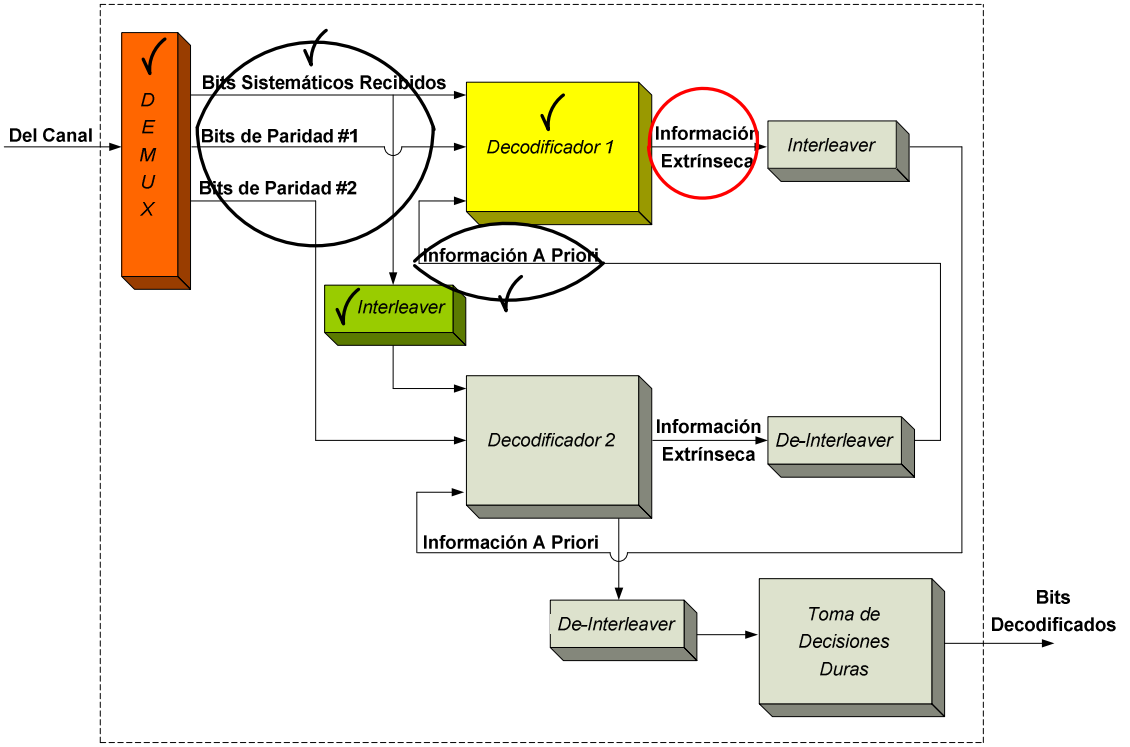


Figura 3.68. Ubicación en el diagrama de bloques de la Información Extrínseca Decodificador 1.

3.3.3.5.1 Descripción

[179]Del algoritmo de decodificación 1, se ha obtenido ***L_total***, la misma que está descrita por la Ecuación 3.14:

$$L\_total = L\_apriori + L\_canal + L\_extrinseca \qquad \textbf{(Ecuación 3.14)}$$

De esta expresión se puede extraer la información extrínseca, que en términos generales, es la información que provee un decodificador basándose en la secuencia de símbolos recibidos del canal (***escala_bit***) y en la secuencia de la información a-priori (***L_a***) proveniente del otro decodificador. Típicamente, un decodificador provee esta información utilizando las restricciones impuestas a la secuencia transmitida por el código usado. Particularmente, la información extrínseca producida por un decodificador SOVA está altamente correlacionada

[179] FUENTE: LANGTON, Charan; Intuitive Guide to Principles of Communications: Turbo Coding and MAP Decoding, Parte 1; Pág: 22.

con la información a-priori que ingresó al decodificador, lo cual se debe a que el algoritmo SOVA produce la salida suave considerando, en vez de todas las trayectorias, únicamente dos, la de máxima probabilidad y la segunda mejor trayectoria. Esto ocasiona que el desempeño ofrecido por SOVA no sea óptimo. Tanto para la tasa de código de 1/2 como 1/3, la información extrínseca del decodificador 1 se almacena en el vector ***Informacion_Extrinseca_Dec_1***.

```
Informacion_Extrinseca_Dec_1 =

    6.9095    6.1012   -7.9963    6.8847    8.6528
```

Figura 3.69. Ejecución de Prueba para la Visualización de la Información Extrínseca del Decodificador 1 para la Primera Iteración con Tasa 1/3.

3.3.3.5.2 Implementación

```
% Decodificador N°1
      L_e = L_total - 2*escala_bit(1,1:2:2*Long_Trama) - L_a;  % Información extrínseca
      Informacion_Extrinseca_Dec_1 = L_e
```

Espacio de Código 3.40. Implementación de la Información Extrínseca Decodificador 1.

3.3.3.6 Interleaving Información Extrínseca Decodificador 1

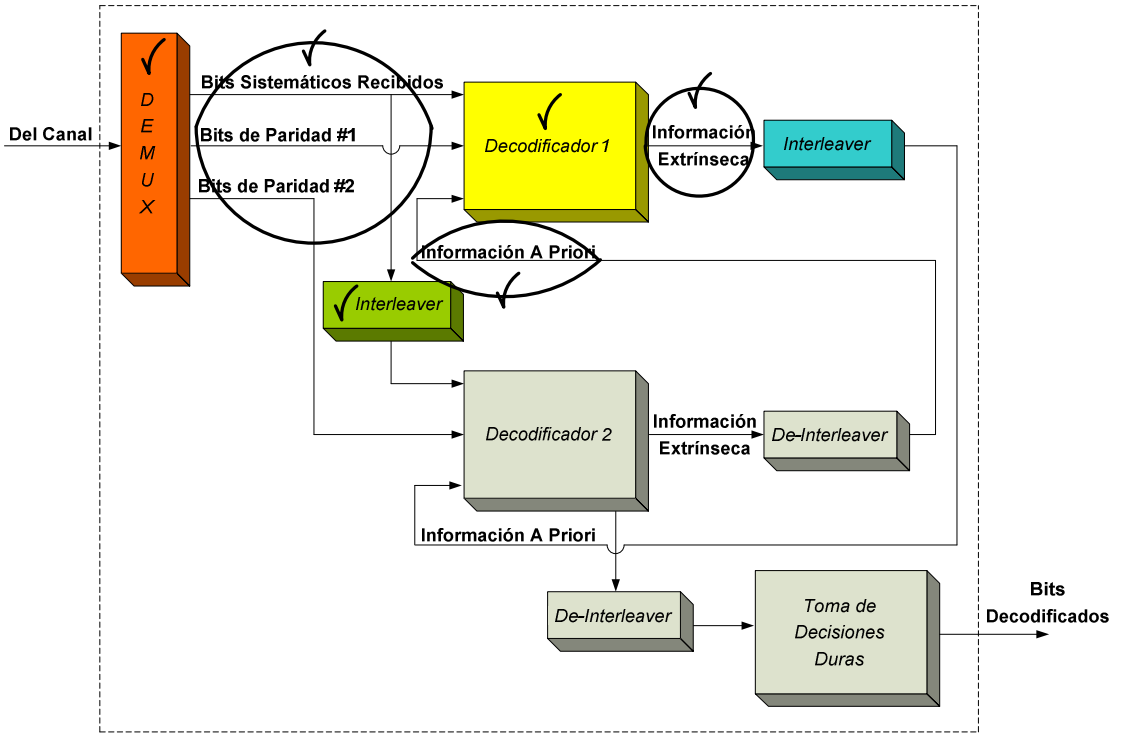


Figura 3.70. Ubicación en el diagrama de bloques del Interleaver de la Información Extrínseca.

3.3.3.6.1 Descripción

Luego de obtener la información extrínseca del decodificador 1, se realiza un interleaving de esta información; al igual que en los anteriores interleavers se usa el estado inicial ***104***. La información extrínseca del decodificador 1 con interleaving, se almacena en ***Informacion_Extrinseca_Dec_1_Con_Interleaving*** La combinación de un Interleaver junto con la utilización del concepto de información extrínseca, permite que la información a-priori empleada por los decodificadores componentes del Turbo Decodificador esté muy poco correlacionada con la secuencia de entrada a cada uno de ellos, haciendo de la decodificación iterativa una estrategia de decodificación bastante óptima.

Figura 3.71. Ejecución de Prueba para la Visualización de la Información Extrínseca del Decodificador 1 con Interleaving para la Primera Iteración y Tasa 1/3.

3.3.3.6.2 Implementación

```
% Decodificador N°2
        L_a = randintrlv(L_e,104);  % Información a priori
        Informacion_Extrinseca_Dec_1_Con_Interleaving = L_a
```

Espacio de Código 3.41. Implementación del Interleaver.

3.3.3.7 Información A-Priori Decodificador 2

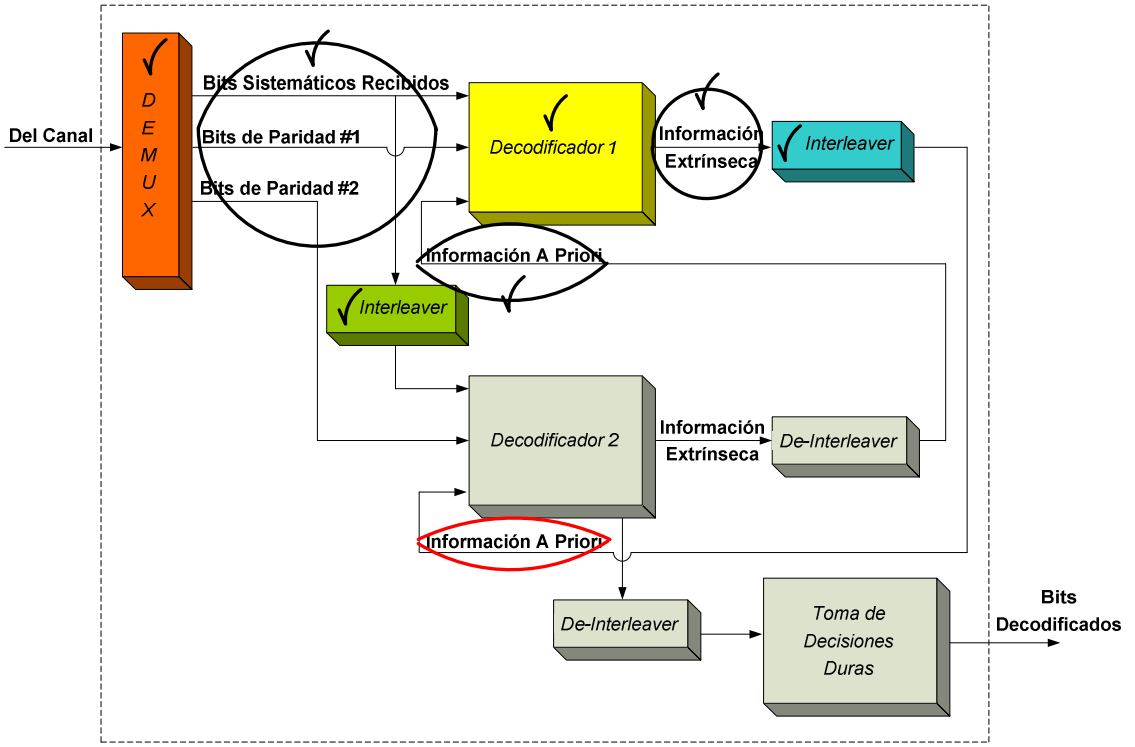


Figura 3.72. Ubicación en el diagrama de bloques de la Información A Priori del Decodificador 2.

3.3.3.7.1 Descripción

Como la Figura 3.72 muestra, la información a priori del decodificador 2 (almacenado en el vector ***Informacion_A_Priori_Dec_2***), es la misma que resulta del proceso de interleaving de la información extrínseca del decodificador 1, que para el caso en cuestión se almacena en el vector ***Informacion_Extrinseca_Dec_1_Con_Interleaving***.

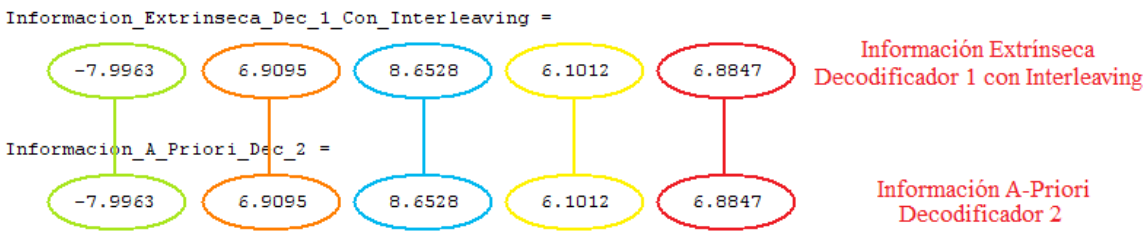


Figura 3.73. Ejecución de Prueba para la Visualización de la Información A Priori del Decodificador 2 para la Primera Iteración con Tasa 1/3.

3.3.3.7.2 Implementación

```
% Decodificador N°2
        L_a = randintrlv(L_e,104);  % Información a priori
        Informacion_Extrinseca_Dec_1_Con_Interleaving = L_a
        Informacion_A_Priori_Dec_2 = L_a
```

Espacio de Código 3.42. Implementación de la Información A Priori del Decodificador 2.

3.3.3.8 Decodificador 2

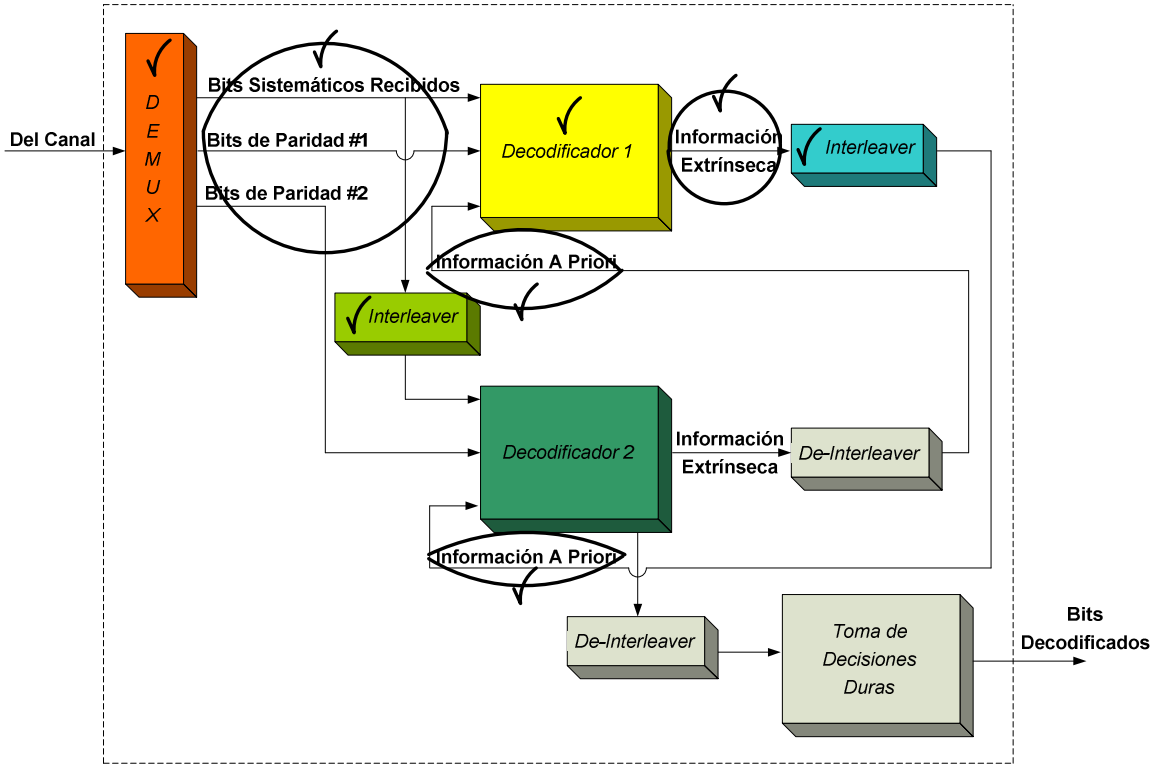


Figura 3.74. Ubicación en el diagrama de bloques del Decodificador 2.

3.3.3.8.1 Descripción

El algoritmo embebido en el decodificador 2 de la Figura 3.74, entra en operación al recibir la información a priori del decodificador 1, la secuencia del canal que contiene la versión suave permutada (pasada por el interleaver) y escalada de los bits sistemáticos y la versión suave escalada de los bits de paridad # 2, estos dos últimos guardados en la segunda fila de la matriz ***escala_bit***.

3.3.3.8.2 Implementación

El Espacio de Código 3.43 permite escoger el tipo de algoritmo de decodificación según los datos recibidos y determina los parámetros necesarios para el Decodificador 2.

```
% Decodificador N°2
        if dec_alg == 0
           L_total = map(escala_bit(2,:), G, L_a, L_c, 2);  % Información completa
        else
           L_total = sova(escala_bit(2,:), G, L_a, L_c, 2);  % Información completa
        end
```

Espacio de Código 3.43. Implementación del Decodificador 2.

3.3.3.9 Información Extrínseca Decodificador 2

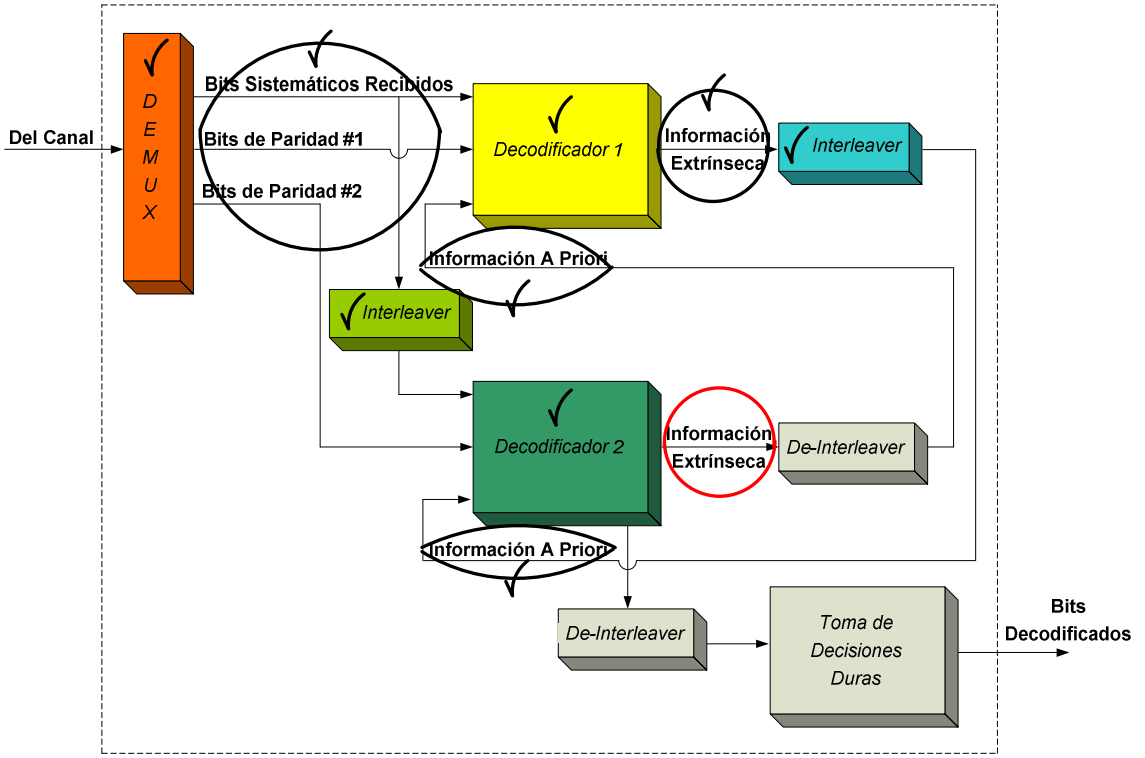


Figura 3.75. Ubicación en el diagrama de bloques de la Información Extrínseca Decodificador 2.

3.3.3.9.1 Descripción

La información extrínseca del decodificador 2 es el resultado de la sustracción de los de bits sistemáticos con interleaving escalados e información a-priori, con

respecto al vector ***L_total*** (entregado por el algoritmo del decodificador 2, luego del proceso de decodificación). Para cualquiera de los valores de tasa de código, la información extrínseca del decodificador 2 está almacenada en el vector ***Informacion_Extrinseca_Dec_2*.**

```
Informacion_Extrinseca_Dec_2 =

    -6.4746    4.308    -1.5934    -0.4933    -0.76175
```

Figura 3.76. Ejecución de Prueba para la Visualización de la Información Extrínseca del Decodificador 2, para la Primera Iteración con Tasa 1/3.

3.3.3.9.2 Implementación

```
% Decodificador N°2
        L_e = L_total - 2*escala_bit(2,1:2:2*Long_Trama) - L_a; % Información extrínseca
        Informacion_Extrinseca_Dec_2 = L_e
```

Espacio de Código 3.44. Implementación de la Información Extrínseca Decodificador 2.

3.3.3.10 De-Interleaver

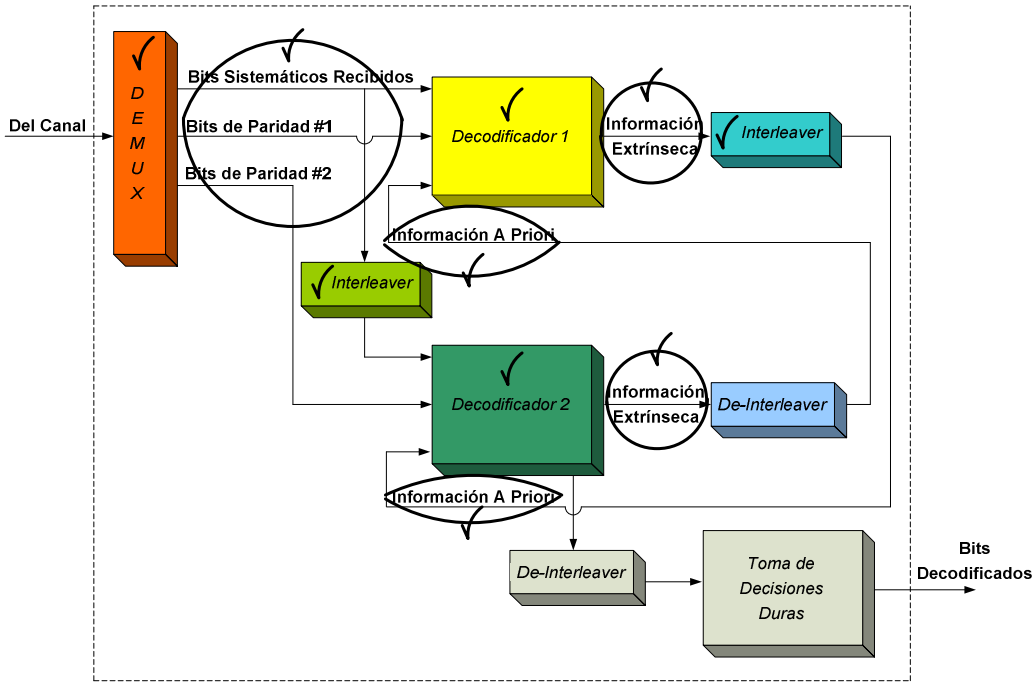


Figura 3.77. Ubicación en el diagrama de bloques del De-Interleaver.

3.3.3.10.1 Descripción

Luego de obtener la información extrínseca del decodificador 2, se realiza un de-interleaving de esta información. El de-interleaving es el proceso inverso al interleaving, y es necesario para poder recuperar la secuencia original de los bits. Para el presente caso se lo hace mediante la función *randdeintrlv* que proporciona Matlab R2008a, utilizando el mismo estado inicial ***104*** que se aplicó en el interleaving, estos valores son almacenados en el vector ***Informacion_Extrinseca_Dec_2_Con_Deinterleaving*** cuyos valores coinciden con el vector ***Informacion_A_Priori_Dec_1***. Como ya se mencionó en el apartado 3.3.3.1 de este proyecto de titulación, este vector es inicializado a cero para la primera iteración.

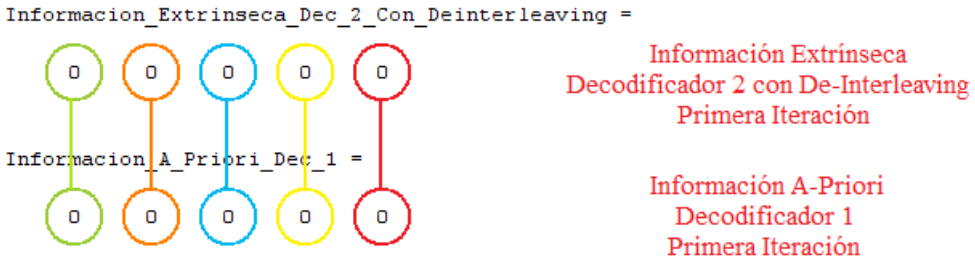


Figura 3.78. Ejecución de Prueba para la Visualización de la Información Extrínseca del Decodificador 2 con De-Interleaving para la Primera Iteración y Tasa de 1/3.

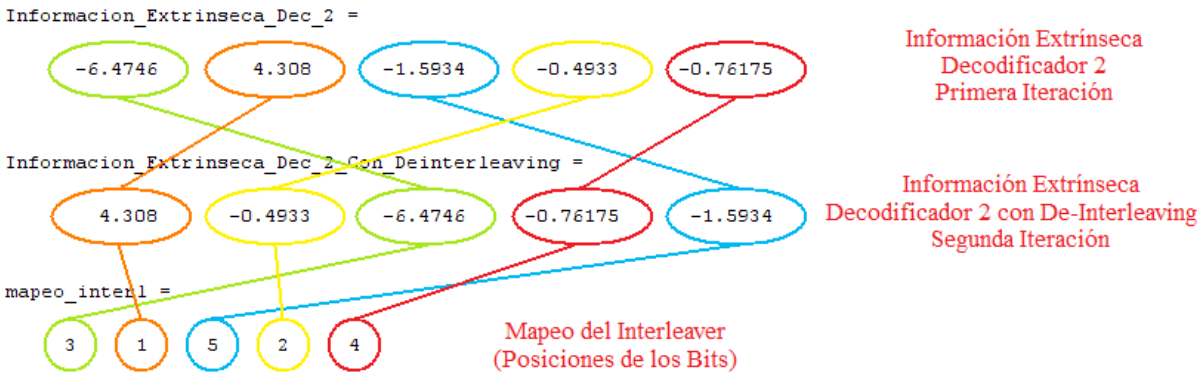


Figura 3.79. Ejecución de Prueba para la Visualización de la Información Extrínseca del Decodificador 2 con De-Interleaving para la Segunda Iteración y Tasa de 1/3.

3.3.3.10.2 Implementación

```
% Decodificador N°1
        L_a = randdeintrlv(L_e,104);
        Informacion_Extrinseca_Dec_2_Con_Deinterleaving = L_a
        Informacion_A_Priori_Dec_1 = L_a
```

Espacio de Código 3.45. Implementación del De-Interleaver.

3.3.3.11 Terminación del Proceso de Iteración (Valores de Salida)

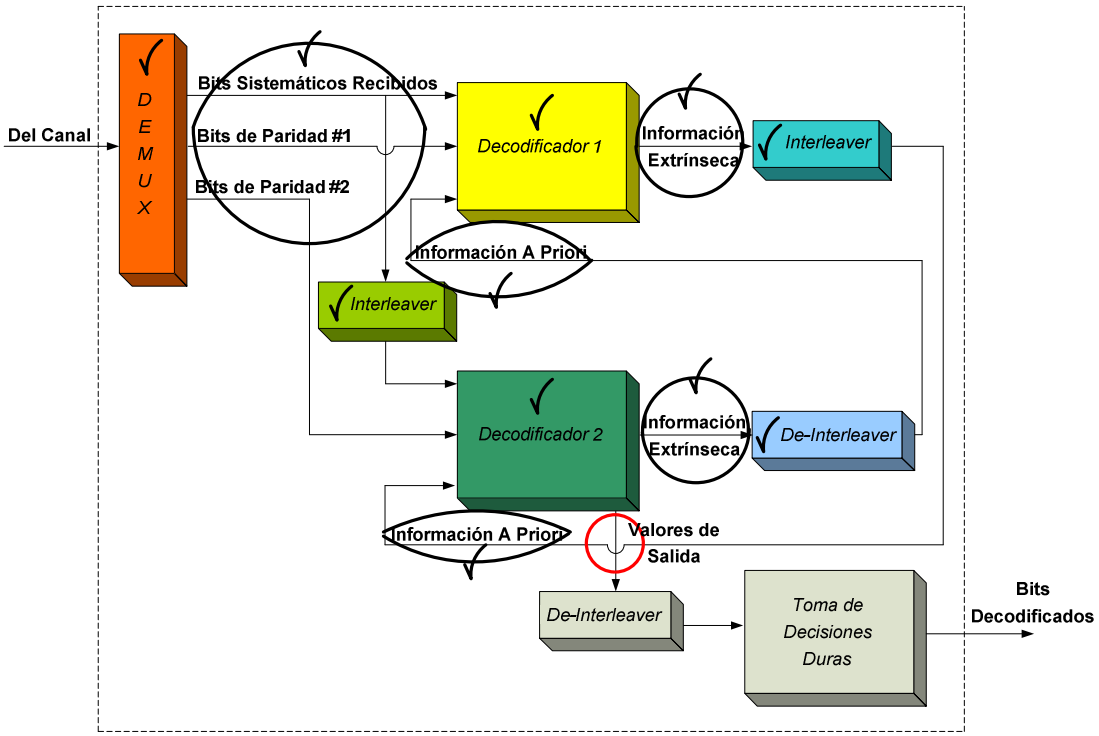


Figura 3.80. Ubicación en el diagrama de bloques de los Valores de Salida.

3.3.3.11.1 Descripción

Una vez que se ha cumplido con el número de iteraciones de trama que el usuario considere conveniente, se termina el proceso iterativo de la turbo decodificación y se entrega como salida por parte del algoritmo del decodificador 2 el vector ***L_total***, que para el caso en cuestión toma el nombre de ***Valor_De_Salida***.

```
Valor_De_Salida =

   -15.305    14.467     7.5433     9.666     8.3739
```

Figura 3.81. Ejecución de Prueba para la Visualización de los Valores de Salida en la Primera Iteración con Tasa 1/3.

3.3.3.11.2 Implementación

```
Valor_De_Salida = L_total
```

Espacio de Código 3.46. Implementación de los Valores de Salida.

3.3.3.12 De-Interleaver

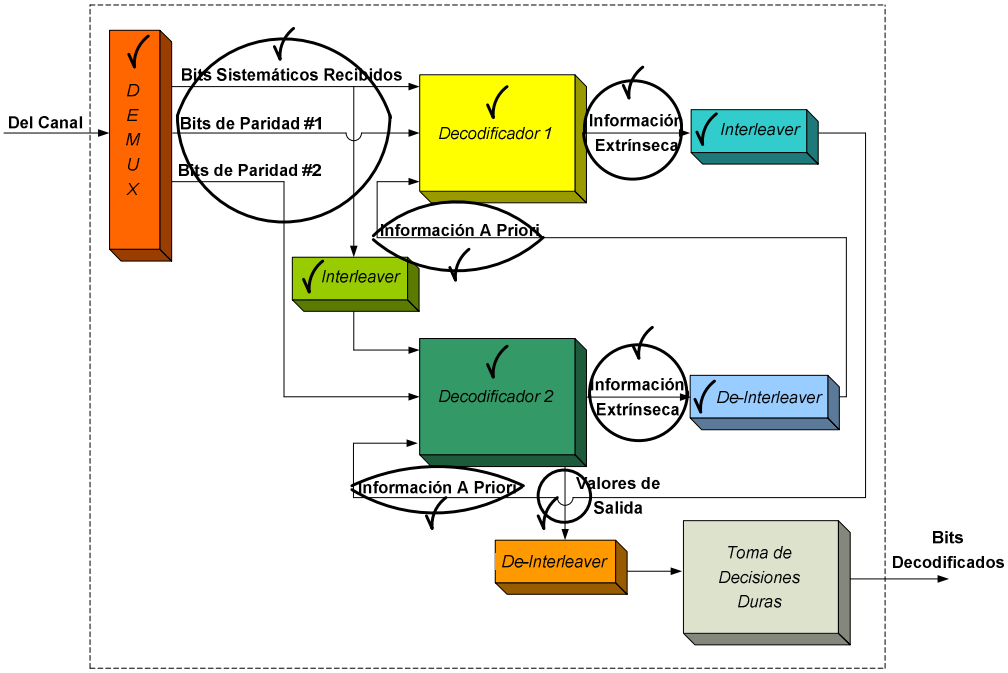


Figura 3.82. Ubicación en el diagrama de bloques del De-Interleaver.

3.3.3.12.1 Descripción

Al vector ***Valor_De_Salida*** se le realiza un proceso de de-interleaving, por medio de la función *randdeintrlv* conservando el estado inicial ***104***; y se lo guarda en el vector ***Valor_De_Salida_Deinterleaver.*** Todo esto con el objeto de proporcionar los bits decodificados en el orden correcto a la salida del Turbo Decodificador.

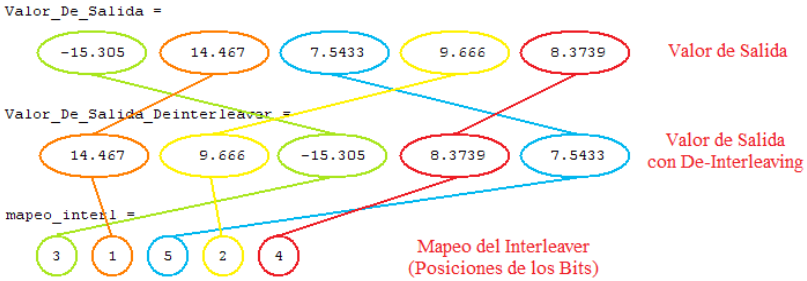


Figura 3.83. Ejecución de Prueba para la Visualización de los Valores de Salida con De-Interleaving en la Primera Iteración con Tasa 1/3.

3.3.3.12.2 Implementación

```
Valor_De_Salida_Deinterleaver = randdeintrlv(L_total,104)
```

Espacio de Código 3.47. Implementación del De-Interleaver.

3.3.3.13 Toma de Decisiones Duras

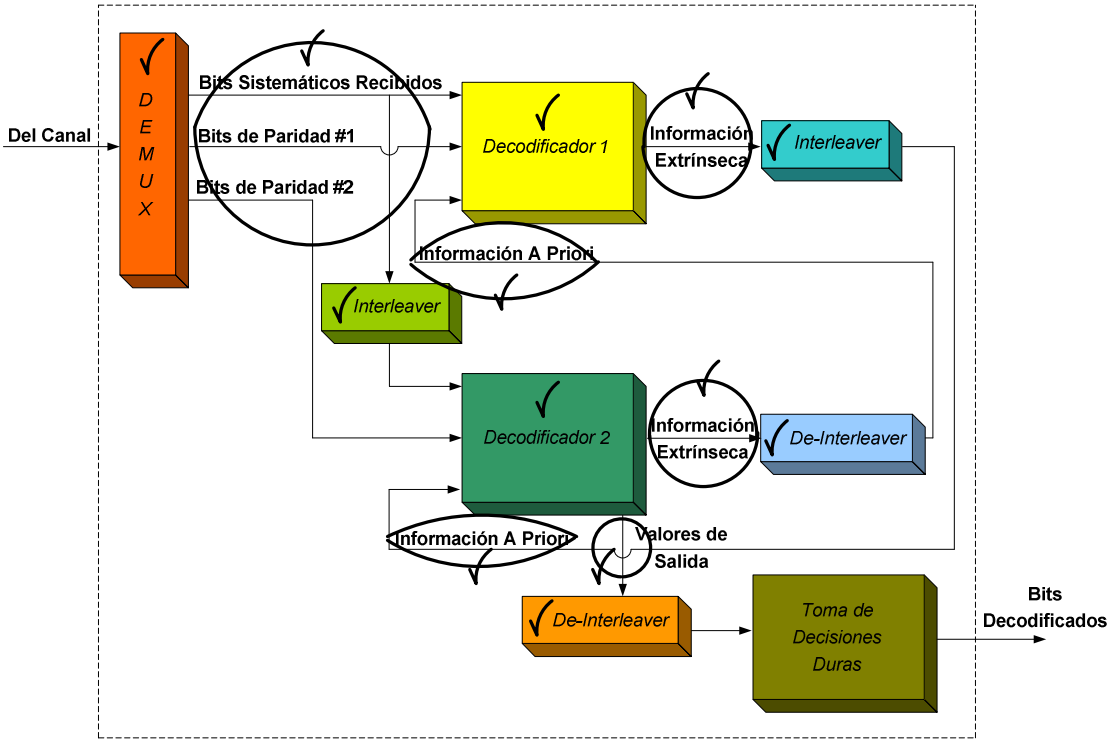


Figura 3.84. Ubicación en el diagrama de bloques de la Toma de Decisiones Duras.

3.3.3.13.1 Descripción

Los LLR´s del vector ***Valor_De_Salida_Deinterleaver*** son convertidos en 1´s y 0´s mediante una toma de decisiones duras, para lo cual primero se hace una estimación de bit de los valores de salida con de-interleaving. El *Estimador de la LLR de cada Bit,* tiene como función la de calcular las *log-likelihood ratios* para cada bit recibido. Para cualquier tasa de código la estimación de bit, de los valores de salida con de-interleaving son los valores unitarios de cada uno de estos números, con su respectivo signo.

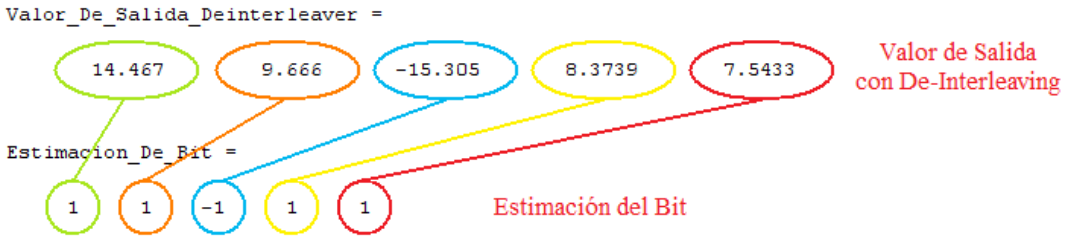


Figura 3.85. Ejecución de Prueba para la Visualización de los Valores de Salida con De-Interleaving en la Primera Iteración con Tasa 1/3.

3.3.3.13.2 Implementación

```
Estimacion_De_Bit = sign(Valor_De_Salida_Deinterleaver)
```

Espacio de Código 3.48. Implementación de la Toma de Decisiones Duras.

3.3.3.14 Bits Decodificados

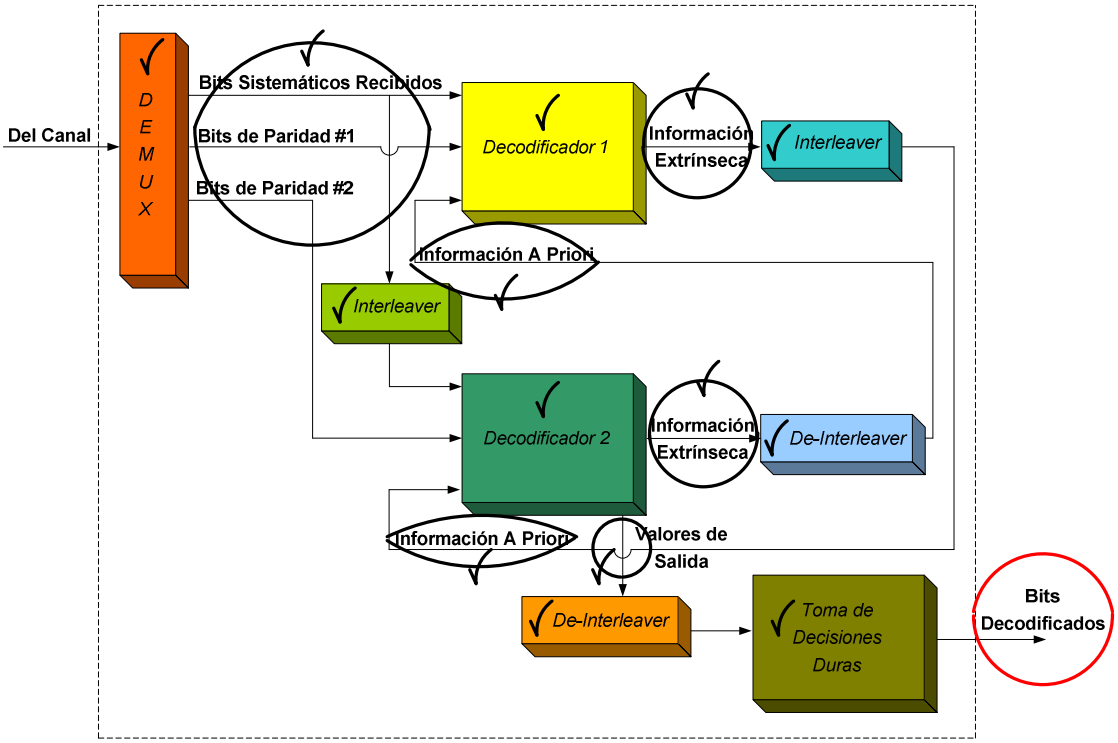


Figura 3.86. Ubicación en el diagrama de bloques de los Bits Decodificados.

3.3.3.14.1 Descripción

Mostrar los bits decodificados termina el proceso de turbo decodificación, siendo el vector ***Bits_Decodificados*** el que almacena a dichos bits. A los valores que se

obtuvieron en la estimación de bit (+1 ó -1), y bajo el criterio que determinan las LLR (positivas y negativas), se puede establecer que:

$$\frac{\text{Estimación Bit} + 1}{2} = \frac{\pm 1 + 1}{2} \qquad \textbf{(Ecuación 3.15)}$$

Fijándose de esta manera, cada uno de los bits decodificados de la trama.

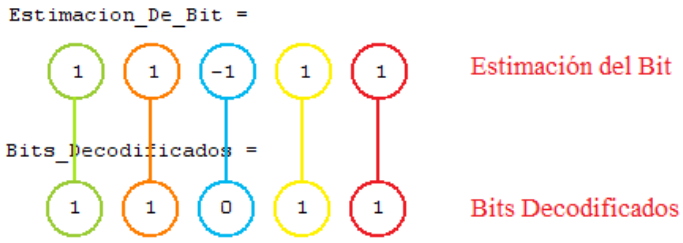


Figura 3.87. Ejecución de Prueba para la Visualización de los Bits Decodificados.

Los bits de información que se obtuvieron en primera instancia (*u*), son comparados con los bits decodificados sin cola; si se constata que los valores son iguales se demuestra que la secuencia de entrada luego de un proceso de turbo codificación y turbo decodificación es la misma, por lo que se contabiliza como trama no errada.

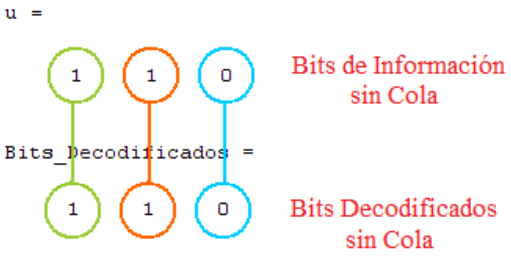


Figura 3.88. Ejecución de Prueba para la Comparación de los Bits de Información con los Bits Decodificados.

Pero si la trama de información al ser comparada con la trama decodificada, no es igual, se determina que la trama llego errada y se la contabiliza.

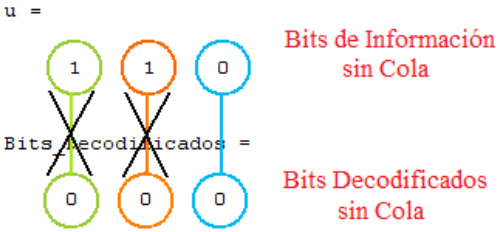


Figura 3.89. Ejecución de Prueba para la Visualización de las Tramas Erradas.

3.3.3.14.2 Implementación

Los Bits Decodificados son implementados en el Espacio de Código 3.49.

```
        u_deco = (Estimacion_De_Bit+1)/2;
        Bits_Decodificados = u_deco
```

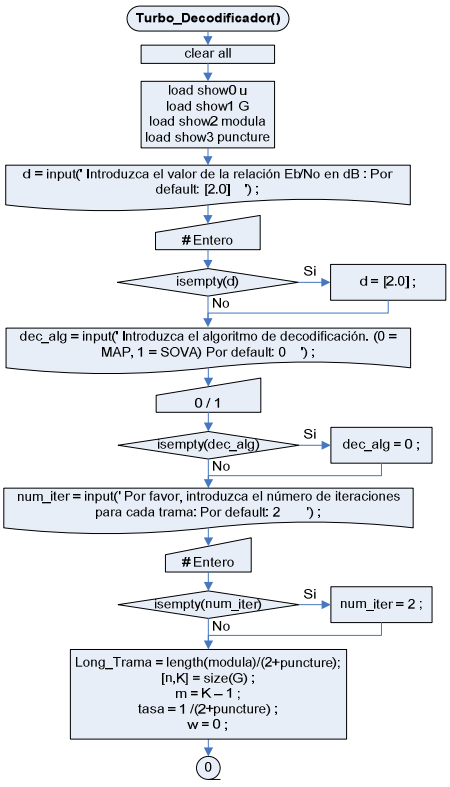
Espacio de Código 3.49. Implementación de los Bits Decodificados.

Mientras que la comparación de secuencia de entrada y salida para contabilizar el número de tramas erradas se la realiza en el Espacio de Código 3.50.

```
% Número de bits errados en la actual iteración
        errores(iter) = length(find(u_deco(1:Long_Trama-m)~=u));

% Limpia el contador de tramas erradas
        num_tramas_erradas(nEN,1:num_iter) = zeros(1,num_iter);

% Contador de tramas erradas para la actual iteración
        if errores(iter)>0
           num_tramas_erradas(nEN,iter) = num_tramas_erradas(nEN,iter)+1;
        end
        [error, ber] = biterr(u,u_deco(1:Long_Trama-m));
        ber_vector(iter,1) = ber;
```

Espacio de Código 3.50. Implementación del Contador de Tramas Erradas.

3.3.2.8.4 Diagrama de Flujo del Turbo Decodificador



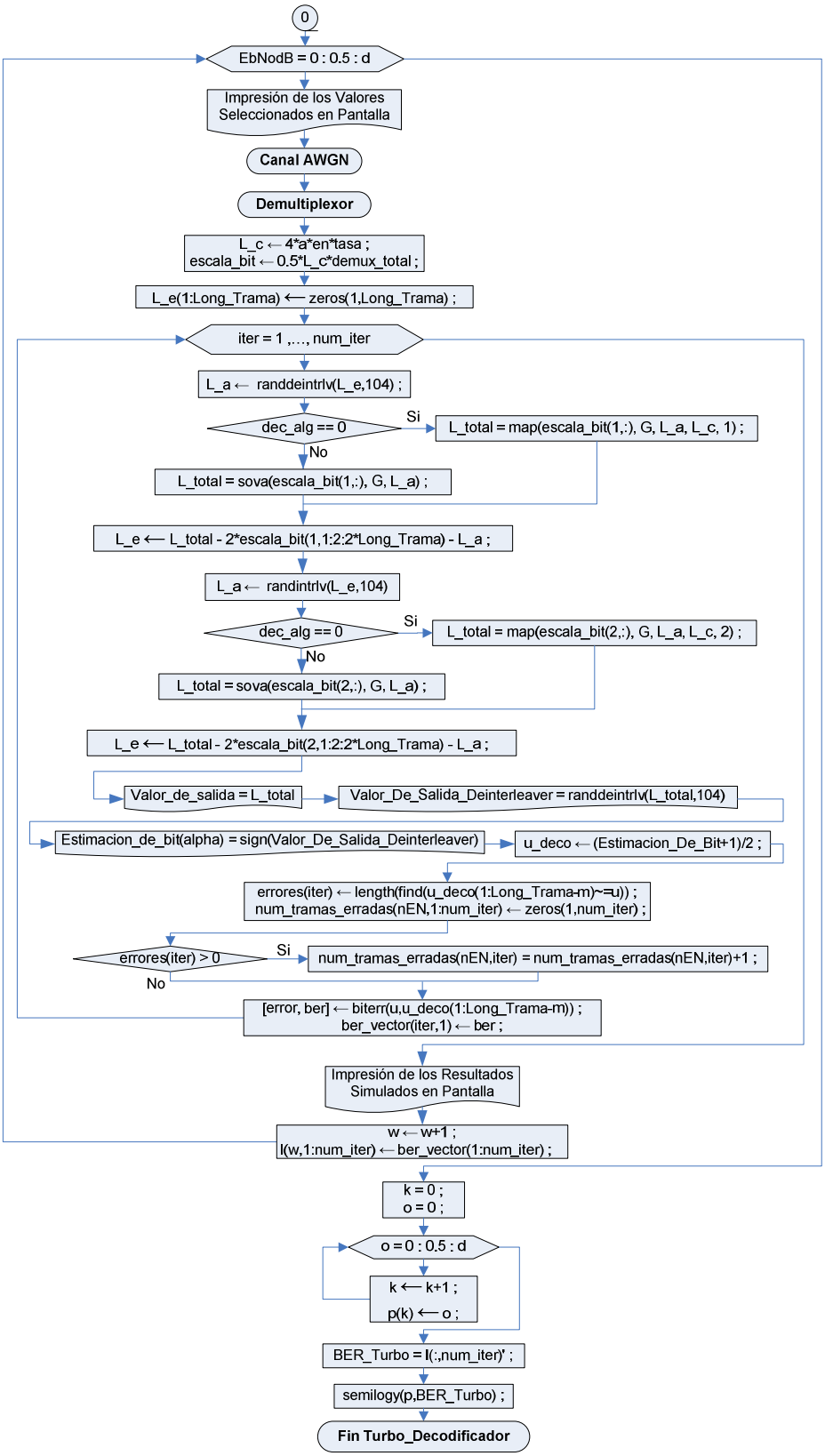
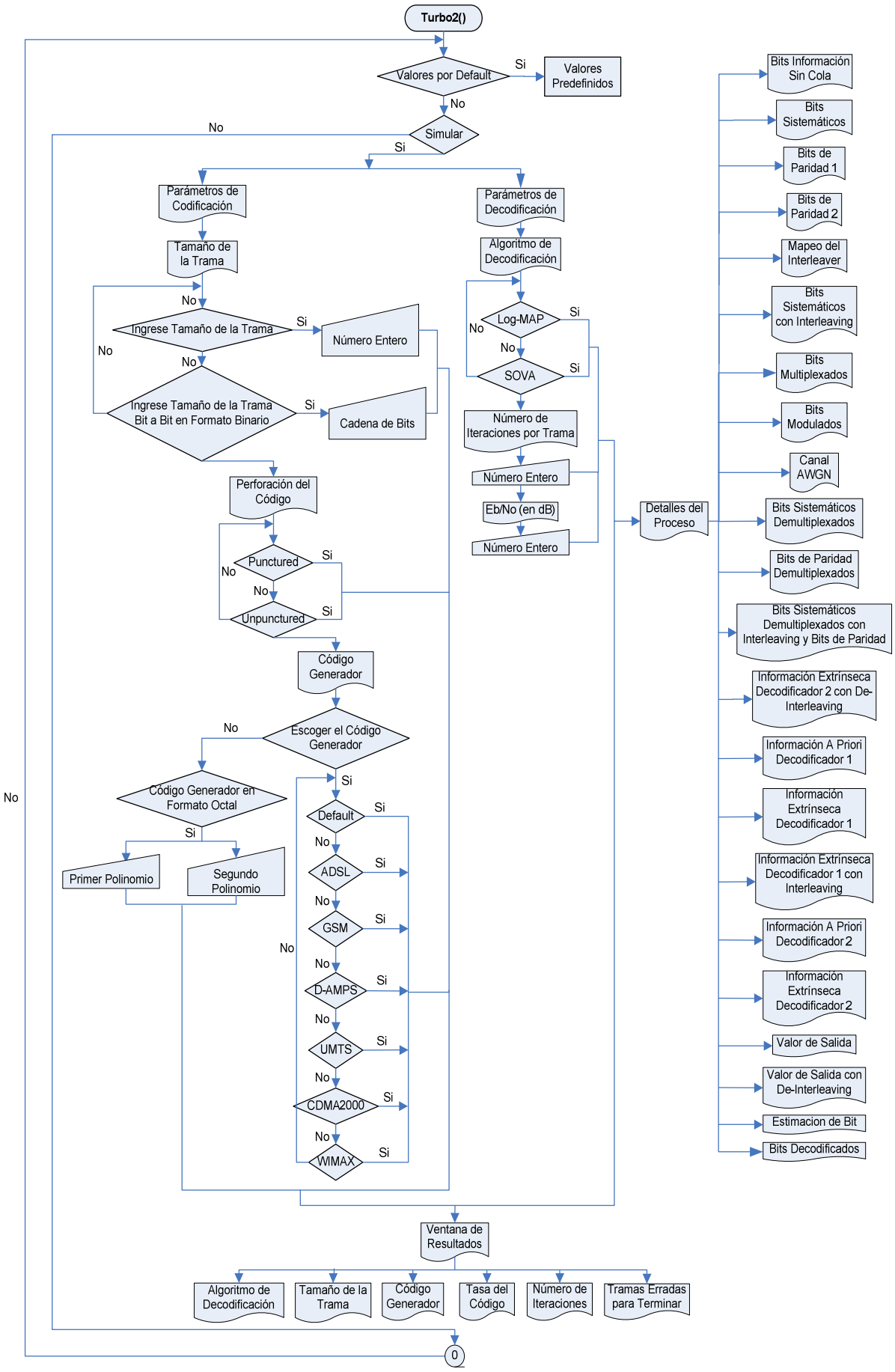


Figura 3.90. Diagrama de Flujo de la Función Turbo_Decodificador.

El diagrama de flujo de la Figura 3.91 explica el funcionamiento completo de la segunda ventana de la interfaz gráfica para la Simulación del Turbo Codec:


Turbo2()
Valores por Default
Si
Valores Predefinidos
No
Simular
No
Si
Parámetros de Codificación
Tamaño de la Trama
No
Ingrese Tamaño de la Trama
Si
Número Entero
No
No
Ingrese Tamaño de la Trama Bit a Bit en Formato Binario
Si
Cadena de Bits
Perforación del Código
Punctured
Si
No
No
Unpunctured
Si
Código Generador
No
Escoger el Código Generador
Código Generador en Formato Octal
Si
Primer Polinomio
Segundo Polinomio
Si
Default
Si
No
ADSL
Si
No
GSM
Si
No
No
D-AMPS
Si
No
UMTS
Si
No
CDMA2000
Si
No
WIMAX
Si
Parámetros de Decodificación
Algoritmo de Decodificación
No
Log-MAP
Si
No
SOVA
Si
Número de Iteraciones por Trama
Número Entero
Eb/No (en dB)
Número Entero
Detalles del Proceso
Bits Información Sin Cola
Bits Sistemáticos
Bits de Paridad 1
Bits de Paridad 2
Mapeo del Interleaver
Bits Sistemáticos con Interleaving
Bits Multiplexados
Bits Modulados
Canal AWGN
Bits Sistemáticos Demultiplexados
Bits de Paridad Demultiplexados
Bits Sistemáticos Demultiplexados con Interleaving y Bits de Paridad
Información Extrínseca Decodificador 2 con De-Interleaving
Información A Priori Decodificador 1
Información Extrínseca Decodificador 1
Información Extrínseca Decodificador 1 con Interleaving
Información A Priori Decodificador 2
Información Extrínseca Decodificador 2
Valor de Salida
Valor de Salida con De-Interleaving
Estimacion de Bit
Bits Decodificados
Ventana de Resultados
Algoritmo de Decodificación
Tamaño de la Trama
Código Generador
Tasa del Código
Número de Iteraciones
Tramas Erradas para Terminar
0

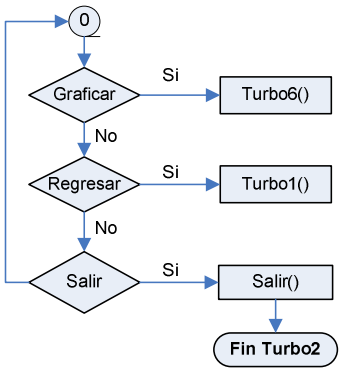


Figura 3.91. Diagrama de Flujo de la Ventana de Simulación del Turbo Codec.

3.3.4 VENTANA DE GRAFICACIÓN DE LAS CURVAS DE RENDIMIENTO

Es menester presentar de manera gráfica los resultados que se han obtenido del proceso de Turbo Codificación / Decodifiación, por esta razón se ha creado una ventana en la cual se muestra la curva de rendimiento BER vs. E_b/N_o. Esta ventana se ejecuta al presionar el botón graficar en el Turbo Codec. La Figura 3.92 muestra la ubicación de dicho botón, que también culmina la implementación de la ventana del Turbo Codec.

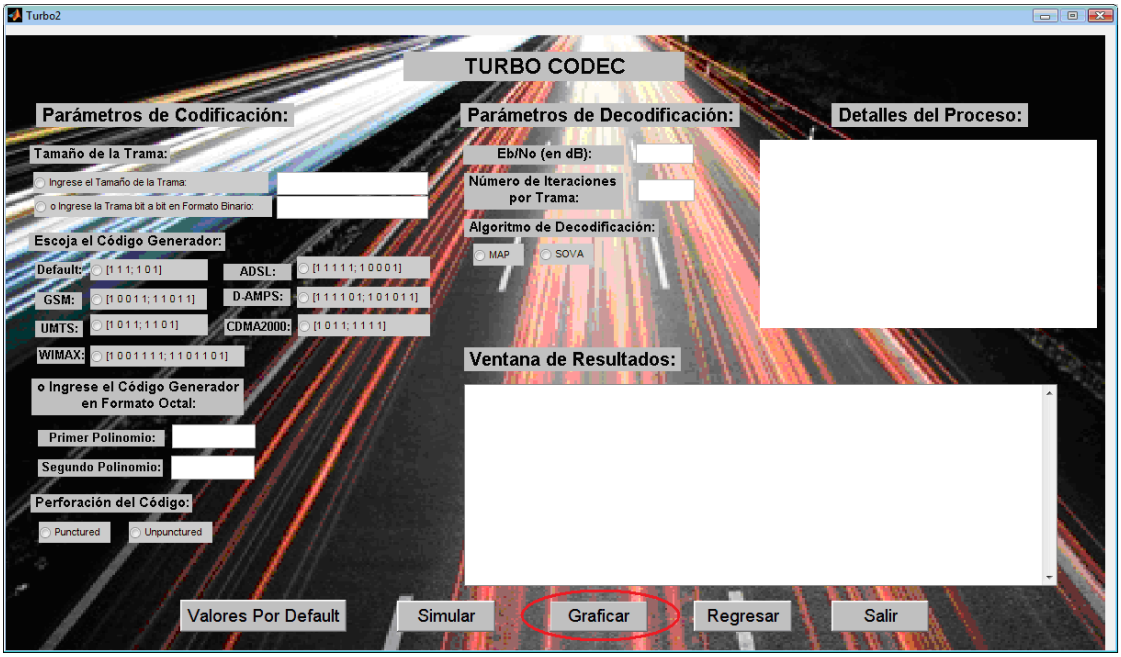


Figura 3.92. Ventana Final de Implementación del Turbo Codec.

La Figura 3.93 muestra la ventana que permite graficar las curvas de rendimiento con base a los parámetros ingresados en el Turbo Codec.

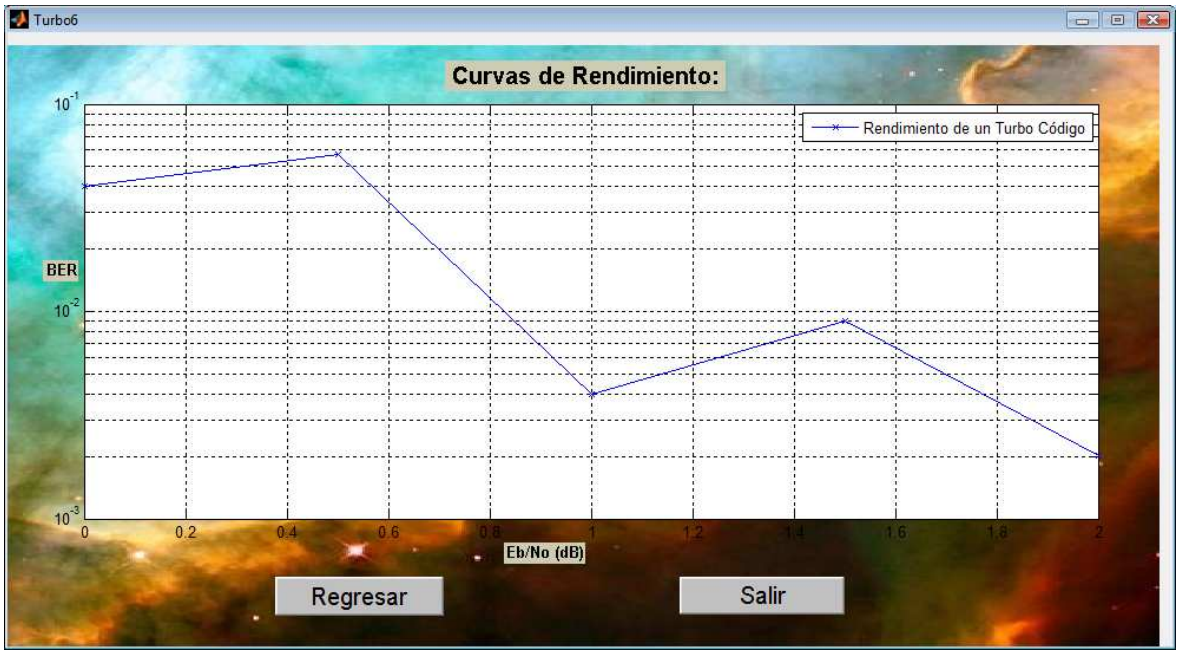


Figura 3.93. Ventana de Gráficas: BER vs. Eb/No.

Se han agregado dos botones que permiten *Regresar* a la ventana de Simulación del Turbo Codec y *Salir* definitivamente del programa.

La Figura 3.94 explica mediante el diagrama de flujo el funcionamiento de la ventana de Gráficas:

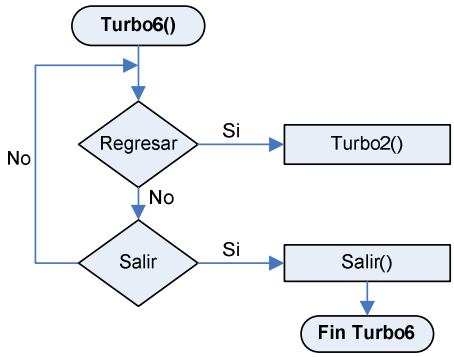


Figura 3.94. Diagrama de Flujo de la Ventana de Gráficas.

3.3.5 VENTANAS DE AYUDA

Mediante el uso del ***Menu Editor*** del MATLAB, se han creado tres sub-menúes en la ventana de Simulación del Turbo Codec, que permiten acceder a *Ayudas* para la ejecución del programa.

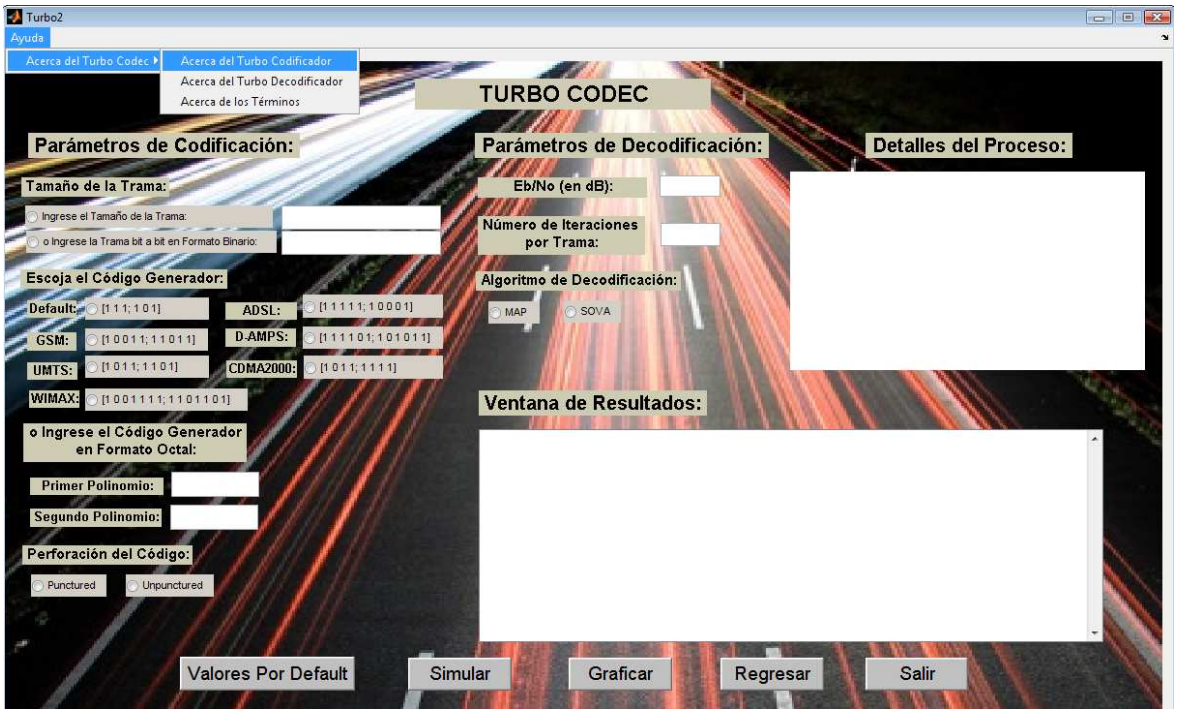


Figura 3.95. Ubicación del Menú de Ayudas en la Ventana de Simulación del Turbo Codec.

El menú de ayudas del Turbo Codec fue creado con la finalidad de sintetizar toda la información que se tiene en sencillos diagramas de bloque, para de esta manera poder cumplir con el cometido de crear un software que sea didáctico y con un interfaz gráfico de usuario sencillo de utilizar, amigable y altamente intuitivo. Las ayudas están organizadas como se muestra en la Figura 3.96:

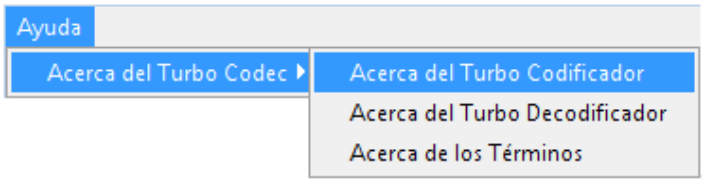
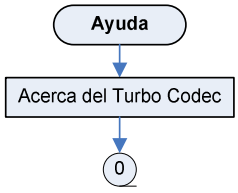


Figura 3.96. Menú de Ayudas del Programa.

La Figura 3.97 explica el funcionamiento del Menú de Ayudas del Turbo Codec:



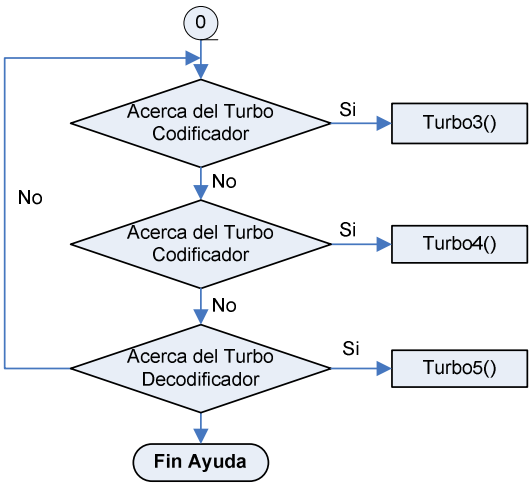


Figura 3.97. Diagrama de Flujo del Menú de Ayudas.

3.3.5.1 Ventana de Ayuda Acerca del Turbo Codificador

3.3.5.1.1 Descripción

La ventana de ayuda *Acerca del Turbo Codificador* muestra el diagrama de bloques implementado para la Codificación Turbo.

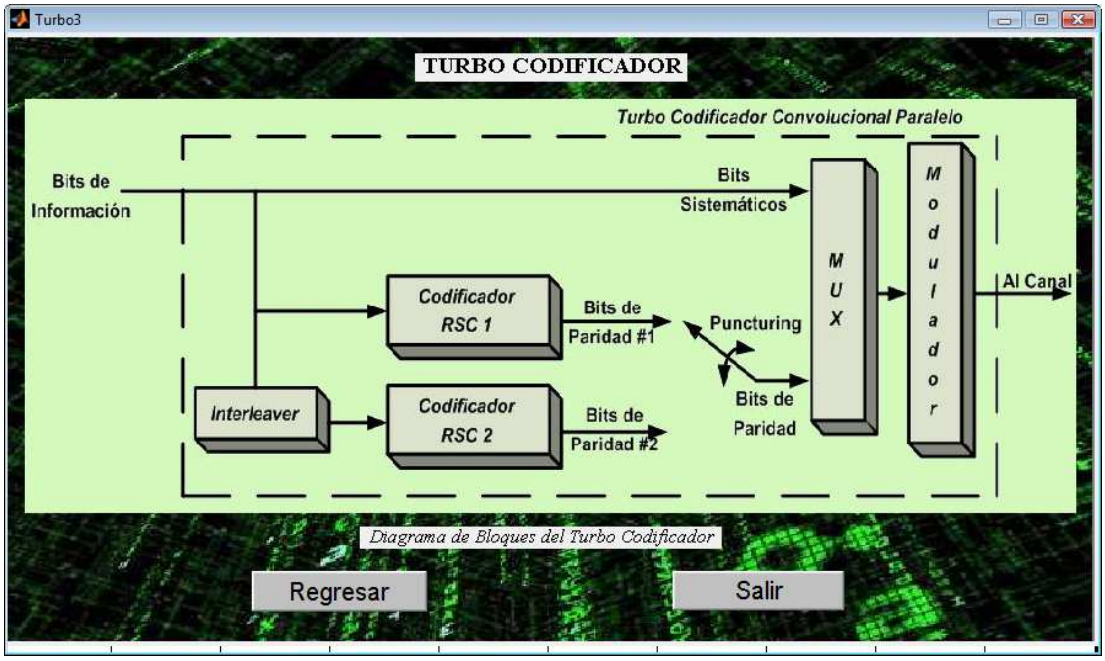


Figura 3.98. Ventana de Ayuda del Turbo Codificador.

3.3.5.1.2 Diagrama de Flujo

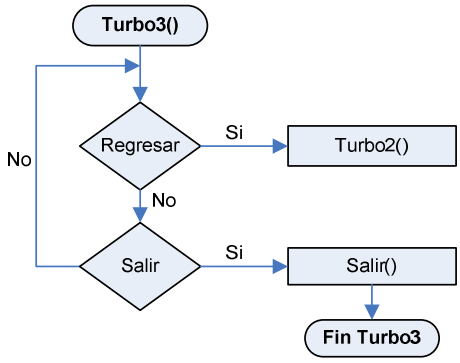


Figura 3.99. Diagrama de Flujo de la Ventana de Ayuda del Turbo Codificador.

3.3.5.2 Ventana de Ayuda Acerca del Turbo Decodificador

3.3.5.2.1 Descripción

La ventana de ayuda *Acerca del Turbo Decodificador* muestra el diagrama de bloques implementado para la Decodificación Turbo.

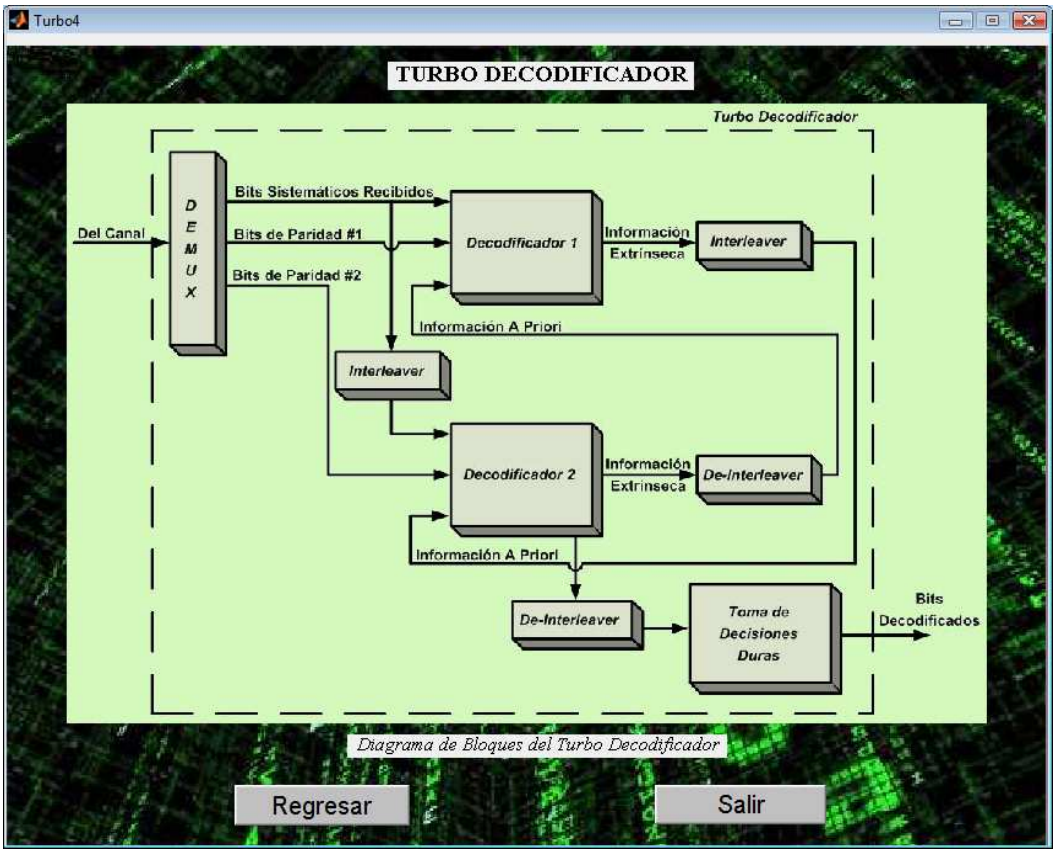


Figura 3.100. Ventana de Ayuda del Turbo Decodificador.

3.3.5.2.2 Diagrama de Flujo

La Figura 3.101 a continuación mostrada, presenta el diagrama de flujo de la ventana *Ayuda del Turbo Decodificador*:

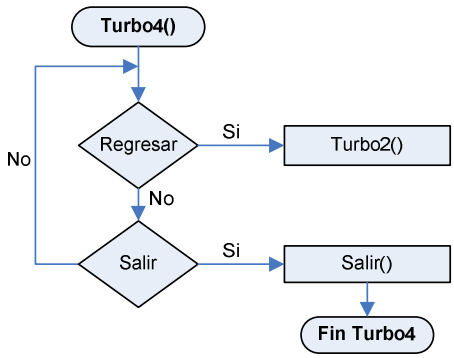


Figura 3.101. Diagrama de Flujo de la Ventana de Ayuda del Turbo Decodificador.

3.3.5.3 Ventana de Ayuda Acerca de los Términos

3.3.5.3.1 Descripción

La ventana de ayuda *Acerca de los Términos* muestra el significado de una serie de palabras y términos que se han empleado tanto en el proceso de codificación como en el de decodificación, para esclarecer y mostrar de manera didáctica las expresiones más utilizadas en este esquema de codificación.

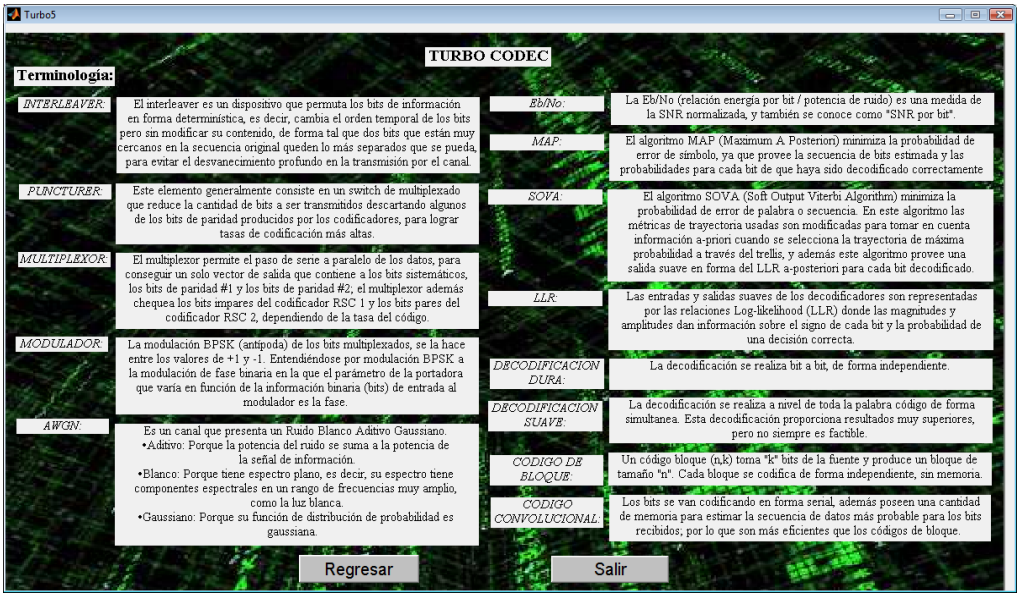


Figura 3.102. Ventana de Ayuda Acerca de los Términos Usados en el Turbo Codec.

3.3.5.3.2 Diagrama de Flujo

El diagrama de flujo de la ventana *Ayuda Acerca de los Términos*, se muestra en la Figura 3.103.

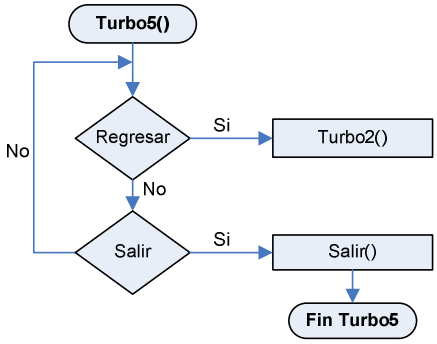


Figura 3.103. Diagrama de Flujo de la Ventana de Ayuda Acerca de los Términos Usados.

3.4 SIMULACIÓN, PRUEBAS DEL TURBO CODEC Y ANÁLISIS DE RESULTADOS

Para poder apreciar el desempeño ofrecido por el Turbo Código, en esta sección del presente proyecto de titulación, la cantidad de información, el código generador, la tasa de código, número de iteraciones empleadas en el proceso de decodificación y algoritmo de decodificación SISO utilizado es investigado a través de la simulación, y los resultados son analizados y presentados en tablas para sintetizar la información obtenida.

De esta manera se presenta a continuación un conjunto de gráficas de BER vs. E_b/N_o obtenidas, luego de simular el Turbo Código implementado.

Puesto que la E_b/N_o (en dB) es un parámetro ingresado por el usuario en forma de número entero, para el propósito de graficación se ha hecho variar este parámetro para cada una de las iteraciones, en intervalos de 0.5 desde 0 hasta el valor ingresado por teclado, así se puede tener un gráfico consistente que permita observar el rendimiento del Turbo Código.

3.4.1 VARIACIÓN DEL TAMAÑO DE LA TRAMA Y PERFORACIÓN DEL CÓDIGO

A continuación, se muestra el desempeño del Turbo Código en función del tamaño del bloque de información. Para estas simulaciones, se utilizaron 2 iteraciones en el proceso de decodificado con un algoritmo de decodificación SISO MAP, tasa de código de 1/3, código generador por default [1 1 1, 1 0 1], y la E_b/N_o es de 2 dB.

El tamaño de la trama está estrictamente ligado con el tamaño de los bits con interleaving, es así que para las pruebas se han usado tamaños de trama de: 256, 1024, 4096 y 16384 bits (incluyendo los bits de cola), con la finalidad de verificar el rendimiento cuando menos correlacionada está la información. Debido a que el sistema diseñado es capaz de operar con dos tasas de código, la básica que es la que no utiliza puncturing 1/3 y la tasa de 1/2 que se utiliza cuando hay puncturing; en primera instancia se utilizará el sistema más básico.

Los detalles de los parámetros utilizados para la simulación se pueden observar en la Tabla 3.7:

Parámetro	**Valor**
Perforación de Código	Unpunctured
Código Generador	[1 1 1, 1 0 1]
E_b/N_o (en dB)	2
Número de Iteraciones por Trama	2
Algoritmo de Decodificación	MAP

Tabla 3.7. Parámetros de Simulación con Variación del Tamaño de la Trama y Unpuncturing.

La Figura 3.104 muestra el conjunto de curvas de rendimiento del Turbo Codec cuando se ha variado el tamaño de la trama y utilizado una tasa de código de 1/3.

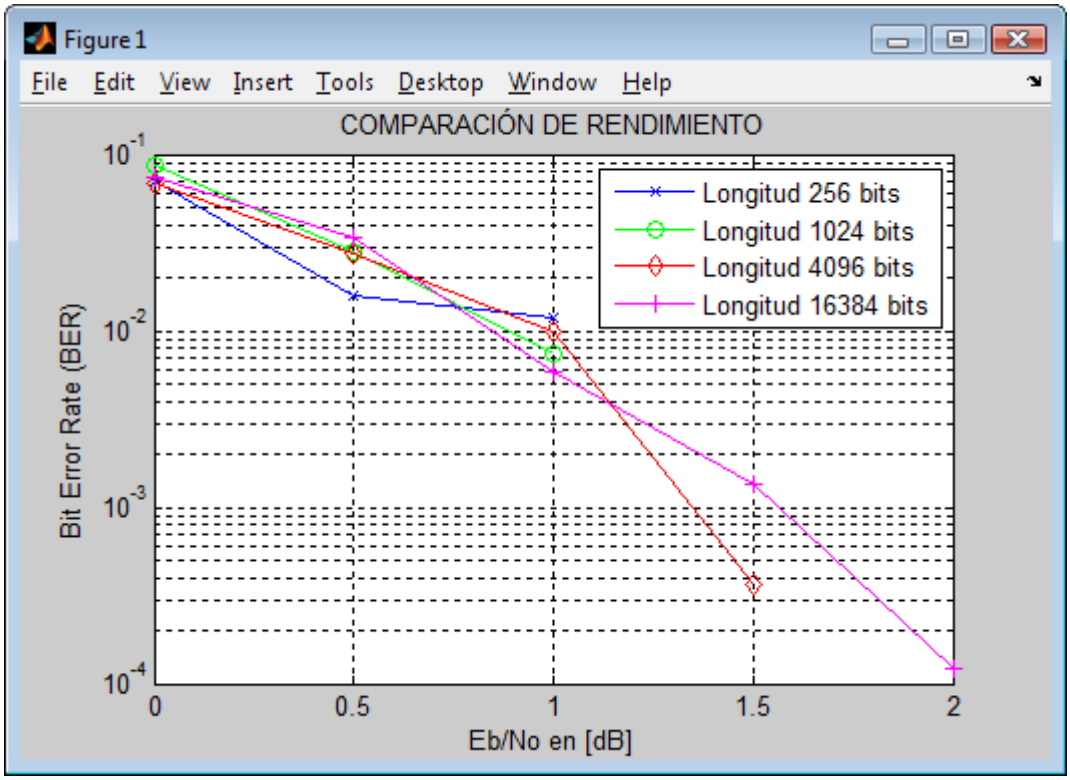


Figura 3.104. Simulación del Turbo Codec con Variación del Tamaño de la Trama y Unpuncturing.

La Tabla 3.8 sintetiza la información obtenida como resultado del proceso de simulación del Turbo Codec con variación de la trama y unpuncturing.

Tamaño de la Trama	**BER (Iteración 1)**				
	0 dB	**0.5 dB**	**1 dB**	**1.5 dB**	**2 dB**
256	7.4803e-002	2.1654e-002	3.7402e-002	9.8425e-003	1.5748e-002
1024	8.5616e-002	4.7456e-002	2.5440e-002	1.4188e-002	7.8278e-003
4096	6.4362e-002	4.6287e-002	3.2853e-002	1.1358e-002	7.9384e-003
16384	7.2854e-002	5.0177e-002	2.8629e-002	1.7916e-002	6.0737e-003
Tamaño de la Trama	**BER (Iteración 2)**				
	0 dB	**0.5 dB**	**1 dB**	**1.5 dB**	**2 dB**
256	7.0866e-002	1.5748e-002	1.1811e-002	0.0000e+000	0.0000e+000
1024	8.6595e-002	2.7886e-002	7.3386e-003	0.0000e+000	0.0000e+000
4096	6.8149e-002	2.7113e-002	9.7704e-003	3.6639e-004	0.0000e+000
16384	7.4564e-002	3.3787e-002	5.8601e-003	1.3429e-003	1.2209e-004

Tabla 3.8. Resultados de la Simulación del Turbo Codec con Variación del Tamaño de la Trama y Unpuncturing.

Como se aprecia, el desempeño del Turbo Código mejora conforme el tamaño del bloque de información *u* se incrementa, esto se debe a que mientras más grande es la cantidad de bits enviados, el tamaño de los bits con interleaving también se incrementan, permitiendo a la información estar menos correlacionada y conseguirse por consiguiente un BER más bajo. No obstante, en general, longitudes de entrada grandes implican un mayor retardo en la decodificación y consume mayor cantidad de recursos del ordenador, el cual puede ser en ocasiones intolerable en aplicaciones en tiempo real (voz, vídeo, etc) y además aumenta la probabilidad de que un bit corrompa la secuencia completa, aumentando así el BER.

Es importante notar, el gran desempeño que el Turbo Código presenta cuando *u* = 16384 bits y se utilizan 2 iteraciones en el proceso de decodificado, aún cuando no es el mejor BER alcanzado. Para una BER = 10^{-4}, se requiere una E_b/N_o de tan solo 2 dB, lo que se traduce en un desempeño que está a 2.5 dB de distancia (aproximadamente) del límite de Shannon, para códigos de tasa *r* = 1/3. Sin embargo, tal desempeño, requerirá de más memoria y un mayor tiempo de decodificado por parte del Turbo Decodificador, por el tamaño del bloque de información. Aunque como se puede observar en la Figura 3.104, para tramas con longitud media (*u* = 1024 bits = menor tiempo de decodificación), el Turbo Código es capaz de proveer una BER = 10^{-3} empleando una SNR por bit de información pequeña.

A continuación se ilustra el desempeño ofrecido por el Turbo Código utilizando varios valores de bits de información en la trama y el efecto que produce variar su tasa de código *R* a una tasa de 1/2, que se utiliza cuando hay puncturing.

Tal desempeño y el efecto que produce en éste la variación de la tasa de código, se muestra en la Figura 3.105, que consiste en un conglomerado de gráficas de BER vs E_b/N_o, luego de simular el Turbo Código para un canal AWGN, empleando modulación BPSK, con 2 iteraciones en el proceso de decodificado, algoritmo de decodificación MAP, código generador [1 1 1, 1 0 1] y E_b/N_o de 2 dB.

Los detalles de los parámetros utilizados para la simulación se pueden observar en la Tabla 3.9:

Parámetro	Valor
Perforación de Código	Punctured
Código Generador	[1 1 1, 1 0 1]
E_b/N_o (en dB)	2
Número de Iteraciones por Trama	2
Algoritmo de Decodificación	MAP

Tabla 3.9. Parámetros de Simulación con Variación del Tamaño de la Trama y Puncturing.

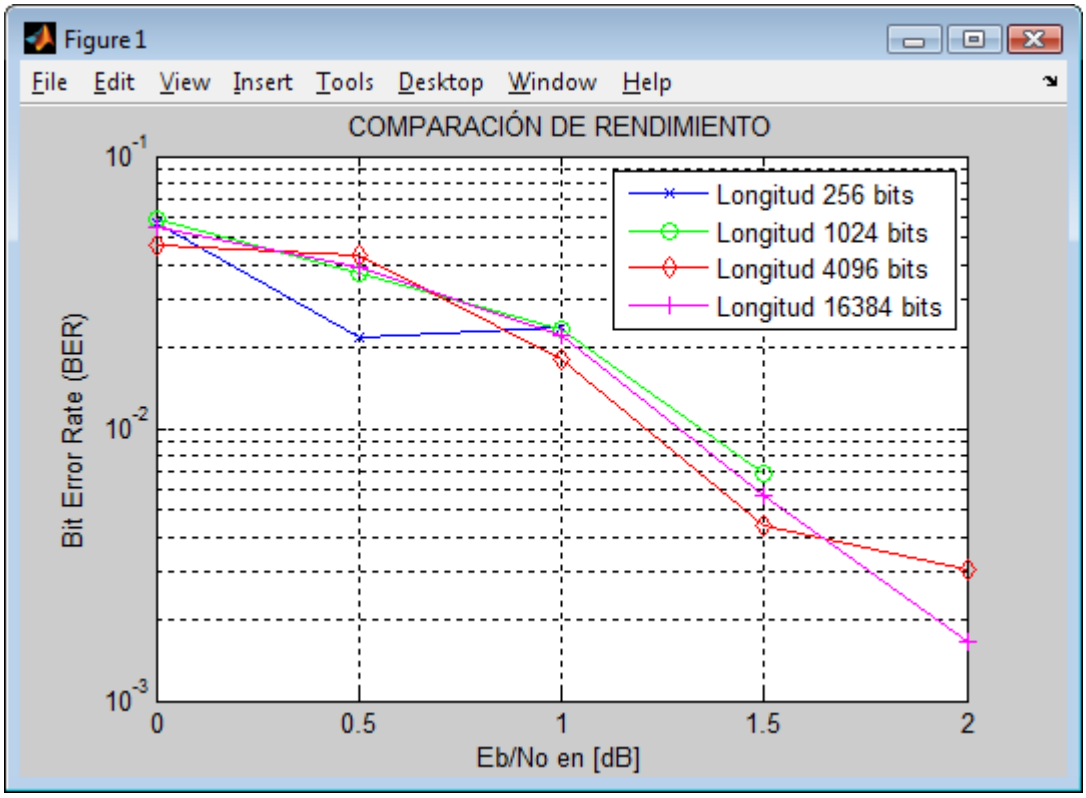


Figura 3.105. Simulación del Turbo Codec con Variación del Tamaño de la Trama y Puncturing.

La Tabla 3.10 sintetiza la información obtenida como resultado del proceso de simulación del Turbo Codec teniendo como variable el tamaño de la trama y operando a una tasa de código de 1/2 que es la que utiliza puncturing.

Tamaño de la Trama	BER (Iteración 1)				
	0 dB	0.5 dB	1 dB	1.5 dB	2 dB
256	5.9055e-002	2.7559e-002	2.3622e-002	2.1654e-002	5.9055e-003
1024	6.0665e-002	4.3053e-002	4.1096e-002	2.0548e-002	1.6145e-002
4096	5.5813e-002	5.0928e-002	3.3097e-002	1.8319e-002	1.7953e-002
16384	5.9517e-002	4.7644e-002	3.6107e-002	1.9930e-002	1.1629e-002
Tamaño de la Trama	**BER (Iteración 2)**				
	0 dB	**0.5 dB**	**1 dB**	**1.5 dB**	**2 dB**
256	5.7087e-002	2.1654e-002	2.3622e-002	0.0000e+000	0.0000e+000
1024	5.8708e-002	3.7182e-002	2.2994e-002	6.8493e-003	0.0000e+000
4096	4.6898e-002	4.3478e-002	1.8075e-002	4.3967e-003	3.0532e-003
16384	5.4969e-002	3.9098e-002	2.2067e-002	5.6464e-003	1.6482e-003

Tabla 3.10. Resultados de Simulación del Turbo Codec con Variación del Tamaño de la Trama y Puncturing.

Se puede apreciar que el desempeño del Turbo Código mejora notablemente conforme su tasa de código se decrementa. En general, cualquier esquema para control de errores FEC presenta un efecto similar en su desempeño; esto se debe a que mientras más baja sea la tasa de un código, más bits de paridad son enviados para proteger la información, sin la necesidad de utilizar bits de relleno.

El desempeño ofrecido por el Turbo Código, de tasa r = 1/3 y 1/2, se ve afectado por el diseño del interleaver, que produce una variación notable entre ambas tasas de código, para nuestro caso este parámetro no es tomado en cuenta ya que, para las dos opciones se usa un interleaver pseudo aleatorio; esto permite afirmar que tanto el interleaver de tasa 1/3, como el interleaver de tasa 1/2, ofrecen un desempeño similar. Sin embargo, si se utiliza un diseño diferente de interleaver para cada tasa de código, es importante tener en cuenta los efectos que este produce.

3.4.2 VARIACIÓN DEL CÓDIGO GENERADOR

Los detalles de los parámetros utilizados para la simulación se pueden observar en la Tabla 3.11 y sus resultados en la Figura 3.106:

Parámetro	Valor
Tamaño de la Trama	500
Perforación de Código	Unpunctured
E_b/N_o (en dB)	2
Número de Iteraciones por Trama	2
Algoritmo de Decodificación	MAP

Tabla 3.11. Parámetros de Simulación con Variación del Código Generador.

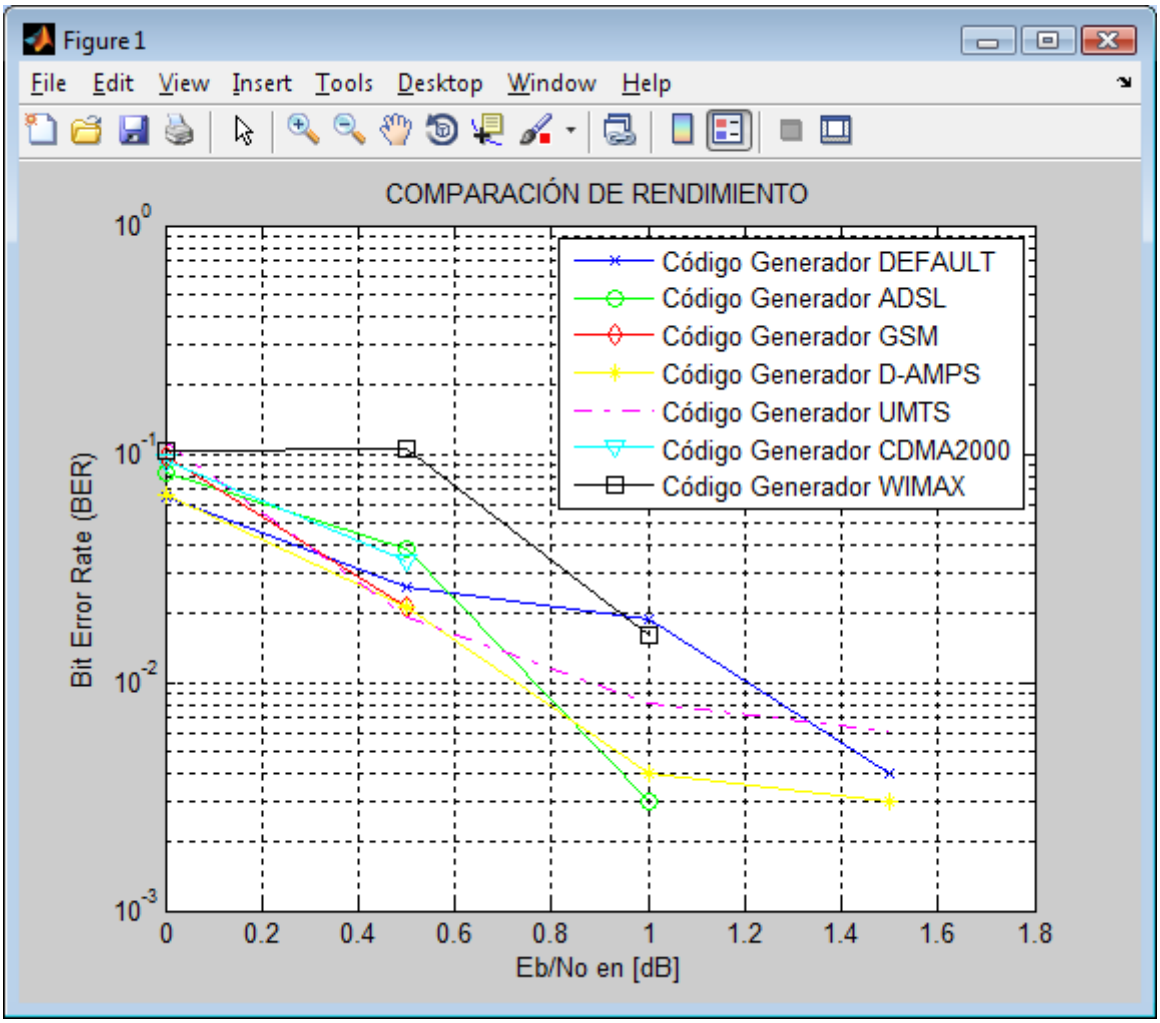


Figura 3.106. Simulación del Turbo Codec con Variación del Código Generador.

La Tabla 3.12 sintetiza la información obtenida como resultado del proceso de simulación del Turbo Codec teniendo como variable el código generador.

Código Generador	BER (Iteración 1)				
	0 dB	**0.5 dB**	**1 dB**	**1.5 dB**	**2 dB**
DEFAULT	5.9237e-002	4.7189e-002	4.2169e-002	2.2088e-002	1.3052e-002
ADSL	7.2435e-002	6.0362e-002	3.8229e-002	8.0483e-003	0.0000e+000
GSM	8.5685e-002	3.1250e-002	1.8145e-002	0.0000e+000	3.0242e-003
D-AMPS	6.0362e-002	3.9235e-002	3.4205e-002	2.5151e-002	3.0181e-003
UMTS	9.8592e-002	3.9235e-002	3.2193e-002	1.7103e-002	0.0000e+000
CDMA2000	8.0483e-002	4.2254e-002	2.4145e-002	2.3139e-002	9.0543e-003
WIMAX	1.0223e-001	9.7166e-002	5.9717e-002	2.5304e-002	0.0000e+000
Código Generador	**BER (Iteración 2)**				
	0 dB	**0.5 dB**	**1 dB**	**1.5 dB**	**2 dB**
DEFAULT	6.5261e-002	2.6104e-002	1.9076e-002	4.0161e-003	0.0000e+000
ADSL	8.2495e-002	3.8229e-002	3.0181e-003	0.0000e+000	0.0000e+000
GSM	9.6774e-002	2.1169e-002	0.000e+000	0.0000e+000	0.0000e+000
D-AMPS	6.6398e-002	2.1127e-002	4.0241e-003	3.0181e-003	0.0000e+000
UMTS	1.1066e-001	1.9115e-002	8.0483e-003	6.0362e-003	0.0000e+000
CDMA2000	9.3561e-002	3.4205e-002	0.000e+000	0.0000e+000	0.0000e+000
WIMAX	1.0324e-001	1.0526e-001	1.6194e-002	0.0000e+000	0.0000e+000

Tabla 3.12. Resultados de Simulación del Turbo Codec con Variación del Código Generador.

Como resultado de esta simulación se observa que el código generador correspondiente a ADSL presenta mejor rendimiento bajo las condiciones mencionadas en la Tabla 3.11. Esto no implica, bajo ninguna circunstancia, que una transmisión de datos en ADSL sea más eficiente que una transmisión de datos en Wi-Max, por ejemplo. Sino, que para fines didácticos se han considerado los códigos generadores de diferentes sistemas de comunicación, para no tomar polinomios generadores al azar. Una comparación real de sistemas de comunicación, debería incluir otros parámetros como son: longitud de trama, modulación, canal de comunicaciones, etc.

3.4.3 VARIACIÓN DE LA E_b/N_o

El desempeño del Turbo Código cuando se varía la energía de bit sobre potencia de ruido E_b/N_o, se presenta a través de la Figura 3.107. Para estas simulaciones,

se utilizaron 2 iteraciones en el proceso de decodificado, un algoritmo de decodificación SISO MAP, tasa de código de 1/3 y código generador [1 1 1, 1 0 1]. Los detalles de los parámetros utilizados para la simulación se pueden observar en la Tabla 3.13:

Parámetro	**Valor**
Tamaño de la Trama	500
Perforación de Código	Unpunctured
Código Generador	[1 1 1, 1 0 1]
Número de Iteraciones por Trama	2
Algoritmo de Decodificación	MAP

Tabla 3.13. Parámetros de Simulación con Variación de la E_b/N_o.

Se debe tener en cuenta que los estándares determinan que para tener un rango aceptable de rendimiento en un Turbo Código, se debe tener una E_b/N_o entre 1.5 y 4 dB, por lo que los gráficos que permiten medir el BER, han sido elaborados únicamente en este rango con saltos de 0.5dB, con el propósito de evaluar cuáles son los efectos que producen el variar la E_b/N_o.

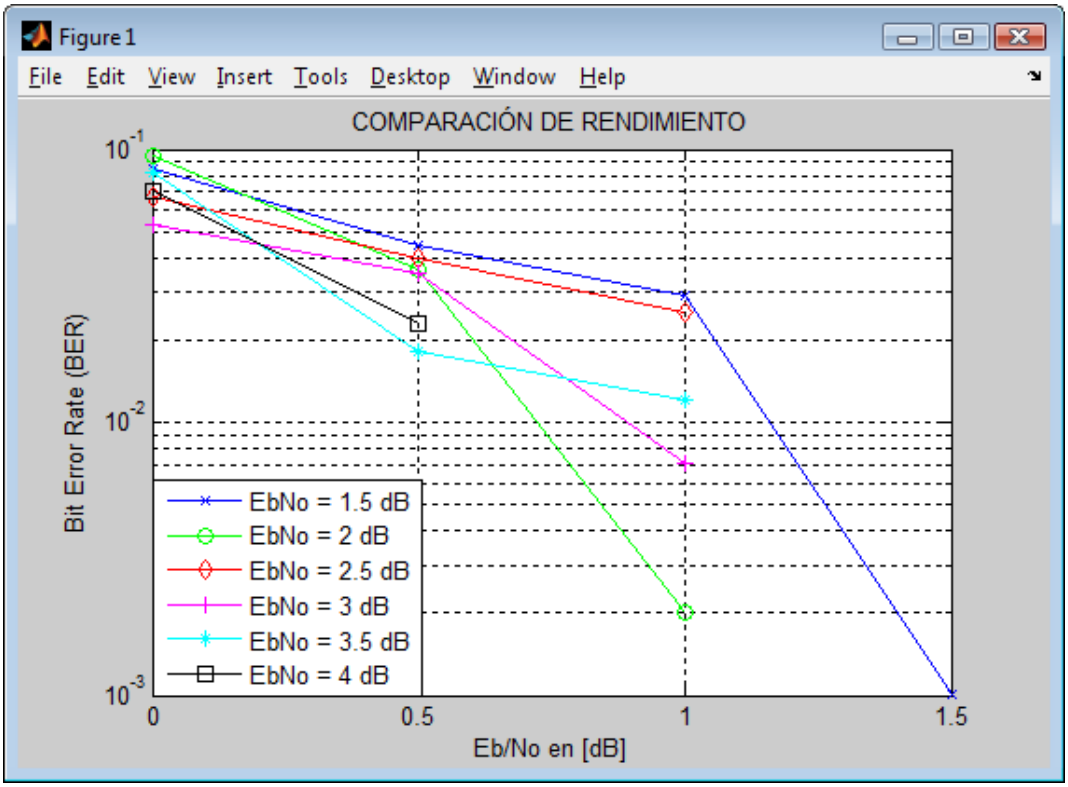


Figura 3.107. Simulación del Turbo Codec con Variación de la Eb/No.

La Tabla 3.14 sintetiza la información obtenida como resultado del proceso de simulación del Turbo Codec teniendo como variable E_b/N_o.

E_b/N_o (en dB)	**BER (Iteración 1)**				
	0 dB	**0.5 dB**	**1 dB**	**1.5 dB**	**2 dB**
1.5	6.5261e-002	4.0161e-002	2.8112e-002	6.0241e-003	---------
	BER (Iteración 2)				
	0 dB	**0.5 dB**	**1 dB**	**1.5 dB**	**2 dB**
	7.5301e-002	3.3133e-002	5.1044e-002	0.000e+000	---------
E_b/N_o (en dB)	**BER (Iteración 1)**				
	0 dB	**0.5 dB**	**1 dB**	**1.5 dB**	**2 dB**
2	9.2369e-002	5.3213e-002	1.9076e-002	1.8072e-002	7.0281e-003
	BER (Iteración 2)				
	0 dB	**0.5 dB**	**1 dB**	**1.5 dB**	**2 dB**
	9.4378e-002	3.6145e-002	2.0080e-003	0.000e+000	0.0000e+000
E_b/N_o (en dB)	**BER (Iteración 1)**				
	0 dB	**0.5 dB**	**1 dB**	**1.5 dB**	**2 dB**
	6.3253e-002	5.7229e-002	5.2209e-002	2.2088e-002	5.0201e-003
	BER (Iteración 1)				
	2.5 dB	**3 dB**	**3.5 dB**	**4 dB**	**4.5 dB**
2.5	0.000e+000	---------	---------	---------	---------
	BER (Iteración 2)				
	0 dB	**0.5 dB**	**1 dB**	**1.5 dB**	**2 dB**
	6.7269e-002	4.0161e-002	2.5100e-002	0.000e+000	0.0000e+000
	BER (Iteración 2)				
	2.5 dB	**3 dB**	**3.5 dB**	**4 dB**	**4.5 dB**
	0.000e+000	---------	---------	---------	---------
E_b/N_o (en dB)	**BER (Iteración 1)**				
	0 dB	**0.5 dB**	**1 dB**	**1.5 dB**	**2 dB**
	6.1245e-002	4.5181e-002	2.3092e-002	1.5060e-002	4.0161e-003
3	**BER (Iteración 1)**				
	2.5 dB	**3 dB**	**3.5 dB**	**4 dB**	**4.5 dB**
	3.0120e-003	0.0000e+000	---------	---------	---------

	BER (Iteración 2)				
	0 dB	**0.5 dB**	**1 dB**	**1.5 dB**	**2 dB**
	5.3213e-002	3.5141e-002	7.0281e-003	0.000e+000	0.0000e+000
	BER (Iteración 2)				
	2.5 dB	**3 dB**	**3.5 dB**	**4 dB**	**4.5 dB**
	0.000e+000	0.0000e+000	---------	---------	---------
E_b/N_o (en dB) **3.5**	**BER (Iteración 1)**				
	0 dB	**0.5 dB**	**1 dB**	**1.5 dB**	**2 dB**
	7.3293e-002	3.8153e-002	4.7189e-002	1.8072e-002	5.0201e-003
	BER (Iteración 1)				
	2.5 dB	**3 dB**	**3.5 dB**	**4 dB**	**4.5 dB**
	0.000e+000	2.0080e-003	0.0000e+000	---------	---------
	BER (Iteración 2)				
	0 dB	**0.5 dB**	**1 dB**	**1.5 dB**	**2 dB**
	8.2329e-002	1.8072e-002	1.2048e-002	0.000e+000	0.0000e+000
	BER (Iteración 2)				
	2.5 dB	**3 dB**	**3.5 dB**	**4 dB**	**4.5 dB**
	0.000e+000	0.0000e+000	0.000e+000	---------	---------
E_b/N_o (en dB) **4**	**BER (Iteración 1)**				
	0 dB	**0.5 dB**	**1 dB**	**1.5 dB**	**2 dB**
	8.0321e-002	3.6145e-002	1.0040e-002	1.7068e-002	4.0161e-003
	BER (Iteración 1)				
	2.5 dB	**3 dB**	**3.5 dB**	**4 dB**	**4.5 dB**
	6.0241e-003	0.0000e+000	0.000e+000	0.000e+000	---------
	BER (Iteración 2)				
	0 dB	**0.5 dB**	**1 dB**	**1.5 dB**	**2 dB**
	7.0281e-002	1.1035e-002	0.000e+000	0.000e+000	0.0000e+000
	BER (Iteración 2)				
	2.5 dB	**3 dB**	**3.5 dB**	**4 dB**	**4.5 dB**
	0.000e+000	0.0000e+000	0.000e+000	0.000e+000	---------

Tabla 3.14. Resultados de Simulación del Turbo Codec con Variación de la E_b/N_o.

Los resultados denotan un decremento en la tasa de bits errados (BER) conforme la E_b/N_o aumenta. El tiempo de latencia que demanda tener una E_b/N_o alta se dispara dramáticamente, demostrándose así que una relación E_b/N_o grande permite tener una buena comunicación de datos acercándose al límite de Shannon, pero consume una gran cantidad de recursos computacionales, lo que limita el sistema de comunicación.

3.4.4 VARIACIÓN DEL NÚMERO DE ITERACIONES

Los detalles de los parámetros utilizados para la simulación se pueden observar en la Tabla 3.15:

Parámetro	Valor
Tamaño de la Trama	500
Perforación de Código	Unpunctured
Código Generador	[1 1 1, 1 0 1]
E_b/N_o (en dB)	2
Algoritmo de Decodificación	MAP

Tabla 3.15. Parámetros de Simulación con Variación del Número de Iteraciones.

Las Figura 3.108 muestra el efecto que produce en la curva de desempeño del Turbo Código, el variar el número de iteraciones utilizado en el proceso de Turbo Decodificación, empleando el algoritmo de decodificación SISO MAP, y un tamaño de bloque de información u = 500 bits. Se ha transmitido la señal con una E_b/N_o de 2 dB y no se ha perforado el código, por lo que su tasa de código es de 1/3.

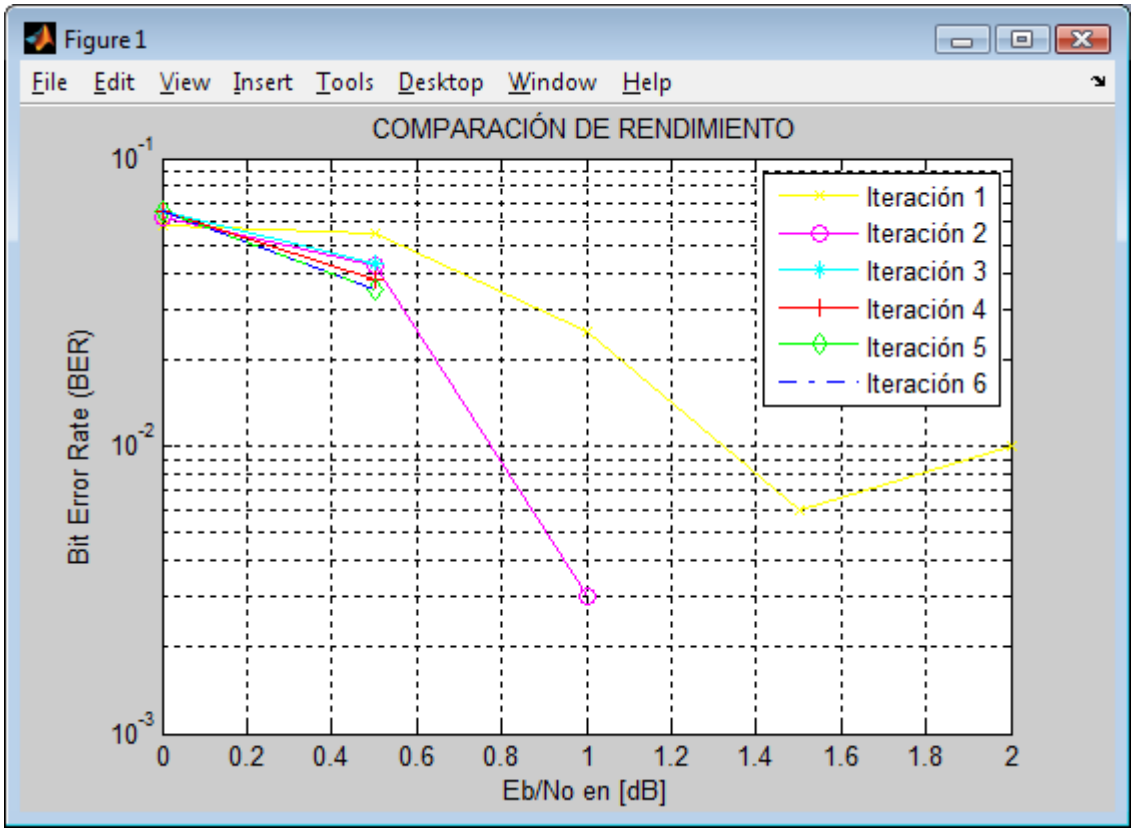


Figura 3.108. Simulación del Turbo Codec con Variación de Número de Iteraciones

La Tabla 3.16 sintetiza la información obtenida como resultado del proceso de simulación del Turbo Codec teniendo como variable el número de iteraciones en el proceso de decodificación.

Número de Iteraciones	**BER**				
	0 dB	**0.5 dB**	**1 dB**	**1.5 dB**	**2 dB**
1	8.0321e-002	6.1245e-002	4.1165e-002	2.4096e-002	9.0361e-003
2	8.5341e-002	5.6225e-002	1.0040e-003	0.0000e+000	0.0000e+000
Número de Iteraciones	**BER**				
	0 dB	**0.5 dB**	**1 dB**	**1.5 dB**	**2 dB**
1	8.2329e-002	5.9237e-002	2.9116e-002	2.7108e-002	0.0000e+000
2	8.0321e-002	4.4177e-002	5.0201e-003	0.0000e+000	0.0000e+000
3	8.1325e-002	3.4137e-002	0.000e+000	0.0000e+000	0.0000e+000
4	8.1325e-002	2.9116e-002	0.000e+000	0.0000e+000	0.0000e+000

Número de Iteraciones	BER				
	0 dB	**0.5 dB**	**1 dB**	**1.5 dB**	**2 dB**
1	5.8233e-002	5.5221e-002	2.5100e-002	6.0241e-003	1.0040e-002
2	6.2249e-002	4.2169e-002	3.0120e-003	0.0000e+000	0.0000e+000
3	6.5261e-002	4.3173e-002	0.000e+000	0.0000e+000	0.0000e+000
4	6.5261e-002	3.8153e-002	0.000e+000	0.0000e+000	0.0000e+000
5	6.5261e-002	3.5141e-002	0.000e+000	0.0000e+000	0.0000e+000
6	6.5261e-002	3.5141e-002	0.000e+000	0.0000e+000	0.0000e+000

Tabla 3.16. Resultados de Simulación del Turbo Codec con Variación del Número de Iteraciones.

Se aprecia como el desempeño del Turbo Código mejora conforme se incrementa el número de iteraciones en el Turbo Decodificador, de forma tal que después de 6 iteraciones, el sistema es capaz de alcanzar una BER = 0 a una E_b/N_o de 1 dB.

3.4.5 VARIACIÓN DEL ALGORITMO DE DECODIFICACIÓN

Finalmente se muestra el desempeño del Turbo Código en función del algoritmo de decodificación SISO utilizado por el Turbo Decodificador. Para estas simulaciones, se utilizaron 2 iteraciones en el proceso de decodificado, y un tamaño del bloque de información u = 500 bits, con tasa 1/3.

Parámetro	**Valor**
Tamaño de la Trama	500
Perforación de Código	Unpunctured
Código Generador	[1 1 1, 1 0 1]
E_b/N_o (en dB)	2
Número de Iteraciones por Trama	2

Tabla 3.17. Parámetros de Simulación con Variación del Algoritmo de Decodificación.

La Figura 3.109 resume el comportamiento de los dos algoritmos en un mismo gráfico.

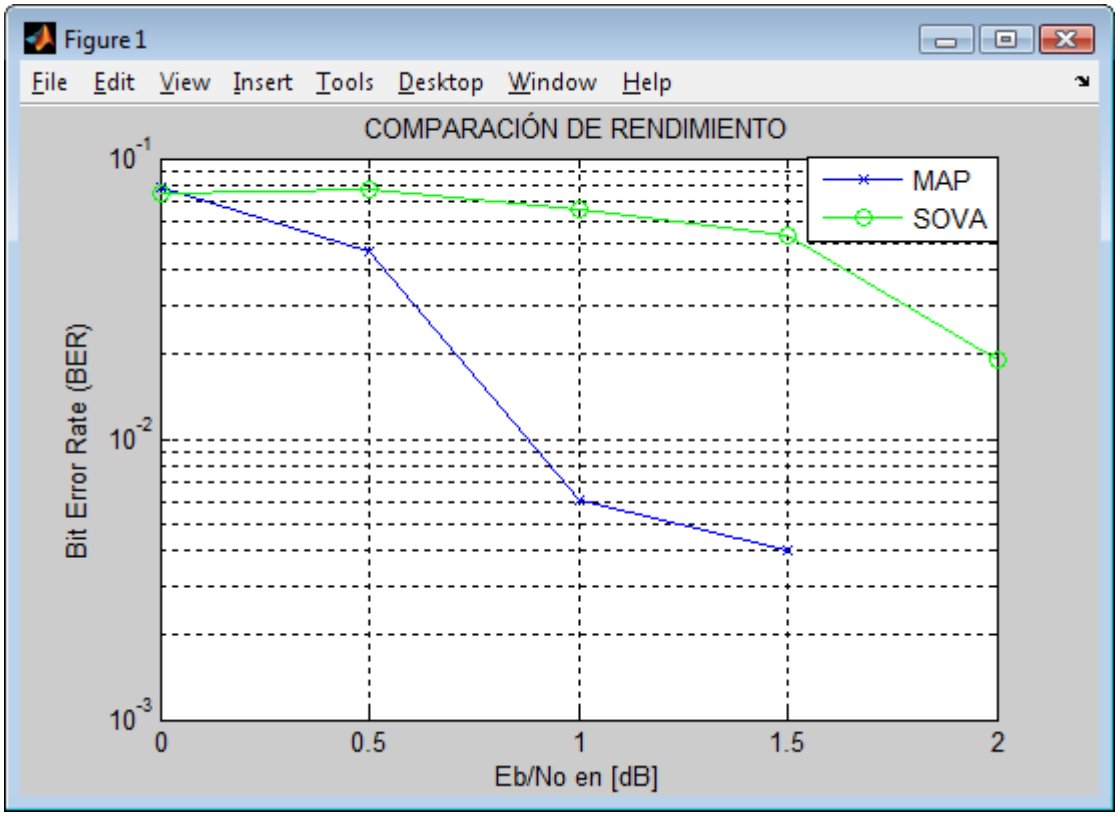


Figura 3.109. Simulación del Turbo Codec con Variación del Algoritmo de Decodificación

La Tabla 3.18 sintetiza la información obtenida como resultado del proceso de simulación del Turbo Codec teniendo como variable el algoritmo de decodificación.

Algoritmo de Decodificación	**BER (Iteración 1)**				
	0 dB	**0.5 dB**	**1 dB**	**1.5 dB**	**2 dB**
MAP	8.6345e-002	3.6145e-002	4.0161e-002	2.0080e-002	7.0281e-003
SOVA	7.2289e-002	7.3293e-002	6.1245e-002	5.0201e-002	1.4056e-002
Algoritmo de Decodificación	**BER (Iteración 2)**				
	0 dB	**0.5 dB**	**1 dB**	**1.5 dB**	**2 dB**
MAP	7.9321e-002	4.6183e-002	6.0244e-003	4.0160e-003	0.0000e+000
SOVA	7.5301e-002	7.7309e-002	6.5261e-002	5.3213e-002	1.9076e-002

Tabla 3.18. Resultados Simulación del Turbo Codec con Variación del Algoritmo de Decodificación

Los algoritmos de decodificación SISO que se utilizaron para estas simulaciones, son el algoritmo MAP y SOVA. Se debe notar como la utilización del algoritmo SOVA, degrada notablemente el desempeño del Turbo Código. Pero, el algoritmo

MAP demanda un mayor consumo de tiempo de procesamiento computacional, debido a la complejidad algorítmica de MAP. Es así, que con la finalidad de analizar el tiempo que ha consumido, cada uno de los algoritmos de decodificación se ha insertado un temporizador a través de la función ***tic toc*** de Matlab, y variando la potencia se han obtenido los resultados que se muestran en la Tabla 3.19.

Algoritmo de Decodificación	**Tiempo de Decodificado [Seg]**				
	0.5 dB	**1 dB**	**1.5 dB**	**2 dB**	**2.5 dB**
MAP	1.1732	1.7265	2.1868	2.6229	3.0443
SOVA	0.83315	1.1428	1.3621	1.5397	1.7984
Algoritmo de Decodificación	**Tiempo de Decodificado [Seg]**				
	3 dB	**3.5 dB**	**4 dB**	**4.5 dB**	**5 dB**
MAP	3.6223	4.0606	4.6285	5.0899	5.5171
SOVA	2.0762	2.3224	2.5171	2.6847	3.0733

Tabla 3.19. Resultados del Tiempo de Decodificado del Algoritmo MAP y SOVA.

3.5 ANÁLISIS COMPARATIVO DE RENDIMIENTO DE LOS TURBO CÓDIGOS

La Figura 3.110 permite visualizar a la secuencia de bits almacenada en el archivo de texto *x.txt* que se generó desde el Espacio de Código 1.1, cuando ha pasado a través de un canal AWGN sin codificar, un código de bloque, un código convolucional y un turbo código. Todo esto con la finalidad de observar la mejora en el rendimiento de los diversos códigos mediante una gráfica BER vs E_b/N_o. La Tabla 3.20 sintetiza la información obtenida como resultado del proceso de simulación de los diferentes esquemas FEC.

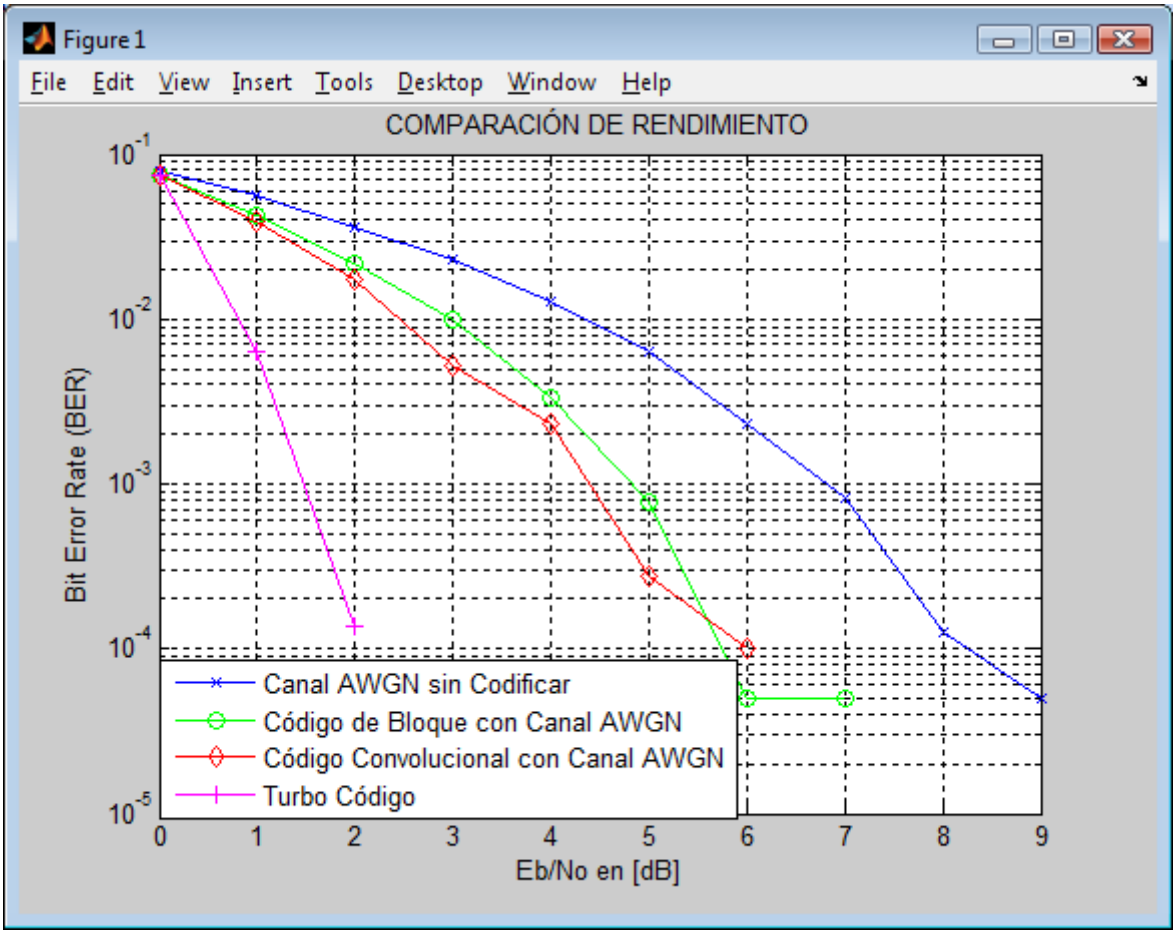


Figura 3.110. Comparación de Rendimiento de un Canal AWGN sin Codificar, un Código de Bloque, un Código Convolucional y un Turbo Código sobre un canal AWGN.

Eb/No (en dB)	**0**	**1**	**2**	**3**	**4**	**5**
BER (Sin Codificar)	0.079079	0.056903	0.036152	0.023051	0.012876	0.0063253
BER (con Código de Bloque)	0.073529	0.042927	0.021651	0.0099255	0.0033002	0.00077504
BER (con codigo Convolucional)	0.075179	0.039077	0.017601	0.0051753	0.0023251	0.00027501
BER (Turbo Código)	7.4991e-002	6.4128e-003	1.3751e-004	0	0	0
Eb/No (en dB)	**6**	**7**	**8**	**9**	**10**	**11**
BER (Sin Codificar)	0.0023001	0.00082504	0.00012501	5.0003e-005	0	0
BER (con Código de Bloque)	5.0003e-005	5.0003e-005	0	0	0	0

BER (Convolucional)	0.00010001	0	0	0	0	0
BER (Turbo Código)	0	0	0	0	0	0

Tabla 3.20. Resultados de Comparación de Rendimiento de un Canal AWGN sin Codificar, un Código de Bloque y un Código Convolucional sobre canal AWGN.

Como se esperaba el Turbo Código posee el mejor rendimiento entre los demás esquemas analizados con la secuencia de entrada *u* = 40000 bits, siendo el esquema sin codificador de errores del canal AWGN el que peores resultados tiene, le sigue a este el código de bloque y luego el código convolucional.

Analizando la información derivada de las simulaciones del Turbo Codec en la Sección 4.1 del presente proyecto de titulación, se puede obtener los parámetros con lo que los Turbo Códigos obtienen su mayor rendimiento, en este caso la información se encuentra detallada en la Tabla 3.21:

Parámetro	**Valor**
Tamaño de la Trama	1024
Perforación de Código	Unpunctured
Código Generador	[1 1 1, 1 0 1]
Algoritmo de Decodificación	MAP
Número de Iteraciones por Trama	6
E_b/N_o (en dB)	4

Tabla 3.21. Parámetros de Simulación del Turbo Código con Mejor Rendimiento.

El resultado de la simulación del Turbo Codec con estos parámetros, da como resultado el gráfico de rendimiento de la Figura 3.111.

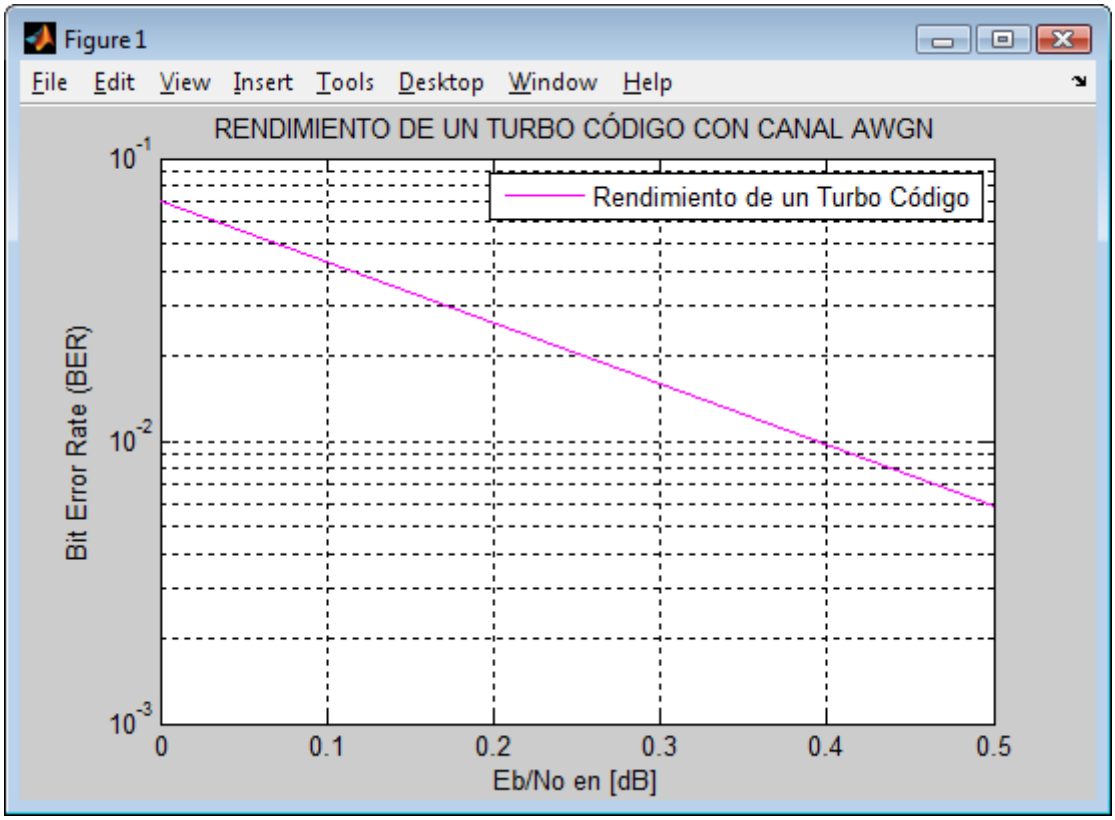


Figura 3.111. Simulación del Turbo Codec con Parámetros de Mejor Rendimento.

E_b/N_o (en dB)	**0**	**0.5**	**1**	**1.5**	**2**	**2.5**
BER (Turbo Código – Última Iteración)	7.9746e-002	4.0060e-002	0	0	0	0
E_b/N_o (en dB)	**3**	**3.5**	**4**	**4.5**	**5**	**5.5**
BER (Turbo Código – Última Iteración)	0	0	0	---------	---------	---------

Tabla 3.22. Resultados de Simulación de un Turbo Código con Parámetros de Mejor Rendimiento.

Con la finalidad de comparar las ventajas de los Turbo Códigos con los parámetros que arrojan el mejor rendimiento, con respecto a los otros esquemas de codificación FEC, se ha propuesto la simulación de la Figura 3.112, que tiene como parámetros de entrada los mostrados en la Tabla 3.21.

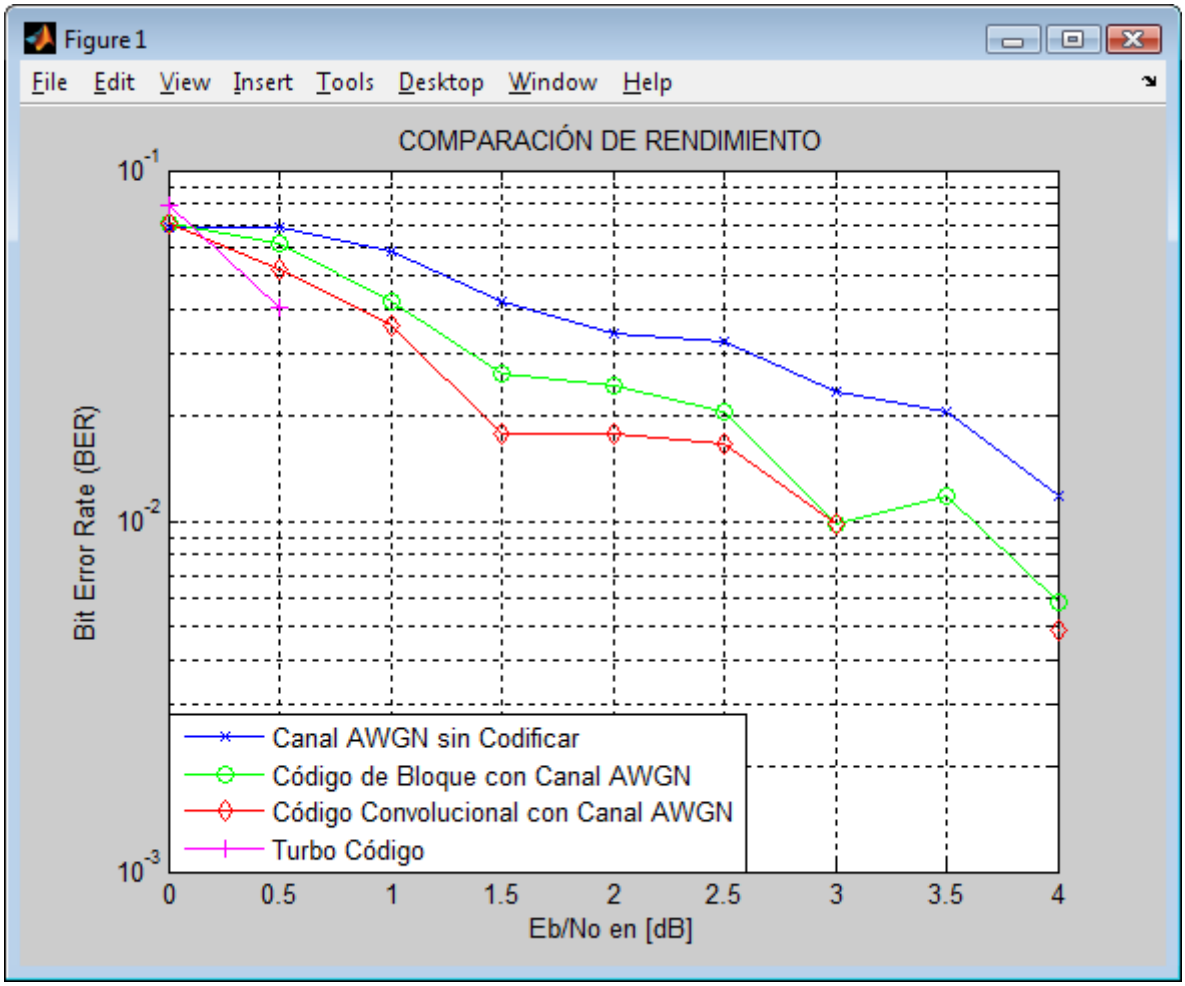


Figura 3.112. Comparación de Rendimiento de un Canal AWGN sin Codificar, Código de Bloque, Código Convolucional y Turbo Código con Parámetros de Mejor Rendimiento, sobre canal AWGN.

Se observa claramente como el Turbo Código para dichas condiciones reduce la tasa de bits errados de manera drástica, mejorando su rendimiento en comparación con los otros esquemas de codificación.

Eb/No (en dB)	**0**	**0.5**	**1**	**1.5**	**2**	**2.5**
BER (Sin Codificar)	0.068493	0.068493	0.058708	0.042074	0.034247	0.03229
BER (con Código de Bloque)	0.07045	0.061644	0.042074	0.026419	0.024462	0.020548
BER (con codigo Convolucional)	0.07045	0.051859	0.036204	0.017613	0.017613	0.016634
BER (Turbo Código)	7.9746e-002	4.0060e-002	0	0	0	0

Eb/No (en dB)	3	3.5	4	4.5	5	5.5
BER (Sin Codificar)	0.023483	0.020548	0.011742	---------	---------	---------
BER (con Código de Bloque)	0.0097847	0.011742	0.0058708	---------	---------	---------
BER (Convolucional)	0.0097847	0	0.0048924	---------	---------	---------
BER (Turbo Código)	0	0	0	---------	---------	---------

Tabla 3.23. Comparación de Rendimiento de un Canal AWGN sin Codificar, un Código de Bloque, un Código Convolucional y un Turbo Código con Parámetros de Mejor Rendimiento.

Con base a los resultados simulados en la Tabla 3.23, se ha decidido realizar un análisis matemático comparativo de parámetros de rendimiento, con el objeto de apreciar cuantitativamente que mejoras presta el uso de un Turbo Código con respecto al resto de esquemas de codificación FEC.

Para calcular el rendimiento y tomando como valor teórico al obtenido por el Turbo Código y valores experimentales a los códigos FEC, se puede calcular la eficiencia de los códigos. A continuación se muestra un ejemplo de cálculo:

$$\eta_{\%} = \frac{V_2 - V_1}{V_2} x100 \longrightarrow \eta_{\%} = \frac{0.068493 - 0.040060}{0.068493} \text{x}100 = 42\%$$

Esto significa que el Turbo Código es un 42% más eficiente que el canal AWGN sin codificar en términos de BER para una potencia (E_b/N_o) de 0.5 dB. La Tabla 3.24 muestra de manera simplificada los resultados de este análisis, para los valores de potencia en los que el Turbo Código presenta un valor distinto de cero.

Eb/No (en dB)	0	0.5
BER (Sin Codificar)	-14%	42%
BER (con Código de Bloque)	-13%	35%
BER (con codigo Convolucional)	-13%	23%

Tabla 3.24. Eficiencia de un Canal AWGN sin Codificar, un Código de Bloque y un Código Convolucional sobre canal AWGN, con respecto al Turbo Código.

Para el particular analizado se observa que, aunque inicialmente para una E_b/N_o de 0 dB el rendimiento del Turbo Código es menor que el de los otros esquemas FEC; al incrementarse la potencia de transmisión a 0.5 dB, este radicalmente supera a dichos esquemas, con un crecimiento de su rendimiento en 54% para el Canal AWGN sin Codificar, 48% para el Código de Bloque y 36% para el Código Convolucional, y en un próximo intervalo de 1 dB termina reduciéndose a cero su correspondiente BER. En términos generales esto da una idea de que los Turbo Códigos superan cualquier otra técnica de codificación que hoy conocemos. [180]Las tres áreas clave en que los Turbo Códigos proveen un mejoramiento son:

- *Capacidad:* los Turbo Códigos logran un desempeño cercano a los límites teóricos de la capacidad de canal de Shannon, con una mejora de hasta el 60% con respecto a los actuales estándares.
- *Eficiencia en el Costo del Sistema:* un usuario es capaz de enviar la misma cantidad de información empleando solo la mitad del ancho de banda.
- *Número de Usuarios:* un proveedor de servicio satelital por ejemplo es capaz de duplicar el número de usuarios sin la necesidad de incrementar la capacidad del satélite.

Debido a su excelente rendimiento y al estado de la industria de la microelectrónica actual, las técnicas de turbo codificación están reemplazando los códigos convencionales utilizados hasta ahora en casi todos los sistemas de comunicaciones. Desde las comunicaciones por satélite y aplicaciones inalámbricas hasta el almacenamiento magnético, la turbo codificación está a punto de convertirse en la tecnología de codificación como *estándar de facto*[181].

Para dar efecto a las comparaciones de rendimiento de los diferentes esquemas de codificación para detección y corrección de errores, en el Interfaz Gráfico de Usuario se ha creado una nueva pantalla con nomenclatura *Comparación de Rendimiento* que permite simular dichos esquemas, obtener el BER y graficarlos. Se puede acceder a esta pantalla a través de la pantalla de presentación del

[180] FUENTE: BARBULESCU, Sorín A.; What a Wonderful Turbo World; Pág: 20.
[181] ESTÁNDAR DE FACTO: es aquel patrón o norma que se caracteriza por no haber sido consensuada ni legitimada por un organismo de estandarización al efecto, se trata de una norma generalmente aceptada y ampliamente utilizada por iniciativa propia de un gran número de interesados.

interfaz gráfico. Adicionalmente permite comparar la información obtenida si se varían los polinomios generadores del Turbo Codec. Al iniciar esta pantalla presenta las dos opciones propuestas (*Comparar Turbo Código con Otros Esquemas FEC* y *Comparar Rendimiento de los Códigos Generadores del Turbo Codec*) tal como se muestra en la Figura 3.113, para esa instancia los demás botones se encontraran bloqueados.

Figura 3.113. Pantalla de Comparación de Rendimiento del Turbo Codec.

Si se elige *Comparar Turbo Código con Otros Esquemas FEC* los botones se activan permitiendo al usuario ingresar los parámetros para la simulación, pero mantiene bloqueado el botón *Graficar*, ya que todavía los vectores que contienen la información de los ejes *X* e *Y* no ha sido cargados. El botón *Valores Por Default,* llena automáticamente estos campos con valores predeterminados. La información resultante de la simulación se muestra en la *Ventana de Resultados* y en *Detalles del Proceso* se visualiza los parámetros que se eligieron como datos de entrada, adicionalmente se activa el botón *Graficar*.

Figura 3.114. Resultados de la Comparación del Turbo Código con Otros Esquemas FEC.

Para plasmar de manera gráfica el rendimiento del sistema planteado con diferentes variaciones, en la pantalla *Comparación de Rendimiento* se ha incorporado un botón que tiene como leyenda *Graficar*, y que traslada hacia la pantalla de gráficos BER vs. E_b/N_o, los vectores correspondientes a estos dos parámetros con el objeto de plotearlos. La Figura 3.115 muestra la pantalla de gráficas, cuando ha sido ejecutada una comparación, entre diferentes esquemas de codificación FEC.

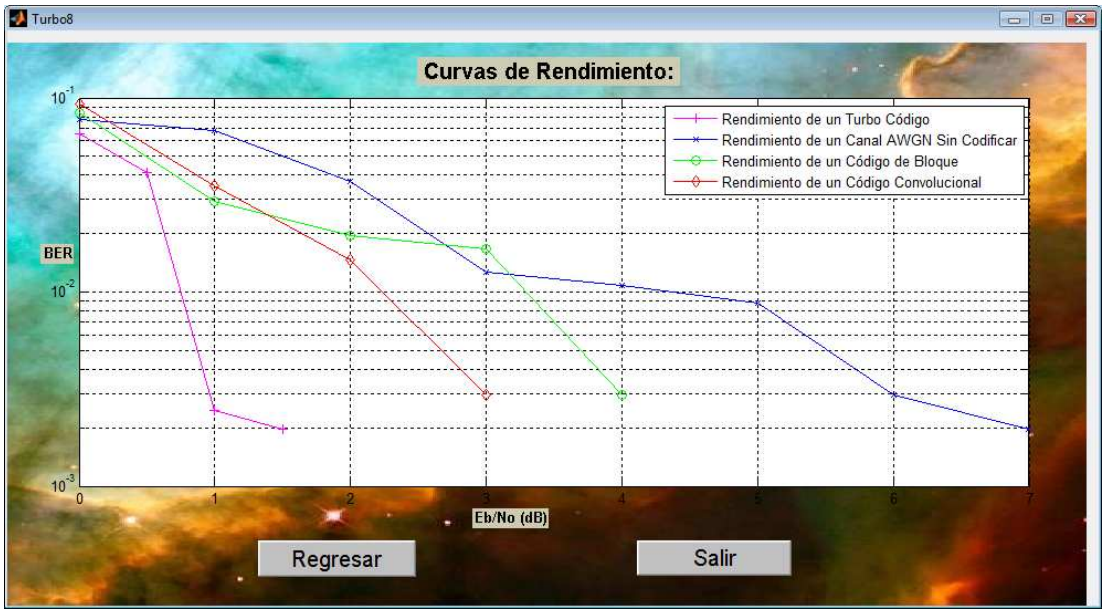


Figura 3.115. Pantalla de Comparación de Gráficas: BER vs. Eb/No con Códigos FEC.

Contrariamente, si se elige en la parte superior de la pantalla *Comparar Rendimiento de los Códigos Generadores del Turbo Codec,* se bloquean las opciones que permitían confrontar los esquemas FEC, y se habilitan los campos destinados a comparar el rendimiento de los polinomios generadores, al igual que en el caso anterior el botón *Valores Por Default,* llena estos espacios con valores predeterminados.

Con el botón *Simular* se inicia el proceso que determinará cual código generador presenta mejores características, y sus resultados se visualizan en la *Ventana de Resultados*. La Figura 3.116 ilustra el proceso simulado.

Figura 3.116. Resultados de la Comparación de Rendimiento de los Códigos Generadores en el Turbo Codec.

La Figura 3.117 a continuación mostrada, presenta la pantalla de gráficas al compararse diferentes polinomios generadores con una misma secuencia de bits.

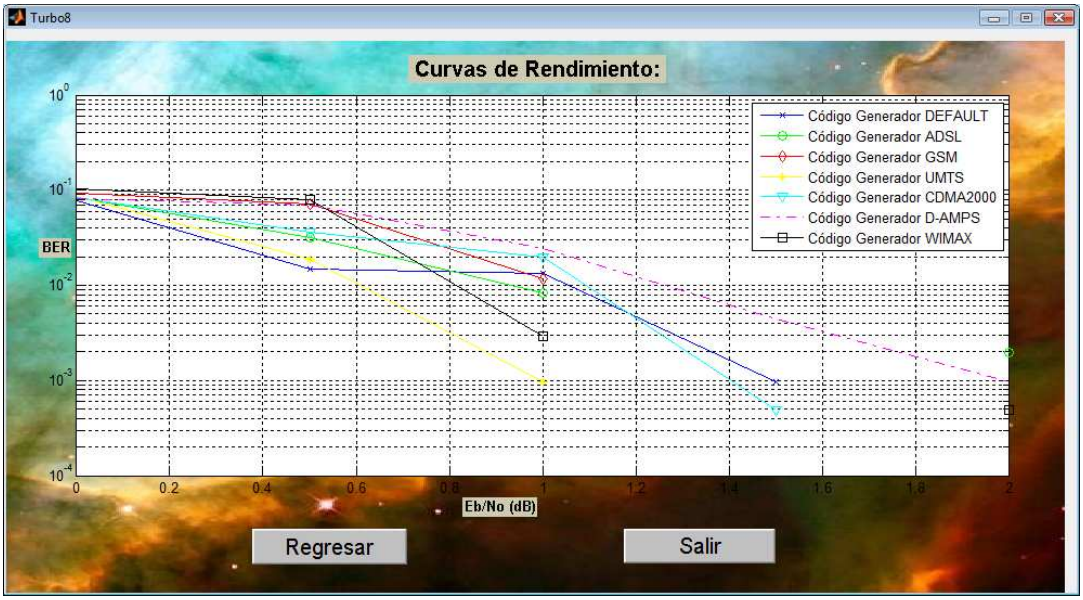


Figura 3.117. Pantalla de Comparación Gráficas: BER vs. Eb/No con Varios Códigos Generadores

Finalmente, en la Figura 3.118 el diagrama de flujo de la pantalla *Comparación de Rendimiento* es mostrado.

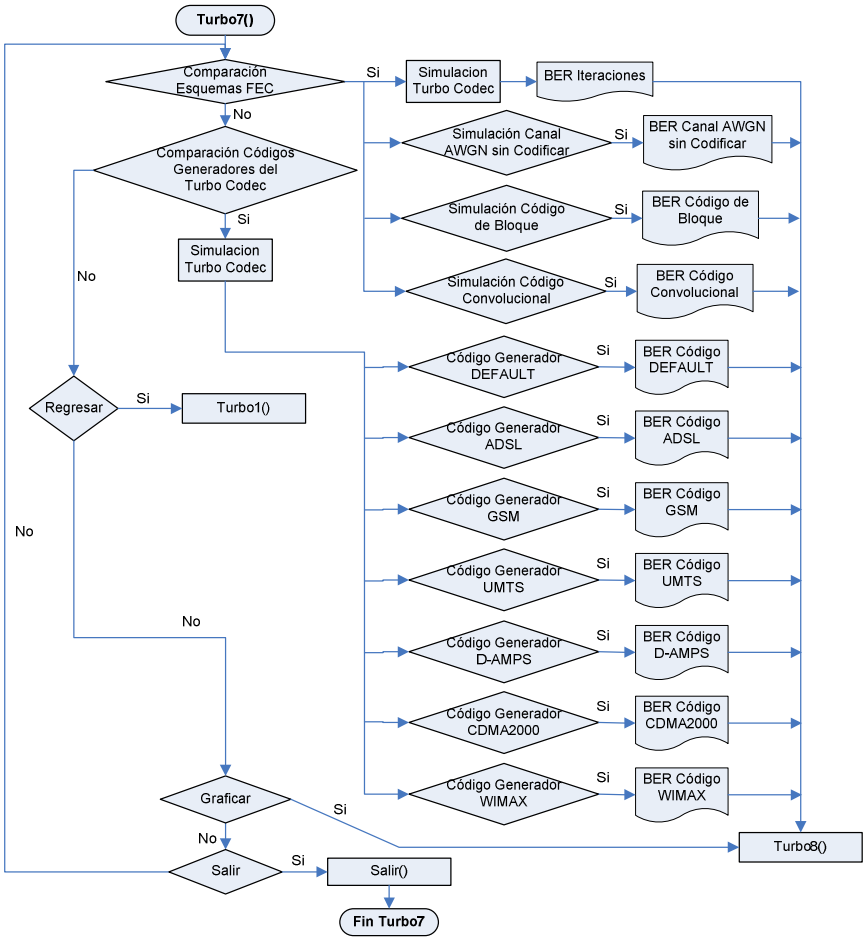


Figura 3.118. Diagrama de Flujo de la Pantalla Comparación de Rendimiento.

CAPÍTULO 4

CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

- El análisis de las curvas de desempeño de los Turbo Códigos BER vs. E_b/N_o, permiten plasmar el efecto que produce el variar algunos parámetros importantes como el algoritmo de decodificación SISO, el número de iteraciones utilizadas en el proceso de decodificado, la tasa de código y el tamaño y diseño del interleaver. De esta manera se concluye, que a mayor tamaño del interleaver y mayor número de iteraciones utilizadas, se incrementa el desempeño ofrecido por un Turbo Código pero aumenta también la memoria y el tiempo requeridos por el turbo decodificador para producir su secuencia de salida. Es por ello que el tamaño del interleaver y el número de iteraciones deben elegirse de forma tal que se satisfagan los requerimientos del sistema de BER, tardanza de decodificación y memoria utilizada por el decodificador.

- [182]El sistema ideal de Shannon tiene un límite teórico que se encuentra alrededor de los 1.2 dB para un BER de 10^{-8}. [183]Teniendo en cuenta que para las aplicaciones que usan al Turbo Código como esquema FEC se considera tener una tasa de bits errados en el rango ***medio*** cuando el BER fluctúa entre 10^{-2} y 10^{-6}, una tasa ***baja*** cuando el BER fluctúa entre 10^{-6} y 10^{-11} y una tasa ***muy baja*** cuando el BER es mayor a 10^{-11}. Se puede concluir que el Turbo Código configurado con los parámetros que se muestran en la Tabla 4.12, a pesar de ser un esquema para fines didácticos y encontrarse únicamente examinado a nivel de simulación tiene un rendimiento aceptable que se enmarca dentro del nivel ***medio***.

[182] FUENTE: BERROU, Claude; The Ten-Year-Old Turbo Codes are Entering to Service; Pág: 8.
[183] FUENTE: BERROU, Claude; The Ten-Year-Old Turbo Codes are Entering to Service; Págs: 15-16.

- El valor de confiabilidad del canal $\boldsymbol{L_c}$ es uno de los parámetros que determinan notablemente cambios radicales en el comportamiento del Turbo Codec. Así, la función utilizada por el MPRG (tanto para el algoritmo Log-MAP como SOVA) que omite dicho parámetro presenta algunos problemas el momento de tratar de recuperar el mensaje aumentando el BER, es por esta razón que al incluir este término en el cálculo de las salidas suaves del decodificador, a pesar de aumentar el tiempo de procesamiento para encontrar la secuencia original, se mejoró el rendimiento.

- Uno de los factores clave para reducir tanto la latencia así como la complejidad en la simulaciones de Matlab, fue la configuración del Trellis, por medio de la función *poly2trellis,* ya que sus salidas han sido diseñadas de manera que no solo obtengan la respuesta esperada sino que también optimicen los recursos del computador.

- Los resultados de las simulaciones nos han mostrado la gran ventaja que supone utilizar turbo codificación frente a un canal sin codificación para control de errores, un código de bloque y la codificación convolucional convencional para longitudes de paquetes de entrada medianos, pero también que no siempre es preferible utilizar la turbo codificación frente a la codificación convolucional. Para longitudes de entrada menores a 160 bits, por ejemplo, se debe utilizar esta última en lugar de la turbo codificación.

- En la práctica no se utiliza un número de iteraciones muy alto para el proceso de turbo decodificación, ya que esto implica mayor tiempo de procesamiento por parte de la computadora en la parte del receptor, haciendo más lento todo el sistema.

- Se da a conocer el uso de una herramienta computacional creada para analizar códigos que detectan y corrigen errores, misma que podría ser usada para desarrollar nuevas técnicas de codificación de canal que mejorarían la confiabilidad de los sistemas de comunicación digitales

existentes o futuros. Con el desarrollo de la interfaz gráfica de esta herramienta de simulación, se presentan de forma clara y explícita cada una de las variables que determinan el rendimiento de los Turbo Códigos, facilitando la labor de investigación, y a la vez, permitiendo su manejo a usuarios con ideas básicas en comunicaciones que deseen conocer la estructura, el funcionamiento y las ventajas de este importante esquema de codificación.

4.2 RECOMENDACIONES

- Para dar continuidad a este trabajo es recomendable que se realice la implementación a través de hardware del prototipo de turbo codificación y decodificación propuesto. En la actualidad la herramienta Quatrus II de Altera Corp, permite comprobar los parámetros de tiempo que requiere el sistema y la asignación de pines para la implementación del Turbo Cocec. HDL Design Series permite la descripción a nivel de diagrama de bloques para luego ser convertida a VHDL. La implementación del interleaver sobre un lenguaje de inferencia de hardware como es el VHDL hace posible su portabilidad a cualquier tipo de tecnología, desde las FPGAs, hasta un diseño 100% ASIC.

- Se recomienda buscar nuevas aplicaciones para el campo de los Turbo Códigos, para realizar comparaciones con los esquemas de codificación utilizados actualmente en esas plataformas, y pensar cuales son los beneficios que implicarían el modificar su esquema de corrección de errores.

- Se recomienda elaborar una propuesta de Turbo Código para un estándar, como es el caso de las comunicaciones móviles 3G que utilizan Turbo Códigos de manera oficial estandarizada. Los Turbo Códigos son una opción muy atractiva para formar parte de estos estándares, pues la ganancia de codificación tan importante que presentan, brinda la posibilidad de transmitir información a tasas muy elevadas, cercanas a la

capacidad del canal, y con un muy buen nivel de confiabilidad (BER ≈ 10^{-6}). Así mismo, brinda la posibilidad de que los dispositivos móviles reduzcan sus niveles de potencia de transmisión, y por consiguiente, de que un mayor número de usuarios pueda compartir el mismo espectro de frecuencia.

- Es recomendable realizar un diseño del interleaver y patrones de perforado o puncturing adecuado, tomando en cuenta parámetros importantes como el análisis de características (distancia, RSC), ya que este es uno de los parámetros que más afecta en el rendimiento del Turbo Codec.

- Es recomendable dar continuidad a este trabajo con la investigación de los Turbo Códigos utilizando algunas modificaciones del esquema clásico de Turbo Codificación como son: la concatenación con otros códigos, la utilización de otros algoritmos de decodificación, etc.

- Para un trabajo futuro, se sugiere analizar la arquitectura SCCC, ya que en este tipo de Turbo Código, el *error floor* en su curva de desempeño se produce a un nivel mucho más bajo de BER que en el caso de la arquitectura PCCC. Esto trae grandes ventajas consigo.

- Se sugiere trabajar sobre los denominados *stream-oriented Turbo-codes*, que consisten en un esquema en el que el Turbo Código, en vez de ser utilizado en conjunto como un gran código de bloque lineal, se utiliza como un gran código Convolucional. Este tipo de Turbo Código presenta grandes ventajas cuando es aplicado en sistemas de conmutación de circuitos.

REFERENCIAS BIBLIOGRÁFICAS

LIBROS:

[1] BARBULESCU, Sorín A.; What a Wonderful Turbo World. Electronic Book, Version 1.2, ISBN 0-9580520-0-X, 2004.

[2] CAMPANELLA, Humberto; NARDUCCI, Margarita; Técnicas y Tecnologías de Comunicaciones Móviles 3G. Electronic Book, ISBN 9588133939. Ediciones Uninorte. Barranquilla. 2005.

[3] EBERSPÄCHER, Jörg; VÖGEL, Hans-Jörg; BETTSTETTER, Christian; HARTMANN, Christian; GSM-Architecture, Protocols and Services. Tercera Edición. John Wiley & Sons Ltd. England. 2009.

[4] GLAVIEUX, Alain; Channel Coding in Communication Networks: From Theory to Turbocodes. Segunda Edición. ISTE Ltd. United States of America. 2007.

[5] HANZO, L.; LIEW, T.; YEAP, B.; Turbo Coding, Turbo Equalization and Space-Time Coding for Transmission over Wireless Channels. Electronic Book, Department of Electronics and Computer Science, Ltd. United Kingdom. 2002.

[6] HARTE, Lawrence J.; SMITH, Adrian D.; JACOBS, Charles A.; Is-136 Tdma Technology, Economics and Services. Artech House Mobile Communications Library. Octubre 1998.

[7] KEATTISAK, Sripimanwat; Turbo Code Applications: A Journey From a Paper to Realization. National Electronics and Computer Technology Center (NECTEC). Pathumthani, Thailand. Octubre, 2005.

[8] LIN, Shu; COSTELLO, Daniel; Error Control Coding: Fundamentals and Applications. Englewood Cliffs, N.J: Prentice-Hall. United States of America. 1983.

[9] MOON, Todd K.; Error Correction Coding: Mathematical Methods and Algorithms. Wiley Interscience. New Jersey, United States of America. 2005.

[10] NEUBAUER, André; FREUDENBERGER, Jürgen; KÜHN, Volker; Coding Theory: Algorithms, Architectures, and Applications. John Wiley & Sons Ltd. England. 2007.

[11] PROAKIS, John G.; Digital Communications. Tercera Edición. McGraw-Hill. New York. 1995.

[12] SCHLEGEL, Christian B.; PÉREZ Lance C.; Trellis and Turbo Coding. Sexta Edición. IEEE Press Series on Digital & Mobile Communication. United States of America. 2004.

[13] SOLEYMANI, M. R.; GAO, Yingzy; VILAIPORNSAWAI, U.; Turbo Coding for Satellite and Wireless Communications. Kluwer Academic Publishers. United States of America. 2002.

[14] TOMASI, Wayne; Sistemas De Comunicaciones Electrónicas. Cuarta Edición. Prentice Hall. 2003

ARTÍCULOS:

[1] ALVARADO, Raúl; Códigos para Detección y Corrección de Errores en Comunicaciones Digitales. Universidad Autónoma de Nuevo León, Facultad de Ingeniería Mecánica y Eléctrica. Vol. 7, N°25. Octubre-Diciembre 2004.

[2] ARÁUZ, Julio; Discrete Rayleigh Fading Channel Modeling. Universidad de Pittsburgh, Departamento de Información Ciencias y Telecomunicaciones. Marzo, 2002.

[3] BARBULESCU, Adrian S.; PIETROBON, Steven S.; TURBO CODES: A Tutorial on a New Class of Powerful Error Correcting Coding Schemes, Part II: Decoder Design and Performance. Institute of Telecommunications Research, University of South Australia. Octubre, 1998.

[4] BERROU, Claude; The Ten-Year-Old Turbo Codes are Entering to Service. IEEE Communications Magazine. ENST Bretagne, France. Agosto, 2003.

[5] European Telecommunications Standards Institute (ETSI); Universal Mobile Telecommunications System (UMTS): Multiplexing and Channel Coding (FDD) – Release 4; 3GPP TS 25.212, Version 4.3.0. Diciembre, 2001.

[6] GILLEY, James E.; Bit-Error-Rate Simulation Using Matlab. Transcrypt International, Inc. Agosto 19, 2003.

[7] HAGENAUER, Joachim; OFFER, Elke; PAPKE, Lutz; Iterative Decoding of Binary Block and Convolutional Codes. IEEE Transactions on Information Theory. Vol. 42, N°2. Marzo, 1996.

[8] LANGTON, Charan; Intuitive Guide to Principles of Communications: Coding and Decoding with Convolutional Codes. Loral Space Systems. Complex to Real. Julio, 1999.

[9] LANGTON, Charan; Intuitive Guide to Principles of Communications: Turbo Coding and MAP Decoding, Parte 1. Loral Space Systems. Complex to Real. 2006.

[10] LARA, Belén; Codificación de Datos: Nuevas Tecnologías en Comunicaciones Móviles. Revista Digital: Investigación y Educación. ISSN 1696-7208. Revista número 10 de Septiembre de 2004.

[11] MAN CHEUK, Ng; MURALIDARAN, Vijayaraghavan; NIRAV, Dave; ARVIND; From WiFi to WiMAX: Techniques for High-Level IP Reuse across Different OFDM Protocols. Massachusetts Institute of Technology, Computer Science and Artificial Intelligence Laboratory.

[12] RODRÍGUEZ, Nibaldo; PALMA, Wenceslao; SOTO, Ricardo; Evaluación de Rendimiento de Turbo Código con Diferentes Intercaladores de Bits para Comunicaciones Satelitales. Pontificia Universidad Católica de Valparaíso, Escuela de Ingeniería Informática. 2004.

[13] ROQUE; SÁENZ; PEÑA; Comunicación de Datos: Códigos Convolucionales. Universidad Nacional de Quilmes, Departamento de Ciencia y Tecnología. Buenos Aires, Argentina.

[14] THE MATHWORKS; Communications Toolbox™ 4: User's Guide. The MathWorks, Inc; 3 Apple Hill Drive; Natick, MA. Marzo, 2009.

[15] Third Generation Partnership Project 2 (3GPP2); Physical Layer Standard for cdma2000 Spread Spectrum Systems - Release C; 3GPP2 C.S0002-C, Version 1.0. Mayo, 2002.

[16] WOODWARD, Jason P.; HANZO, Lajos; Comparative Study of Turbo Decoding Techniques: An Overview. IEEE Transactions on Vehicular Technology, Vol. 49, N°6. Noviembre, 2000.

PROYECTOS DE TITULACIÓN:

[1] CABALLERO Z., Rafael; Master's Thesis: Use of Convolutional Codes for Collaborative Networks. Dept. of Electrical and Computer Engineering, Concordia University; Montreal. Julio 20, 2009.

[2] FRANCOS, Alfonso; Estudio Teórico de la Arquitectura de Turbo-Códigos para Aplicaciones de Telefonía Celular de 3G. Universidad de las Américas Puebla, Escuela de Ingeniería y Ciencias, Departamento de Computación, Electrónica y Mecatrónica; Puebla - México. Septiembre 17, 2007.

[3] HUANG, Fu-hua; Master's Thesis: Evaluation of Soft Output Decoding for Turbo Codes. Virginia Polytechnic Institute and State University. Mayo 29, 1997.

[4] INGVARSSON, Per Ola; SVENELL, Henrik; Master Thesis: Error Performance of Turbo Codes. Diciembre, 1998.

[5] SACANAMBOY, Maribell; Tesis de Maestría: Diseño e Implementación de los Turbo Codificadores definidos en los estándares de Telecomunicaciones cdma2000 (TIA/EIA 2002.2D) y WCDMA (3GPP TS 25.212 v7.2.0) usando Hardware Reconfigurable. Universidad del Valle, Facultad de Ingeniería; Santiago de Cali. Octubre 20 del 2006.