ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELECTRÍCA Y ELECTRÓNICA

ESTUDIO DE LOS EFECTOS REALES DEL CANAL INALÁMBRICO POR MEDIO DE LA TECNOLOGÍA DE RADIO DEFINIDO POR SOFTWARE Y TÉCNICAS PARA SU COMPENSACIÓN.

COMPARACIÓN Y ANÁLISIS DE RESULTADOS OBTENIDOS EMPLEANDO LAS TÉCNICAS DE CORRECCIÓN DE ERRORES (CONVOLUCIONAL, REED SOLOMON Y TURBO CÓDIGOS) EMPLEADOS EN DISTINTOS ESTÁNDARES.

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN TELECOMUNICACIONES

CRISTIAN MIGUEL GUANOPATÍN PAILLACHO

cristian.guanopatin@epn.edu.ec

DIRECTOR: DR. ROBIN ÁLVAREZ

robin.alvarez@epn.edu.ec

DMQ, febrero 2022

CERTIFICACIONES

Yo, Cristian Miguel Guanopatín Paillacho declaro que el trabajo de integración curricular 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.

CRISTIAN GUANOPATÍN

Certifico que el presente trabajo de integración curricular fue desarrollado por Cristian Miguel Guanopatín Paillacho, bajo mi supervisión.

ROBIN ÁLVAREZ
DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

CRISTIAN GUANOPATÍN

ROBIN ÁLVAREZ

DEDICATORIA

Este trabajo lo dedico a mis abuelos Sabina y Rafael, a mis padres Lourdes y Guillermo, a mi tío David, a mi tía Luz y a todos mis hermanos y hermanas, quienes fueron parte fundamental de este proceso.

A mis amigos que siempre estuvieron apoyándome para que siga adelante y no me deje vencer.

AGRADECIMIENTO

A mis abuelos Sabina y Rafael, que pese a todos los problemas que se han suscitado desde el momento que nací siempre me han sabido brindar todo su apoyo incondicional, inculcándome valores y principios que eternamente estaré agradecido.

A mi madre Lourdes, que es la persona que nunca ha dejado de confiar en mí, y que con su amor incondicional siempre me motiva para seguir adelante.

A mi padre Guillermo, que supo estar en momentos exactos de mi carrera universitaria.

A mi tío David, que lo considero como un hermano porque siempre desde pequeño a estado velando por mi bien.

A mi tía Luz que siempre ha estado pendiente de mi.

A mis hermanas, que con sus locuras siempre me sacan una sonrisa.

A mis amigos, que con todas las aventuras que vivimos hicieron de este camino demasiado divertido.

A mi tutor, que con su paciencia ha sabido guiarme de la mejor manera para realizar este proyecto.

ÍNDICE DE CONTENIDO

RESUMEN

En el presente trabajo se realiza el estudio, simulación e implementación práctica, empleando tecnología SDR (Software Defined Radio) sobre tres diferentes técnicas de corrección de errores: Convolucional, Reed – Solomon y Turbo códigos. Se analiza cuál de las tres resulta ser la mejor opción al momento de enviar información en tiempo real. Se debe tener en cuenta que la técnica convolucional y Turbo códigos son a nivel de bit y Reed – Solomon a nivel de byte.

Tanto en la parte de simulación como en la parte práctica se manejan distintos mapeos para analizar su influencia en la cantidad de bits errados. En la parte práctica se utilizan dos equipos SDR Adalm Pluto conectados cada uno a un computador diferente de manera que se puedan alejar una distancia considerable.

Después de realizadas las pruebas respectivas, en todas las distancias y para todas las opciones de mapeo, se concluye que la mejor técnica es Turbo códigos seguida por Reed – Solomon y finalmente está el Convolucional.

PALABRAS CLAVE: Convolucional, Reed - Solomon, Turbo códigos, SDR Adalm Pluto, bit, BER, Matlab, .txt.

ABSTRACT

In the present work, the study, simulation and practical implementation using SDR (Software Defined Radio) technology are carried out, on three different error correction techniques: Convolutional, Reed - Solomon and Turbo codes. It is analyzed which of the three turns out to be the best option when sending information in real time. It should be noted that the convolutional technique and Turbo codes are at the bit level and Reed-Solomon at the byte level.

Both in the simulation part and in the practical part, different mappings are used to analyze their influence on the number of erroneous bits. In the practical part, two SDR Adalm Pluto devices are used, each one connected to a different computer so that they can be separated a considerable distance.

After performing the respective tests, at all distances and for all mapping options, it is concluded that the best technique is Turbo codes followed by Reed - Solomon and finally the Convolutional.

KEYWORDS: Convolutional, Reed - Solomon, Turbo codes, SDR Adalm Pluto, bit, BER, MATLAB, .txt.

1. INTRODUCCIÓN

En el presente trabajo de integración curricular en primer lugar se realiza un estudio teórico de nuestras tres diferentes técnicas de corrección de errores que vamos analizar las cuales son las siguientes: Convolucional, Reed – Solomon y Turbo códigos, en donde se investiga cuál de estas tres técnicas resulta ser la mejor opción al momento de realizar una transmisión y recepción de cualquier texto. Después de realizar el estudio nos enfocamos en crear un programa en la plataforma de programación Matlab desarrollado por MathWorks para cada técnica y con estos scripts podemos mostrar o verificar si la teoría investigada va de acuerdo con la práctica, esto quiere decir que los resultados que obtengamos en cada técnica tienen que ir de la mano de la teoría o si no sacar las respectivas conclusiones y comparaciones. En las simulaciones debemos de tener muy en cuenta que algunas técnicas se manejan a nivel de bit las cuales son: Convolucional y Turbo códigos y tenemos a Reed – Solomon la cual se maneja a nivel de byte.

En la parte de simulación y práctico se tendrá varios tipos de mapeos para escoger los cuales son: BPSK, QPSK, 16QAM y 64 QAM.

Cuando ya realizamos el análisis teórico y práctico de nuestras tres diferentes técnicas de corrección de errores seguimos con el siguiente paso el cual es enfocarse en realizar una transmisión y recepción de un texto en tiempo real utilizando dos equipos llamados SDR Adalm Pluto, dos computadoras (laptops), una que nos sirva de transmisor y otra de receptor.

Al realizar esta simulación en tiempo real nosotros nos vamos a manejar con diferentes distancias, siendo la distancia más larga de 18 metros, otra aproximadamente el 70% del total de distancia que vendría a darnos 13 metros, la siguiente nos manejamos con el 50% que sería 9 metros y la última distancia va a ser aproximadamente el 30% del total que vendría a darnos 5 metros y ahí verificaremos que tanto puede afectar la distancia al aplicar las tres distintas técnicas de corrección de errores. Cabe recalcar que nuestra simulación va hacer con línea de vista, sin ningún obstáculo para verificar nuestro análisis teórico y así obtener distintas conclusiones y comprobar cuál será la mejor técnica para aplicarla en futuras aplicaciones.

1.1. OBJETIVO GENERAL

Realizar el estudio, simulación e implementación práctica de los tres métodos de corrección de errores: Convolucional, Reed - Solomon y Turbo códigos.

1.2. OBJETIVOS ESPECÍFICOS

1.2.1. Realizar el estudio, simulación e implementación práctica del método de corrección de errores Convolucional.

1.2.2. Realizar el estudio, simulación e implementación práctica del método de corrección de errores Reed - Solomon.

1.2.3. Realizar el estudio, simulación e implementación práctica del método de corrección de errores Turbo códigos.

1.3. ALCANCE

En primer lugar, se realiza una simulación de las tres técnicas en donde codificamos y decodificamos un texto y se comprueba cuantos caracteres errados vamos a tener con las distintas técnicas de corrección de errores.

La información que se va a transmitir en tiempo real va a estar en formato de texto y esta parte se va a realizar con el uso de dos SDR Adalm Pluto y dos laptops, una para el transmisor y otra para el receptor separados por una distancia de 18 metros.

En la Figura 1.1 se muestra la forma en la cual se va a realizar la parte de las simulaciones, en donde tenemos una laptop y un SDR Adalm Pluto con sus respectivas antenas de transmisión y recepción.

Figura 1.1 Diagrama representativo de la parte de simulación del proyecto.

En la Figura 1.2 se muestra el diagrama representativo de nuestro proyecto práctico en donde nos manejamos con dos laptops, dos SDR Adalm Pluto con sus respectivas antenas y trabajando a una distancia máxima de 18 metros

Figura 1.2 Diagrama representativo del proyecto práctico para trabajar en tiempo real.

1.4. MARCO TEÓRICO

1.4.1. SDR ADALM PLUTO

En nuestro proyecto se va a manejar equipos llamados Adalm Pluto (Figura 1.3) con tecnología SDR (Software Defined Radio), tiene un rango de frecuencia de operaciones que va desde los 325 MHz hasta los 3.8 GHz, posee unas antenas que trabajan en dos diferentes bandas de frecuencia: la primera se maneja entre los 824 hasta los 894 MHz y la segunda se maneja entre los 1710 hasta los 2170 MHz. [1]

Figura 1.3. Dispositivo SDR Adalm Pluto

Los principales elementos dentro de nuestro SDR Adalm Pluto son:

- Una parte de RF analógica.
- Una parte de banda base analógica.
- Varias unidades de procesamiento de señal.

El SDR de Adalm Pluto tiene la capacidad de realizar múltiples aplicaciones como: GPS, receptor de satélite meteorológico, redes de área personal ad-hoc, etc.

En la parte delantera del Adalm Pluto SDR tenemos un transceptor AD9363 de *Analog Devices Inc.*, el cual nos ayuda a capturar y digitalizar los datos de RF. Este transceptor nos brinda amplificación, conversión digital y filtrado de señales transmitidas y recibidas.[2]

1.4.1.1. CONFIGURACIÓN SDR ADALM PLUTO

El paquete llamado *"Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio"* permite utilizar el dispositivo SDR Adalm Pluto para diseñar y verificar sistemas prácticos, en condiciones reales, con Matlab y Simulink [3]. La configuración del dispositivo se realiza a través de objetos globales, uno para el transmisor y otro para el receptor. Antes de describir la configuración en Matlab hay que tener en cuenta las especificaciones del dispositivo SDR, que se muestran en la Tabla 1.1.

Tabla 1.1 Especificaciones principales del dispositivo SDR Adalm Pluto [1].

Especificaciones	Valores típicos
Entrada DC (USB)	4.5V a 5.5V
Frecuencia de muestreo ADC y DAC	65.2 ksps a 61.44 Msps
Resolución ADC y DAC	12 bits
Precisión de frecuencia	±25 ppm
Rango de Sintonización	325 MHz a 3800 MHz
Ancho de banda instantáneo	Hasta 20MHz
Conectores	SMA 50 Ω
Modo de transmisión	Half duplex o Full duplex
Potencia de salida en Tx	7dBm
Figura de ruido Rx	< 3.5 dB
Precisión de modulación en Tx y Rx	-34 dB (2%)
Blindaje RF	No
USB	2.0 On-the-Go
Temperatura	10°C a 40°C

Para mayor información de esta parte se puede encontrar en la tesis de la siguiente referencia [4].

Es importante mencionar que debemos de configurar los dos SDR Adalm Pluto para la transmisión y recepción. El objeto de transmisión viene dado por sdrtx () que tiene una sintaxis *txAdalm = sdrtx('Pluto',Name,Value),* y posee las siguientes propiedades que se muestras en la Tabla 1.2. adjuntando los valores con los cuales se trabajará en este proyecto.

Tabla 1.2 Propiedades del objeto transmisor SDR Adalm Pluto [9]

PARÁMETRO	DESCRIPCIÓN	VALOR
DeviceName	Se refiere al modelo del dispositivo SDR	'Pluto'
RadioID	Es el identificador de Interfaz USB donde se conecta el SDR	'usb:0'
CenterFrecuency	Es la frecuencia central de RF en Hz	860e6
Gain	Se refiere a la ganancia en dB	-2
Baseband-SampleRate	Es la frecuencia de muestreo en banda base	2000e3
ShowAdvanced-Properties	Muestra las opciones avanzadas	0

El objeto de recepción viene dado por sdrtx () que tiene una sintaxis *rxAdalm = sdrrx('Pluto',Name,Value)* y posee las siguientes propiedades que se muestran en la Tabla 1.3, adjuntando los valores con los que se trabajará en este proyecto.

Tabla 1.3 Propiedades del objeto receptor SDR Adalm Pluto [10]

PARÁMETRO	DESCRIPCIÓN	VALOR
DeviceName	Es el modelo del dispositivo SDR	'Pluto'
RadioID	Es el Identificador de Interfaz USB donde se conecta el SDR	'usb:0'
CenterFrecuency	Es la frecuencia central de RF en Hz	860e6
GainSource	Es la fuente de ganancia:	'Manual'
Gain	Es la ganancia en dB	20
Baseband-SampleRate	Es la frecuencia de muestreo en banda base	2000e3
OutputDataType	Es el tipo de datos de la señal de salida	'single'
SamplesPer-Frame	Número de muestras por trama (tamaño de trama), entero positivo	800000
ShowAdvanced-Properties	Mostrar las opciones avanzadas	1
Frequency-Correction	Valor de corrección de frecuencia en ppm.	0

1.4.1.2. INSTALACIÓN DEL TOOLBOX PARA MANEJO DEL ADALM PLUTO

Se debe instalar una versión de Matlab posterior a 2018-A, dado que estas versiones son compatibles con la librería que permite controlar el dispositivo SDR Adalm Pluto.

Para la operación de dispositivos SDR Adalm Pluto en Matlab, sus desarrolladores han implementado el *toolbox* "*Communications Toolbox Support Package for Analog Devices* ADALM-Pluto Radio", el cual permite configurar las características de funcionamiento del equipo SDR tanto para la transmisión como para la recepción de señales, facilitando realizar pruebas de sistemas inalámbricos como FM, WLAN, etc. [1].

El procedimiento de la instalación del Toolbox se lo encuentra mejor detallado en la tesis de la siguiente referencia [4].

1.4.2. TÉCNICAS DE CORRECCIÓN DE ERRORES

En un sistema de comunicación siempre tendremos diversos problemas cuando realicemos una transmisión y recepción de cualquier tipo de información, y mucho más cuando lo realizamos en tiempo real. Es necesario conocer cuales tipos de corrección de errores se tiene y tratar de aplicar el más adecuado, entre los diversos modelos de corrección de errores están:

- Dígito verificador.
- FEC (Forward Error Correction)
- Código Binario de Golay.
- Código Hamming.
- Bit de paridad.
- Reed-Solomon.

Este proyecto va a estar enfocado en los FEC (Forward Error Correction), los cuales generan una cantidad de datos adicionales para enviarlos con la información a transmitir.

Es necesario identificar como trabaja cada tipo de corrección de errores, hay algunos que se manejan a nivel de bit y otros a nivel de byte.

A la tesis de la siguiente referencia [4] se le añade las siguientes técnicas de corrección de errores.

1.4.2.1. CONVOLUCIONAL

La codificación convolucional es una técnica en donde los bits que se obtienen a la salida del sistema son determinados mediante operaciones lógicas en el bit actual y en un grupo de bits anteriores [5]. Es decir, el codificador convolucional es un sistema con memoria, esto debido a que en la salida depende de los bits previos. Los códigos convolucionales son códigos no sistemáticos, es quiere decir que la información codificada no se puede ver de manera explícita en la palabra codificada [6].

El funcionamiento de esta técnica de codificación se puede representar a través de diagrama de estados, registros, diagrama de árbol, diagrama de trellis y convolución siendo esta última implementada intrínsicamente en los métodos anteriores [7]. El diagrama de trellis es el método más utilizado para describir a los códigos convolucionales y con el cual se implementa el codificador convolucional en Matlab.

El diagrama de Trellis es una representación basada en nodos de los posibles estados que tomara el codificador convolucional [8]. Dicho diagrama será explicado en la parte de transmisión y recepción

El codificador convolucional tendrá un diagrama de Trellis (2, 1, 3) y con polinomios generadores $g_1 = (101)$ y $g_2 = (111)$, para entenderlo mejor se puede tomar como en cuenta el siguiente diagrama de estados de la Figura 1.4.

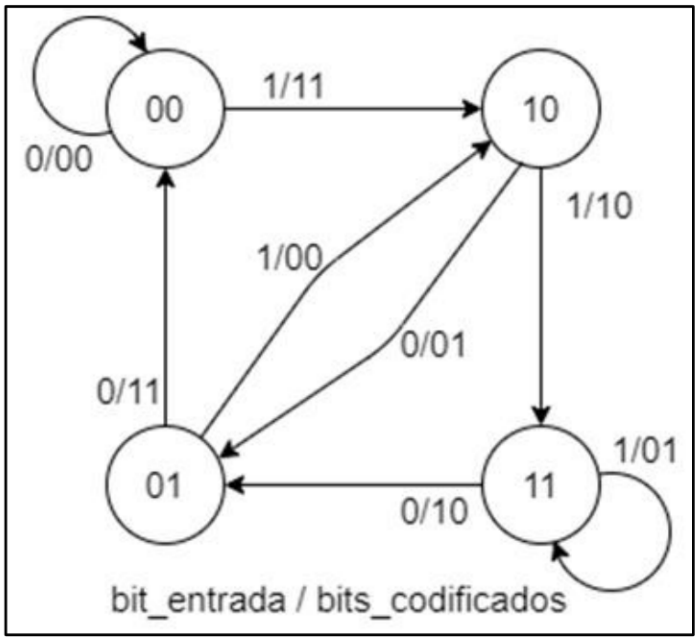


Figura 1.4. Diagrama de estados Codificador convolucional (2, 1, 3).

Para entender la Figura 1.4 hay que seguir los siguientes pasos:

- El estado inicial es 00, por el método de truncado el bit de ingreso es 1 y por lo tanto los bits codificados son 11. 10 es el siguiente estado.
- Después del estado actual como resultado tenemos el valor de 10, luego se ingresa el segundo bit que corresponde a 1 y los bits codificados son 10. 11 es el siguiente estado.
- Luego se ingresa el tercer bit que es 0 y los bits codificados obtenidos son 10. 01 es el siguiente estado.
- Para concluir, se ingresa el cuarto bit que corresponde a 1 y los bits codificados son 00.

En Matlab para realizar el codificador y decodificador se utilizan las siguientes sentencias:

P_Trellis = poly2trellis (ConstraintLength, CodeGenerator)

Los parámetros de entrada se detallan a continuación:

- **ConstraintLength:** especifica un vector fila de longitud ***k*** y el retardo de los flujos de bits de entrada al codificador o la longitud de restricción. Esto quiere decir, que especifica la memoria ***m*** del codificador. [13]
- **CodeGenerator**: específica una matriz de números octales de longitud ***k*** por ***n*** y las conexiones de salida para los bits de entrada al codificador, en otras palabras, los polinomios generadores. Entonces se dice que ***n*** es igual a la cantidad de polinomios generadores que se tienen. [13]

CodConv = convenc (msg, P_Trellis)

Los parámetros de entrada se detallan a continuación:

- **msg:** específica el mensaje binario como un vector de bits.
- **P_Trellis:** es la estructura de trellis que se obtuvo previamente mediante la función *poly2trellis*.

decod_out = vitdec (codedin, P_trellis, tbdepth, opmode, dectype)

La función *vitdec* decodifica convolucionalmente los datos binarios del vector *codedin* utilizando el algoritmo de Viterbi [14]. Los parámetros de entrada se detallan a continuación.

- **codedin:** Es un vector que abarca los datos codificados convolucionalmente. Este vector puede tener valores binarios o numéricos.

- **P_trellis:** Es la estructura de Trellis que se utiliza codificar y decodificar, este parámetro se crea con la función directa *polly2trellis* descrita anteriormente.
- **tbdepth:** Es la profundidad de rastreo y se detalla como un número entero positivo.
- **opmode:** Es el modo de funcionamiento y se puede especificar como; 'cont', 'term' o 'trunc'. Con 'cont' se especifica un modo de funcionamiento continuo. Con 'term' un funcionamiento que inició y terminó con los estados en cero y con 'trunc' se detalla el modo truncado, es decir que el codificador tuvo un inicio en ceros.
- **dectype:** Detalla el tipo de datos a decodificar que contiene *codedin*, puede ser 'unquant', 'hard' o 'soft'. En 'unquant' se espera valores de entrada numéricos. En el modo 'hard' se espera valores binarios y en el modo 'soft' se espera valores de entrada enteros.

1.4.2.2. REED - SOLOMON

El código Reed-Solomon es un subconjunto de los códigos BCH (Bose Chaudhuri Hocquenqhem), son códigos cíclicos que presentan entre sus parámetros (n, k, t) una relación entre los símbolos de datos (k), del código total (n) y del número máximo de errores por ser corregidos (t), y son de bloques lineales. Un código Reed-Solomon se especifica como RS(n, k) con símbolos de s bits. Existen $n - k$ símbolos de paridad de s bits cada uno. Un decodificador puede corregir hasta t símbolos que contienen errores en una palabra de código, donde $2t = (n - k)$. La Figura 1.5 se muestra una típica palabra de código Reed-Solomon que se conoce como un código sistemático puesto que los datos se dejan inalterados y los símbolos de paridad se anexan. [11]

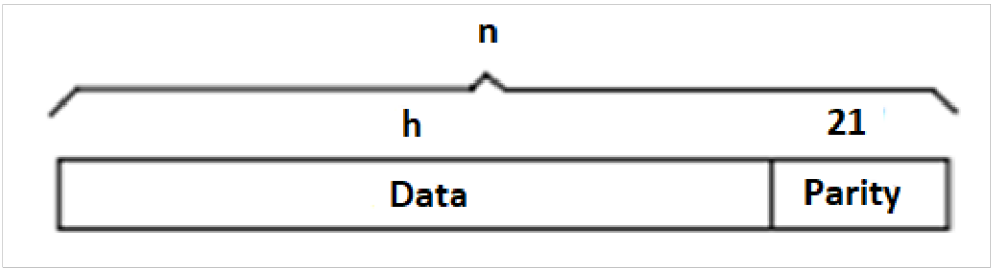


Figura 1.5 Palabra de código Reed – Solomon [11].

El código Reed-Solomon tiene las siguientes características:

- Cada símbolo está constituido por "m" bits consecutivos agrupados.
- Cada palabra-código consta de "k" símbolos de información (en lugar de bits), y "r" símbolos de paridad.

- La longitud de la palabra código es: $n = k + r$ símbolos, (longitud=nm expresada en número de bit).
- Se establece la relación: $n = 2\text{^}m - 1$ entre la longitud de la palabra código (n) y el número de símbolos $(2\text{^}m)$.
- Es capaz de corregir errores en "t" símbolos, donde $t = r/2$.

En Matlab para realizar el codificador y decodificador se utilizan las siguientes sentencias:

datosGalois = gf (auxDecimal, m)

Se crea una matriz de campo de Galois a partir de la matriz *auxDecimal*. El campo de Galois tiene 2m elementos, donde *m* es un número entero del 1 al 16 [15]. Los parámetros de entrada se detallan a continuación:

- **auxDecimal:** matriz con valores mayores o iguales a cero.
- **m:** Tamaño de símbolo, especificado como un entero positivo de 1 a 16.

CodRS = rsenc (datosGalois, NRS, KRS)

La función rsenc codifica el mensaje datosGalois usando un código Reed – Solomon con el polinomio generador en estado estricto. Cada fila de elementos *KRS* de *datosGalois* representa una palabra de mensaje, donde el símbolo más a la izquierda es el más significativo. NRS es como máximo $2m - 1$. Los símbolos de paridad se encuentran al final de cada palabra en CodRS. [16]. Los parámetros de entrada se detallan a continuación.

- **datosGalois:** Mensaje a codificar en forma de matriz.
- **NRS:** Longitud del codificador, en este caso para el Reed – Solomon ocuparemos un valor de 5.
- **KRS:** Símbolos originales, en este caso para el Reed – Solomon ocuparemos un valor de 3.

decRS = rsdec (datGalois, NRS, KRS)

Los parámetros de entrada se detallan a continuación:

- **datosGalois:** Mensaje a decodificar en forma de matriz.
- **NRS:** Longitud del codificador, en este caso para el Reed – Solomon ocuparemos un valor de 5.
- **KRS:** Símbolos originales, en este caso para el Reed – Solomon ocuparemos un valor de 3.

1.4.2.3. TURBO CÓDIGOS

Los turbo códigos son una clase nueva de códigos de corrección de errores (FEC), que se insertaron, junto con un algoritmo de decodificación. Los Turbocódigos permiten una comunicación fiable y su eficiencia energética está muy cerca del límite teórico predicho por Shannon. Desde su creación, los turbo códigos se han utilizado en aplicaciones de baja potencia, como: las comunicaciones por satélite, aplicaciones de interferencia limitada, servicios de tercera generación (3G) y de comunicaciones móviles.[12]

En mostrar un diagrama de bloques de un Turbo códigos se muestra en la Figura 1.6

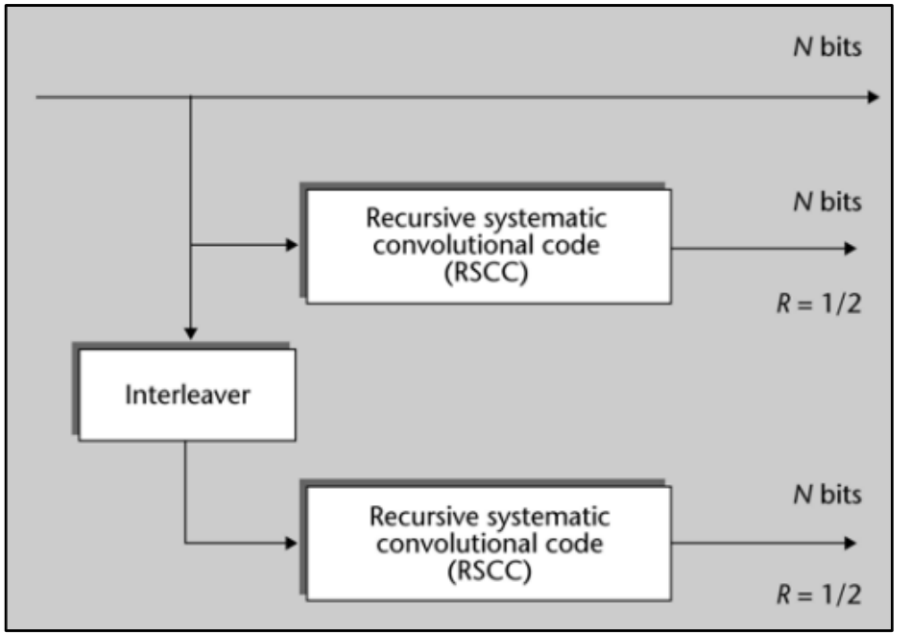


Figura 1.6 Diagrama de bloques de una implementación de Turbo códigos[19].

Los Turbo códigos tienen su función en diferentes aplicaciones:

- Los Turbo códigos son utilizados ampliamente en telefonía móvil 3G y 4G; por ejemplo, en HSPA, EV-DO y LTE.
- En MediaFLO, sistema de televisión móvil terrestre de Qualcomm.
- En el canal de interacción de los sistemas de comunicación por satélite, como DVB-RCS[4] y DVB-RCS2.
- En las misiones de la NASA, como Mars Reconnaissance Orbiter, se utilizan los turbo códigos como alternativa a la corrección de errores Reed-Solomon y Códigos decodificadores Viterbi.

- En la IEEE 802.16 (WiMAX), un estándar de red metropolitana inalámbrica, utiliza codificación turbo de bloque y codificación turbo convolucional.

En Matlab para realizar el codificador y decodificador se utilizan las siguientes sentencias:

turboenc = comm.TurboEncoder (P_Trellis, intrlvrindices)

El objeto *comm.TurboEncoder* aplica un esquema de codificación concatenado paralelo a un mensaje de entrada binario [17]. Los parámetros de entrada se detallan a continuación:

- ***P_Trellis*:** estructura de Trellis, en el Turbo códigos se utiliza el siguiente valor [13 15 17]
- ***intrlvrindices*:** Es un índice de entrelazado como un vector columna de enteros., el vector debe tener una longitud de mensajes de entrada binario.

turbodec = comm.TurboDecoder(P_Trellis, intrlvrindices, numiter)

El objeto comm.TurboDecoder utiliza un esquema de decodificación concatenado paralelo para decodificar una señal de entrada codificada [18]. Los parámetros de entrada se detallan a continuación.

- ***P_Trellis*:** estructura de Trellis, en el Turbo códigos se utiliza el siguiente valor [13 15 17]
- ***intrlvrindices*:** Es un índice de entrelazado como un vector columna de enteros., el vector debe tener una longitud de mensajes de entrada binario.
- ***numiter*:** Símbolos de iteraciones de decodificación.

1.4.3. ETAPAS DE TRANSMISIÓN Y RECEPCIÓN

Para estas etapas nos manejaremos con los siguientes diagramas de bloques para la respectiva transmisión que se encuentra en la (Figura 1.7) y recepción (Figura 1.8)

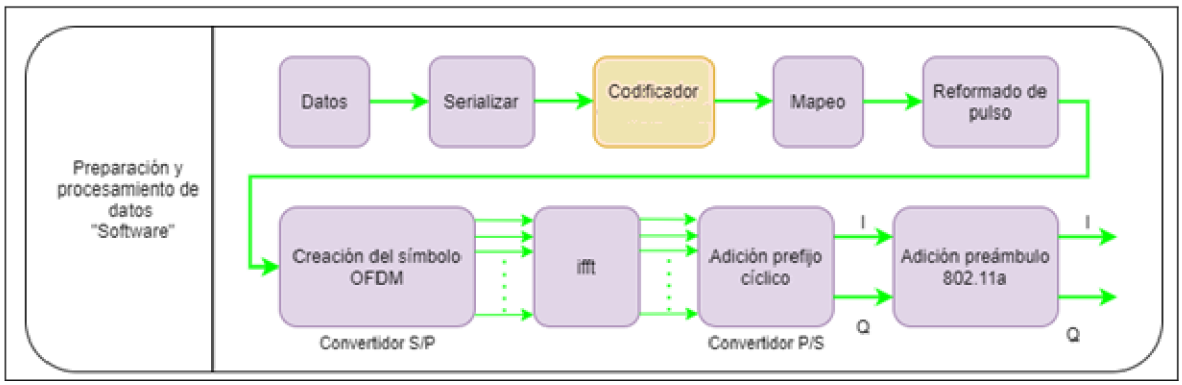


Figura 1.7 Etapas de transmisión.

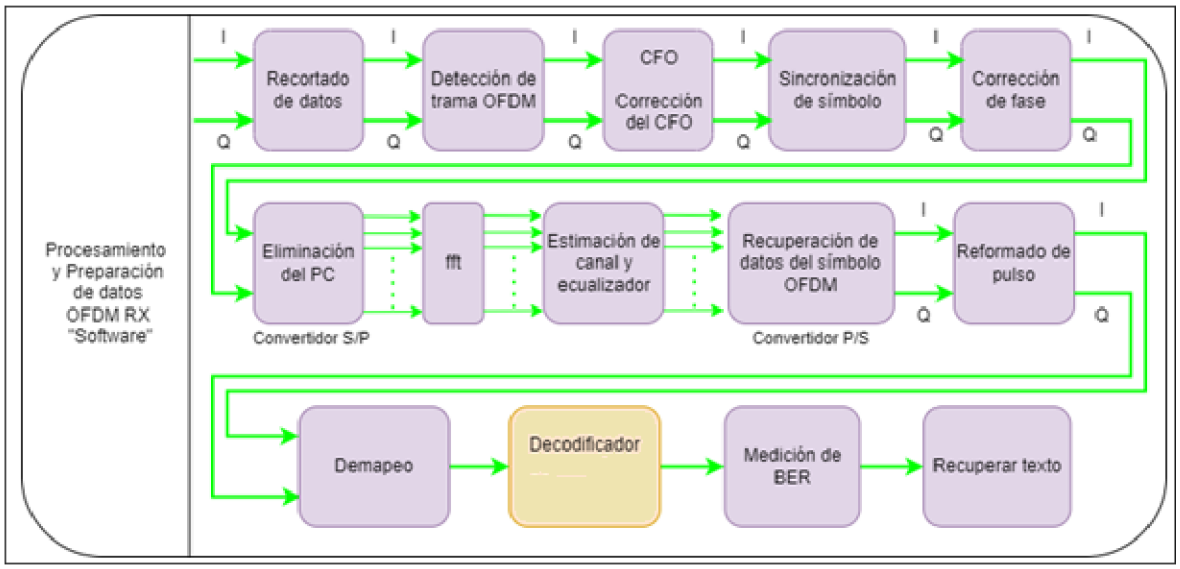


Figura 1.8 Etapas de recepción.

Es necesario mencionar ciertos bloques de estas dos distintas etapas.

LECTURA DE DATOS

Para la lectura de datos se utiliza la sentencia:

uigetfile

Esta sentencia lo que hace es permitir a un usuario seleccionar o introducir el nombre de un archivo. Si el archivo existe y es válido, devuelve el nombre de archivo cuando el usuario hace clic en Abrir. Si el usuario hace clic en Cancelar o en el botón cerrar ventana (X), devuelve la sentencia *.uigetfileuigetfile0* [20].

El programa Matlab reúne cada uno de los caracteres como caracteres Unicode, en donde se encuentran los primeros 128 símbolos que corresponden a caracteres ASCII, esto quiere decir que Unicode y ASCII poseen los mismos códigos numéricos [21]. Esto se aplica para matrices de caracteres y de cadenas (string). Cada carácter es representado con un número decimal, el cual a su vez puede representarse en binario. La Tabla 1.4 muestra los caracteres ASCII imprimibles, con su representación en decimal.

Tabla 1.4. Caracteres del código ASCII [22].

Caracter	Código ASCII	Caracter	Código ASCII	Caracter	Código ASCII	Caracter	Código ASCII
espacio	32	**8**	56	**P**	80	**h**	104
!	33	**9**	57	**Q**	81	**i**	105
"	34	**:**	58	**R**	82	**j**	106
#	35	**;**	59	**S**	83	**k**	107
$	36	**<**	60	**T**	84	**l**	108

%	37	=	61	U	85	m	109
&	38	>	62	V	86	n	110
'	39	?	63	W	87	o	111
(	40	@	64	X	88	p	112
)	41	A	65	Y	89	q	113
*	42	B	66	Z	90	r	114
+	43	C	67	[	91	s	115
,	44	D	68	\	92	t	116
-	45	E	69	]	93	u	117
.	46	F	70	^	94	v	118
/	47	G	71	_	95	w	119
0	48	H	72	`	96	x	120
1	49	I	73	a	97	y	121
2	50	J	74	b	98	z	122
3	51	K	75	c	99	{	123
4	52	L	76	d	100	|	124
5	53	M	77	e	101	}	125
6	54	N	78	f	102	~	126
7	55	O	79	g	103		

MODULACIÓN DIGITAL

Cuando se hable de modulación digital se trata de tener un mensaje digital que modula una señal de forma de onda continua, que manipula la información en amplitud y fase de la señal durante cada cierto periodo de tiempo [23]. Se tiene diferentes técnicas de modulación digital como; modulación por desplazamiento de amplitud (ASK), modulación por desplazamiento de frecuencia (FSK), modulación por desplazamiento de fase (PSK) y modulación de amplitud en cuadratura (QAM) [24]. Sin embargo, hay que tener en cuenta que estas técnicas de modulación digital no pueden ser usadas en transmisiones OFDM (Orthogonal Frequency Division Multiplexing).

OFDM

La transmisión OFDM realiza una multiplexación de un conjunto de ondas portadoras ortogonales. Estas portadoras deben estar moduladas previamente con cualquier técnica de modulación digital, esto debe realizarse sin que se afecte la ortogonalidad de las portadoras [6]. En el estándar WLAN IEEE 802.11A establece varios parámetros pero uno de ellos es el tipo de modulación a utilizar para las portadoras de datos de OFDM y estos son: BPSK, QPSK, 16-QAM y 64-QAM [25].

- BPSK (Modulación por desplazamiento de fase de 2 símbolos)

  Consiste en que cada bit de entrada se le asigna una posición en el diagrama de constelación que se muestra en la Figura 1.9.

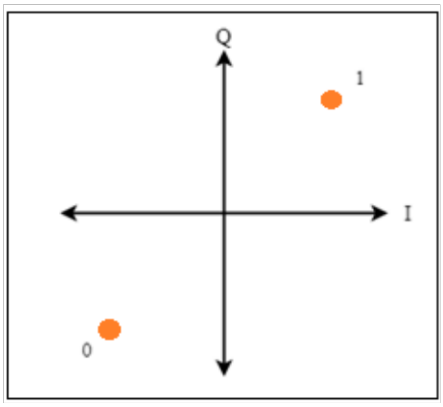


Figura 1.9 Diagrama de constelación para BPSK.

Y contiene sus respectivos símbolos de mapeado $-0.707 - 0.707i$ para 0 bits y $0.707 + 0.707i$ para 1 bit.

- QPSK (Modulación por desplazamiento de fase cuaternario)

En este caso no se utiliza solo 1 bit de entrada, aquí ya se utiliza dos bits de entrada para cada símbolo en el diagrama de constelación, que se muestra en la Figura 1.10.

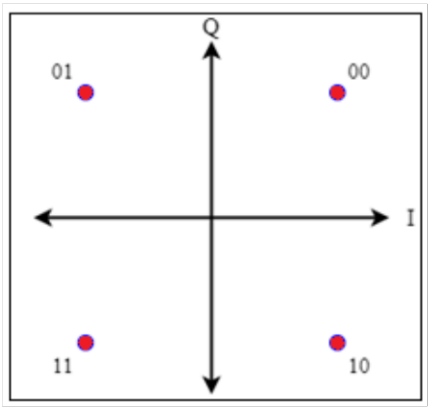


Figura 1.10 Diagrama de constelación para QPSK.

Para expresar sus símbolos se muestra en la Tabla 1.5.

Tabla 1.5 Valores de símbolos para QPSK

Bits	Símbolos
00	+0.707 + 0.707i
01	-0.707 + 0.707i
10	+0.707 - 0.707i
11	-0.707 - 0.707i

- 16 QAM (Modulación de Amplitud en Cuadratura)

QAM es una técnica de modulación digital en donde el mensaje se encuentra en la variación de la fase como es PSK y de amplitud como es ASK. QAM se basa en una modulación donde transmite dos mensajes distintos por un solo camino y adicionalmente se desfasa en 90° una portadora de la otra [24]. 16-QAM contiene un total de 16 símbolos y por cada símbolo se utilizan 4 bits. En la Figura 1.11 se puede observar todos los símbolos con sus bits de entrada para el mapeo 16-QAM.

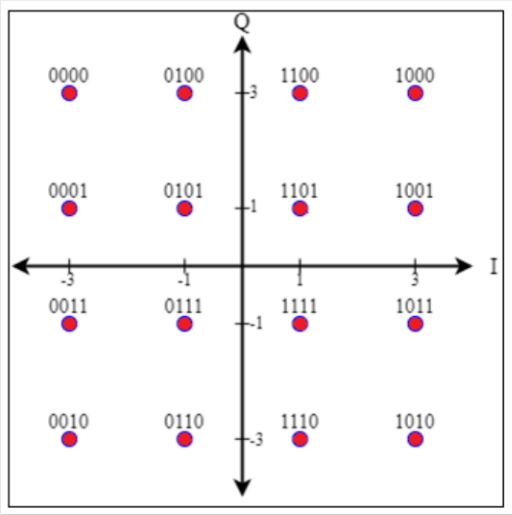


Figura 1.11 Diagrama de constelación 16-QAM

- 64-QAM (Modulación de Amplitud en Cuadratura)

En la modulación 64-QAM se tiene un total 64 símbolos en el diagrama de constelación y cada uno de ellos emplea 6 bits. En la Figura 1.12 se muestran todos los símbolos para el mapeo 64-QAM.

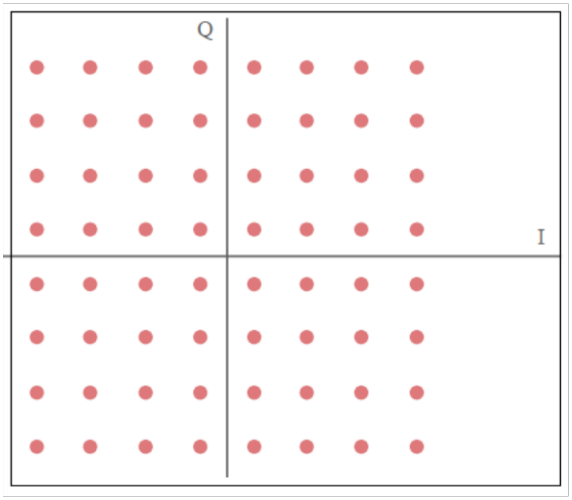


Figura 1.12 Diagrama de constelación 64-QAM

1.4.3.1. TRANSMISIÓN

En el caso de la transmisión se va a seguir los siguientes pasos para las tres técnicas de corrección de errores.

1) Datos. – Selecciona y abre el archivo de texto a transmitir.
2) Serializar. – Se encarga de representar cada carácter en binario y luego serializar los datos.
3) Codificación: En esta etapa que se utiliza para la corrección de errores a nivel de bit (Convolucional y Turbo códigos) y a nivel de byte (Reed - Solomon).
4) Mapeo. – En esta etapa se representa los bits serializados a un esquema de modulación digital que puede ser: BPSK, QPSK, 16-QAM y 64-QAM.
5) Creación del símbolo OFDM. – Aquí se construye cada símbolo OFDM con sus respectivas portadoras de datos, piloto, DC y de guarda.
6) IFFT (Transformada Inversa de Fourier). – En esta etapa se encarga de pasar los símbolos OFDM del dominio de la frecuencia al dominio del tiempo.
7) Adición del prefijo cíclico. – En esta etapa se agrega las últimas muestras de cada símbolo OFDM al inicio de cada uno.
8) Adición preámbulo 802.11a. – En esta etapa se añade el preámbulo que da el estándar IEEE 802.11a. Este preámbulo se compone de secuencias cortas y de secuencias largas.
9) Normalizar datos. - Se evalua la parte real y la parte imaginaria de los vectores, para encontrar el módulo máximo de la parte real e imaginaria.
10) Transmitir los datos. – En esta etapa se transmiten los datos al canal inalámbrico utilizando el equipo SDR Adalm Pluto mediante el objeto global de transmisión.

1.4.3.2. RECEPCIÓN

En el caso de la recepción se va a seguir los siguientes pasos para las tres técnicas de corrección de errores.

1) Selección del archivo para medir el BER. – En esta parte se selecciona el archivo que se transmitió, para poder comparar con el texto recuperado y obtener el valor del BER.
2) Recibir datos. – En esta etapa se recibe los datos del canal inalámbrico mediante el uso de un equipo SDR Adalm Pluto. Para esto, se utiliza el objeto global de recepción *rxAdalm*. El canal va a estar en muestra de manera repetitiva hasta que se detecte los datos transmitidos.

3) Recortado de datos. – En esta etapa lo principal es el recorte los datos recibidos mediante el umbral para reducir la carga de procesamiento en las etapas posteriores.
4) Detección de trama OFDM. – En esta etapa se permite detectar si el bloque de datos pertenece a una secuencia OFDM.
5) Corrección de CFO. – En esta etapa se permite estimar y corregir el desplazamiento de frecuencia de los datos que se recibe.
6) Sincronización de símbolo. – En esta etapa es donde se determina el comienzo exacto de la trama OFDM, esto significa que se realiza la sincronización en tiempo.
7) Corrección de fase. – En esta etapa se estima y se corrige el desplazamiento de fase de todo el bloque de datos que se obtuvo en la sincronización de símbolo.
8) Eliminación del prefijo cíclico (PC). – Aquí se elimina las muestras que se agregaron en la transmisión al inicio de cada símbolo OFDM.
9) FFT (Transformada Rápida de Fourier). – En esta etapa se pasa los símbolos OFDM del dominio del tiempo al dominio de la frecuencia.
10) Estimación de canal y ecualizador. – En esta etapa cada símbolo OFDM realiza la estimación del canal a través de las portadoras piloto.
11) Recuperación de datos del símbolo OFDM. – Etapa en donde se realiza el desamblaje de los símbolos OFDM, esto quiere decir que se eliminan las portadoras de guarda, las piloto y la DC, y se mantiene solo las portadoras con los datos.
12) Demapeo. – En esta etapa se pasa por los esquemas de modulación BPSK, QPSK, 16-QAM o 64-QAM a bits serializados.
11) Decodificador– Etapa que se utiliza para la corrección de errores a nivel de bit (Convolucional y Turbo códigos) y a nivel de byte (Reed - Solomon).
13) Medición del BER. – En esta etapa se compara los bits serializados recibidos con los bits serializados del archivo original.
14) Recuperar texto. – Esta etapa se encarga de recuperar el texto inicial que se transmitió.

2. METODOLOGÍA

El presente proyecto es una investigación de tipo experimental mixta (cualitativa y cuantitativa), es decir que nosotros realizamos un proceso deductivo en el cual realizamos la recolección de datos numéricos y comparamos entre sí para conocer cuál de todos ellos es la mejor opción para realizar una simulación y transmisión en tiempo real de un texto. Se realiza también un proceso inductivo que mediante nuestra investigación se da a conocer que dos de las tres técnicas se manejan a nivel de bit y la otra a nivel de byte y

por lo que necesitamos saber la mejor opción al momento de realizar una transmisión en tiempo real de un texto.

Para la recolección y comparación de datos lo que realizamos primero es un script en Matlab para cada técnica en el cual primero cargamos nuestro archivo de texto, después lo transformamos a binario para poder aplicar el codificador que compete a cada técnica de error en los cuales tendremos varias opciones de Mapeo, los cuales son: BPSK, QPSK, 16-QAM y 64-QAM. El siguiente paso ya es realizar nuestro Mapeo para consiguiente añadir ruido y poder darnos cuenta que tanto afecta en nuestra simulación.

Para la decodificación el primer paso es realizar el Demapeo, en el cual nos manejamos con nuestros Datos Mapeados para que luego pasen por un decodificador dependiendo de la técnica de error que se esté utilizando para después realizar la medición del BER y luego realizar la recuperación del texto.

Para tener más información contundente y poder sacar las distintas conclusiones nos manejamos con la medición del BER y a parte también la cantidad de letras erróneas de nuestro texto original.

Después de haber realizado todos los requerimientos para que nuestras simulaciones funcionen correctamente en Matlab, para la segunda parte se aplica la transmisión y recepción en tiempo real de nuestro texto, en el cual se necesitará dos SDR Adalm Pluto con sus respectivas antenas, se manejará dos laptops, una para la enviar la información y la otra para recibir la misma.

2.1. SIMULACIÓN

En nuestras simulaciones se va a transmitir y recibir el siguiente texto llamado *ArchivoTextoRT3.txt.*

> Este texto se va a transmitir y recibir utilizando dispositivos SDR Adalm Pluto con Matlab 2021a.
> 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
> a e i o u á é í ó ú ñññññ(x5) [] {} () ** ¿? ¡!
> Fragmento de texto de la síntesis histórica de la Escuela Politécnica Nacional.
>
> "La Escuela Politécnica Nacional fue fundada el 30 de agosto de 1869 por el Presidente García Moreno, quien contó con
> el apoyo del Superior de la Compañía de Jesús. La naciente Politécnica fue concebida como el primer centro de docencia
> e investigación científica, como órgano integrador del país y como ente generador del desarrollo nacional."

2.1.1. CONVOLUCIONAL

Para realizar esta simulación nos vamos a manejar con un solo script en el cual tendrá los siguientes procesos: Transmisor en donde consta: Datos de entrada (Abrir un archivo .txt), selección de Mapeo, codificador convolucional, Mapeo y Añadir ruido AWGN, después tenemos el Receptor en donde encontramos el Demapeo de datos, decodificador convolucional, medición del BER y recuperación del texto.

El archivo de texto tiene el nombre de *ArchivoTextoRT3.txt*

Para elegir nuestro Mapeo tenemos las siguientes opciones: BPSK, QPSK, 16-QAM y 64-QAM y la asignación de valores esta dado de la siguiente forma: 1 para BPSK, 2 para QPSK, 3 para 16-QAM y 4 para 64-QAM esto lo escogemos en la siguiente línea de código que se muestra a continuación.

```
seleccionMapeo=1;
```

El código se llama *Simulacion_Convolucional.m* y se encuentra en el Anexo I

Hay que mencionar que se maneja un ruido SNR igual a 6 dB, lo cual podemos verlo en la siguiente sentencia:

```
datosMapeo = awgn(datosMapeados,6,'measured'); % Señal con ruido
```

También para realizar la comparación de las tres técnicas se debe manejar el mismo tipo de mapeo, escogimos el mapeo 16-QAM, porque al realizar pruebas con BPSK y QPSK no se diferencia mucho los errores y es preferible mostrar diferencias cuando se realiza la simulación.

Después de simular nuestro programa obtenemos la siguiente respuesta en nuestro Workspace que se muestra en la Figura 2.1

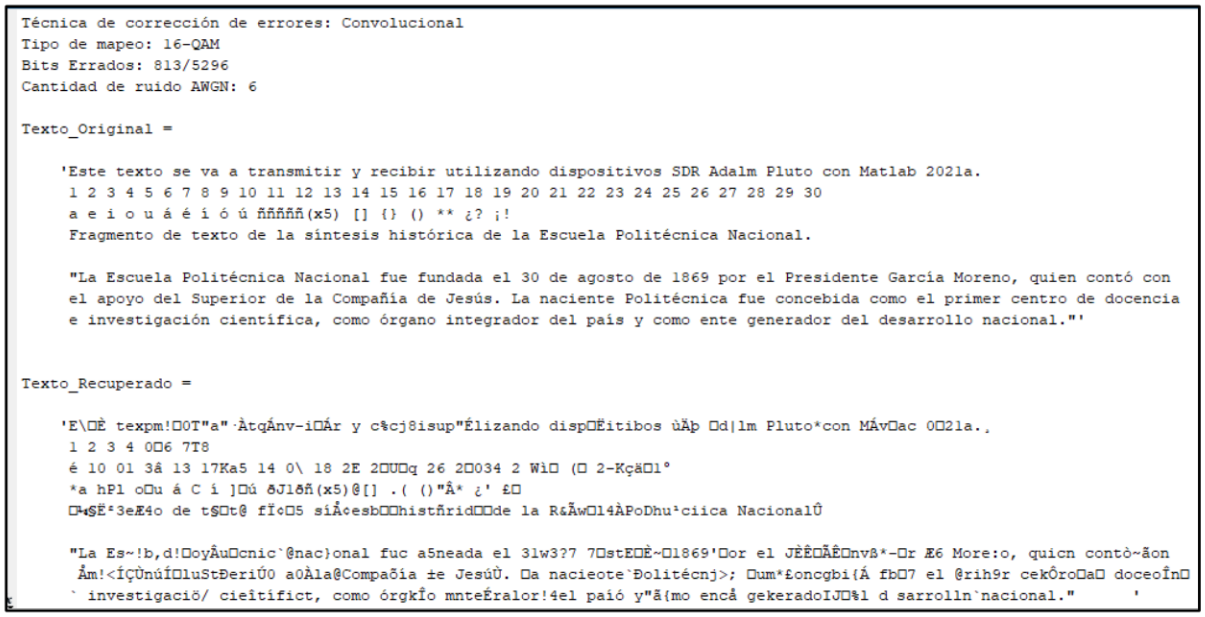


Figura 2.1 Primera prueba: Bits Errados, texto original y recuperado con Mapeo 16-QAM con la técnica de errores Convolucional.

Al momento de receptar la información se obtiene cierta cantidad de bits errados, como muestra la Figura 2.1 se obtuvo 813, este valor no siempre se lo va a obtener para demostrar esto se realizó dos pruebas más.

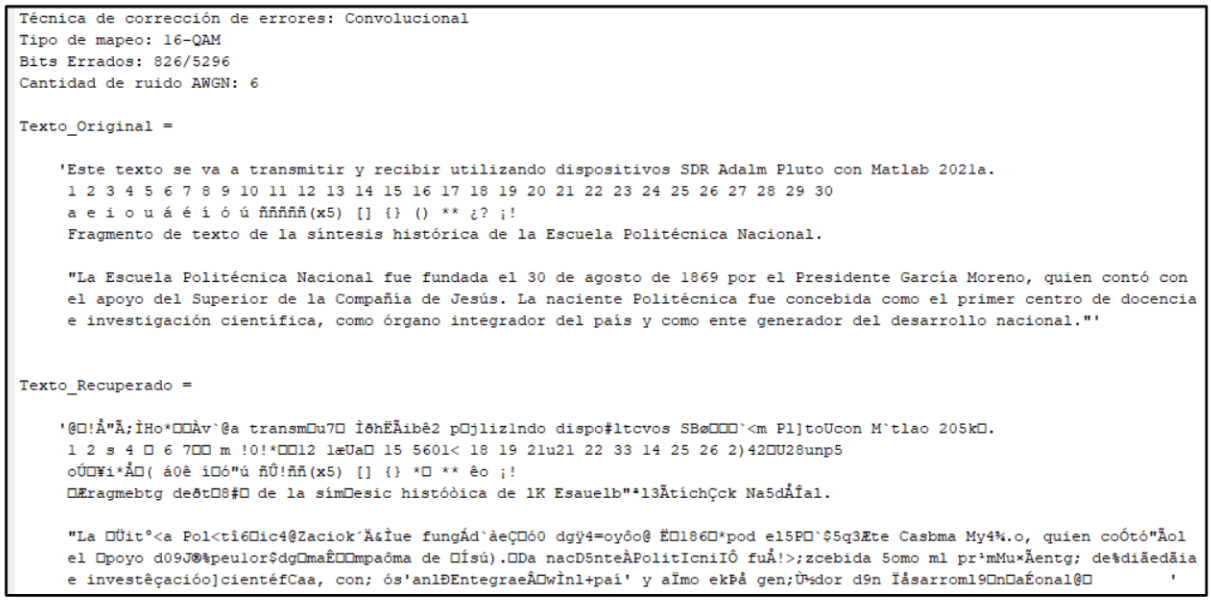


Figura 2.2 Segunda prueba: Bits Errados, texto original y recuperado con Mapeo 16-QAM con la técnica de errores Convolucional.

Como se muestra en la Figura 2.2 se obtuvo 826 bits errados, un valor distinto, pero no muy lejano de la prueba 1 que se muestra en la Figura 2.1

Se realiza una tercera simulación para ver qué valor de bits errados nos da.

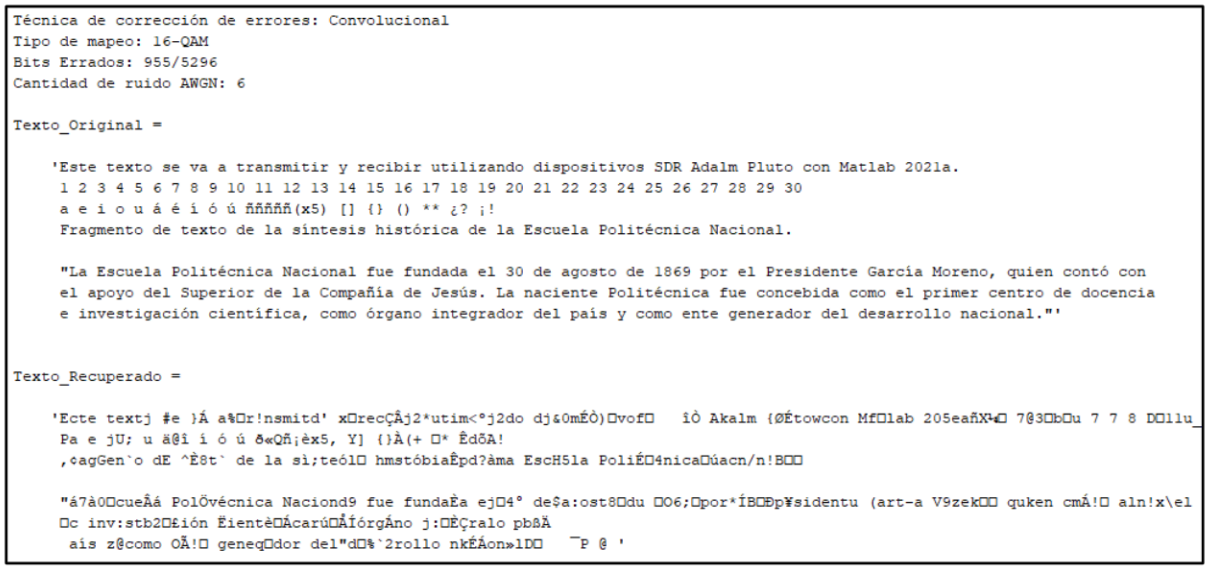


Figura 2.3 Tercera prueba: Bits Errados, texto original y recuperado con Mapeo 16-QAM con la técnica de errores Convolucional.

Como se muestra en la Figura 2.3 se obtuvo un valor de 955 bits errados, aquí se comprueba que los valores obtenidos no siempre están en un rango fijo.

2.1.2. REED - SOLOMON

Para realizar nuestra simulación con la técnica de errores Reed – Solomon nos basamos en si con pasos similares con la anterior técnica Convolucional.

En nuestro Transmisor abrimos el archivo de texto *ArchivoTextoRT3.txt* y de ahí se realizan los siguientes pasos: selección de Mapeo, codificador Reed - Solomon, Mapeo y Añadir ruido AWGN, después tenemos el Receptor el Demapeo de datos, decodificador Reed - Solomon, medición del BER y recuperación del texto.

Nuestro tipo de Mapeo es el de 16-QAM e igual añadimos un valor de SNR igual a 6 dB para poder después realizar comparaciones, como dato adicional cuando ejecutamos el programa se demora un lapso de 13 segundos en obtener resultados, esto depende principalmente por la sentencia *rsenc,* la cual se utiliza para la codificación Reed – Solomon.

El código se llama *Simulacion_ReedSolomon.m* y se encuentra en el Anexo II.

Después de simular nuestro programa obtenemos la siguiente respuesta en nuestro Workspace que se muestra en la Figura 2.4.

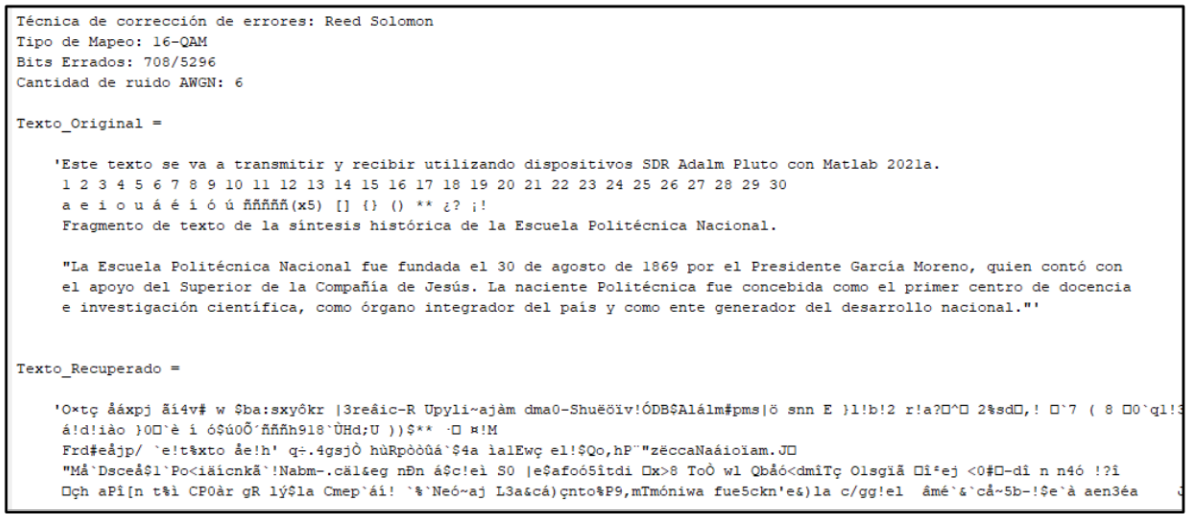


Figura 2.4 Primera prueba: Bits Errados, texto original y recuperado con Mapeo 16-QAM con la técnica de errores Reed – Solomon.

Como en la parte de convolución, aquí también se realiza dos pruebas más para ver cuanto varían los bits errados.

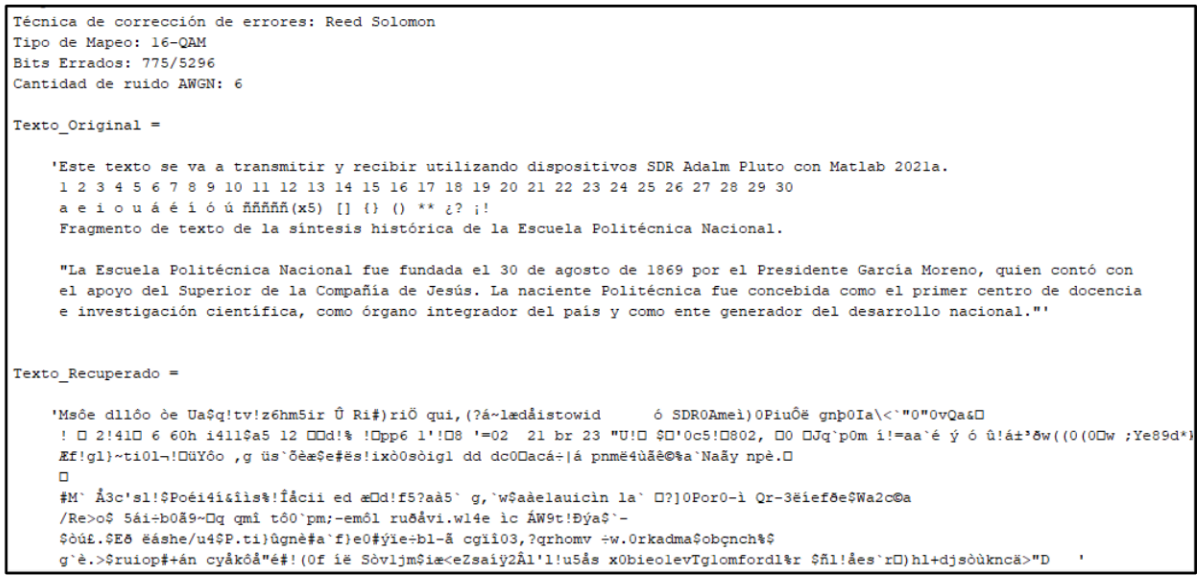


Figura 2.5 Segunda prueba: Bits Errados, texto original y recuperado con Mapeo 16-QAM con la técnica de errores Reed – Solomon.

Como se muestra en la Figura 2.5 se obtiene un valor de 775 bits errados, distinto al valor de la prueba, pero igual no está muy lejano.

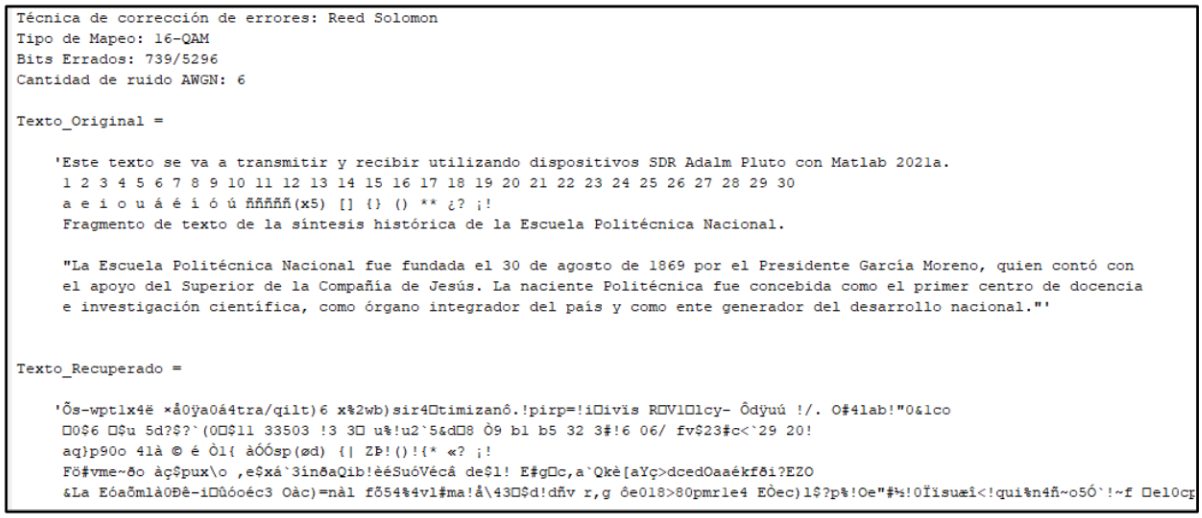


Figura 2.6 Tercera prueba: Bits Errados, texto original y recuperado con Mapeo 16-QAM con la técnica de errores Reed – Solomon.

En la Figura 2.6 se obtuvo el valor de 739 bits errados, en comparación con las otras dos pruebas se mantiene en los valores de 700. Estos resultados tendrán sus respectivas conclusiones.

2.1.3. TURBO CÓDIGOS

Esta técnica también se asimila a las dos anteriores y cumple con el siguiente proceso:

En nuestro Transmisor abrimos el archivo de texto *ArchivoTextoRT3.txt* y de ahí se realizan los siguientes pasos: codificador Turbo código, Mapeo y Añadir ruido AWGN, después tenemos el Receptor en donde realizamos el Demapeo de datos, decodificador Turbo código, medición del BER y recuperación del texto.

Nuestro tipo de Mapeo es el de 16-QAM e igual añadimos un valor de SNR igual a 6 dB para poder después realizar comparaciones

El código se llama *Simulacion_Turbocodigo.m* y se encuentra en el Anexo III.

Después de simular nuestro programa obtenemos la siguiente respuesta en nuestro Workspace que se muestra en la Figura 2.7.

```
Técnica de corrección de errores: Turbo código
Tipo de Mapeo: 16-QAM
Bits Errados: 0/5296
Cantidad de ruido AWGN: 6

Texto_Original =

    'Este texto se va a transmitir y recibir utilizando dispositivos SDR Adalm Pluto con Matlab 2021a.
     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
     a e i o u á é í ó ú ñññññ(x5) [] {} () ** ¿? ¡!
     Fragmento de texto de la síntesis histórica de la Escuela Politécnica Nacional.

     "La Escuela Politécnica Nacional fue fundada el 30 de agosto de 1869 por el Presidente García Moreno, quien contó con
     el apoyo del Superior de la Compañía de Jesús. La naciente Politécnica fue concebida como el primer centro de docencia
     e investigación científica, como órgano integrador del país y como ente generador del desarrollo nacional."'


Texto_Recuperado =

    'Este texto se va a transmitir y recibir utilizando dispositivos SDR Adalm Pluto con Matlab 2021a.
     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
     a e i o u á é í ó ú ñññññ(x5) [] {} () ** ¿? ¡!
     Fragmento de texto de la síntesis histórica de la Escuela Politécnica Nacional.

     "La Escuela Politécnica Nacional fue fundada el 30 de agosto de 1869 por el Presidente García Moreno, quien contó con
     el apoyo del Superior de la Compañía de Jesús. La naciente Politécnica fue concebida como el primer centro de docencia
     e investigación científica, como órgano integrador del país y como ente generador del desarrollo nacional."'
```

Figura 2.7 Primera prueba: Bits Errados, texto original y recuperado con Mapeo 16-QAM con la técnica de errores Turbo código.

En este caso se obtiene un valor de 0 bits errados, como en las dos anteriores técnicas se realizará dos pruebas más para ver cuanto varia nuestro resultado.

```
Técnica de corrección de errores: Turbo código
Tipo de Mapeo: 16-QAM
Bits Errados: 6/5296
Cantidad de ruido AWGN: 6

Texto_Original =

    'Este texto se va a transmitir y recibir utilizando dispositivos SDR Adalm Pluto con Matlab 2021a.
     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
     a e i o u á é í ó ú ñññññ(x5) [] {} () ** ¿? ¡!
     Fragmento de texto de la síntesis histórica de la Escuela Politécnica Nacional.

     "La Escuela Politécnica Nacional fue fundada el 30 de agosto de 1869 por el Presidente García Moreno, quien contó con
     el apoyo del Superior de la Compañía de Jesús. La naciente Politécnica fue concebida como el primer centro de docencia
     e investigación científica, como órgano integrador del país y como ente generador del desarrollo nacional."'


Texto_Recuperado =

    'Este texto se va a transmitir y recibir utilizando dispositivos SDR Adalm Pluto con Matlab 2021a.
     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
     a e i o u á é í ó ú ñññññ(x5) [] {} () ** ¿? ¡!
     Fragmento de texto de la síntesis histórica de la Escuela Politécnica Nacional.

     "La Escuela Politécnica Nacional fue fundada el 30 de agosto de 1869 por el Presidente García Moreno, quien contó con
     el apoyo del Superior de la Compañía de 4esús. La naciente Politécnica fue concebida como el primer centro de docencia
     e investigación científica, como órgano integrador del país y como ente generador del desarrollo nacional."'
```

Figura 2.8 Segunda prueba: Bits Errados, texto original y recuperado con Mapeo 16-QAM con la técnica de errores Turbo código.

Como se observa en la Figura 2.8 se obtuvo un valor de 6 bits errados, distinto a la de la prueba uno, pero los dos valores se encuentran demasiado cerca.

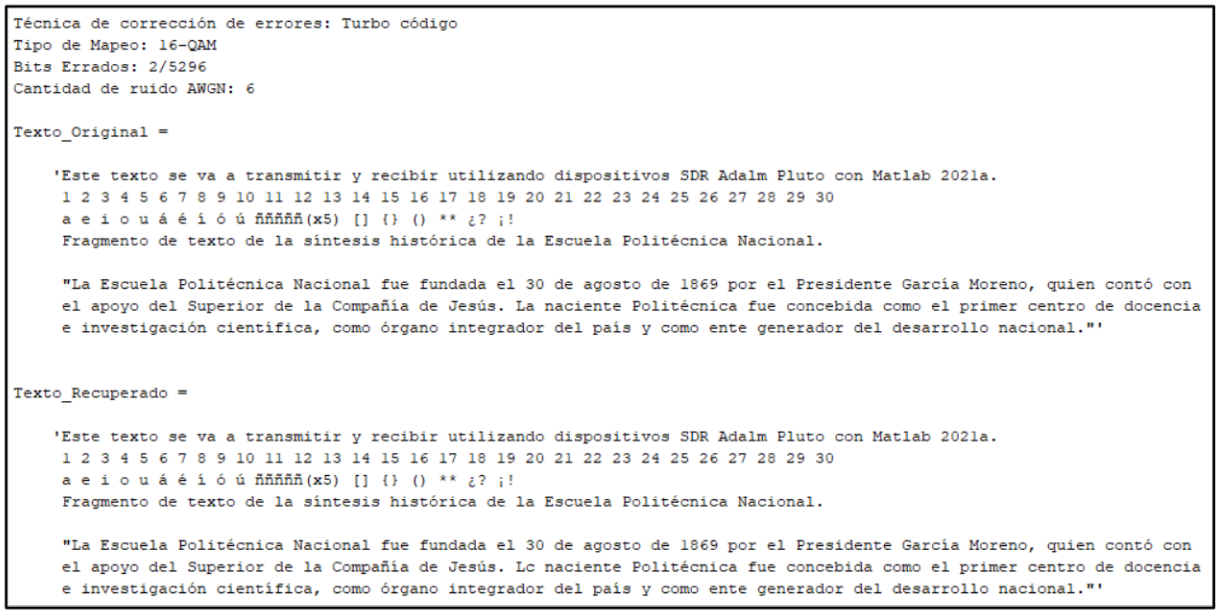


Figura 2.9 Tercera prueba: Bits Errados, texto original y recuperado con Mapeo 16-QAM con la técnica de errores Turbo código.

Como se observa en la Figura 2.9 se obtuvo un valor de 2 bits errados, no es idéntico a las otras dos pruebas, pero los valores están demasiado cerca. Estos resultados tendrán sus respectivas conclusiones.

2.2. IMPLEMENTACIÓN PRÁCTICA EN BASE A TECNOLOGÍA SDR

En esta sección se va a realizar con dos scripts uno para la transmisión y otro para la recepción utilizando dos SDR Adalm Pluto con sus respectivas antenas y dos laptops. Es preferible realizar este proceso a distancia considerable, porque si se encuentran cerca no vamos a poder comparar las diferentes técnicas, ya que obtendremos un valor de bits errados igual a 0.

Para la realización de esta implementación práctica primero tenemos que ejecutar el programa *configurarAdalmPluto.m* que se encuentra en el Anexo IV, el cual contiene todas las configuraciones para la transmisión y recepción. Dichos parámetros se muestran a continuación:

- Ganancia de transmisión igual a -2 dB.
- Ganancia de recepción igual a 20 dB.
- Tamaño del bloque de datos igual a 800000.
- Frecuencia central igual a 860MHz.
- Frecuencia de muestreo en 2MHz.

Para realizar una comparación se debe estar a una misma distancia en este caso 3 metros, se debe utilizar el mismo tipo de Mapeo, el cual será 16-QAM y ejecutar el mismo programa que es ArchivoTextoRT3.txt. Todo esto para poder sacar conclusiones sobre las distintas técnicas de corrección de errores.

Cabe recalcar que se escoge el mapeo de 16-QAM, esto porque en los mapeos de BPSK y QPSK no es obtuvo mucha diferencia para poder sacar conclusiones.

2.2.1. CONVOLUCIONAL

Los nombres de los programas a ejecutar son: *transmisor_Convolucional.m* que se encuentra en el Anexo V y *receptor_Convolucional.m* que se encuentran en el Anexo VI.

Obteniendo el siguiente resultado en la Figura 2.10

```
Corrección de errores implementado
Medición del BER:
Bits errados/Bits totales = 82/5296
BER = 0.0154834

Texto_Original =

    'Este texto se va a transmitir y recibir utilizando dispositivos SDR Adalm Pluto con Matlab 2021a.
     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
     a e i o u á é í ó ú ñññññ(x5) [] {} () ** ¿? ¡!
     Fragmento de texto de la síntesis histórica de la Escuela Politécnica Nacional.

     "La Escuela Politécnica Nacional fue fundada el 30 de agosto de 1869 por el Presidente García Moreno, quien contó con
     el apoyo del Superior de la Compañía de Jesús. La naciente Politécnica fue concebida como el primer centro de docencia
     e investigación científica, como órgano integrador del país y como ente generador del desarrollo nacional."'

Texto recuperado:

Texto_Recuperado =

    'Mste texto se ~a a transmitir y reĚébir utilizando dispositivos SDR Adalm Plutmúcon Matlab 2021a.
     □ 2 3 4 5 3 7 8 9 10 11 12 13 14 15 16 17 18 19 20"21 22 23 24 25 26 27 28 29 30!
     a e i nuu á é í óuú ñññññ(x5) [] {} () ** ¿? ¡!
     Fragmento□de texto%de la sínqesis histórica de la EscÊ%la Rolitécnica Nacional.

     "La Escuela Politécnica Nacional fue fundada gl 30 de a}osto de 1869 por el!Presidente García Moreno, quien coÎtó con
     eD ap□yo del Superior de la Compañía de Jesús/ La naciente Politécnica fue cĬncebida aomo el primer centro de docenaia
     e invmstigacióc científica, como órgano intmgrador del país y como en$e gen`2ador del desarrol□o naÃional."'
```

Figura 2.10 Primera prueba: Texto Recuperado con Mapeo 16-QAM a una distancia de 3 metros con técnica de errores Convolucional.

Se va a realizar dos pruebas más para ver que valores de bits errados obtenemos, si nos da un valor similar o en qué valor vario.

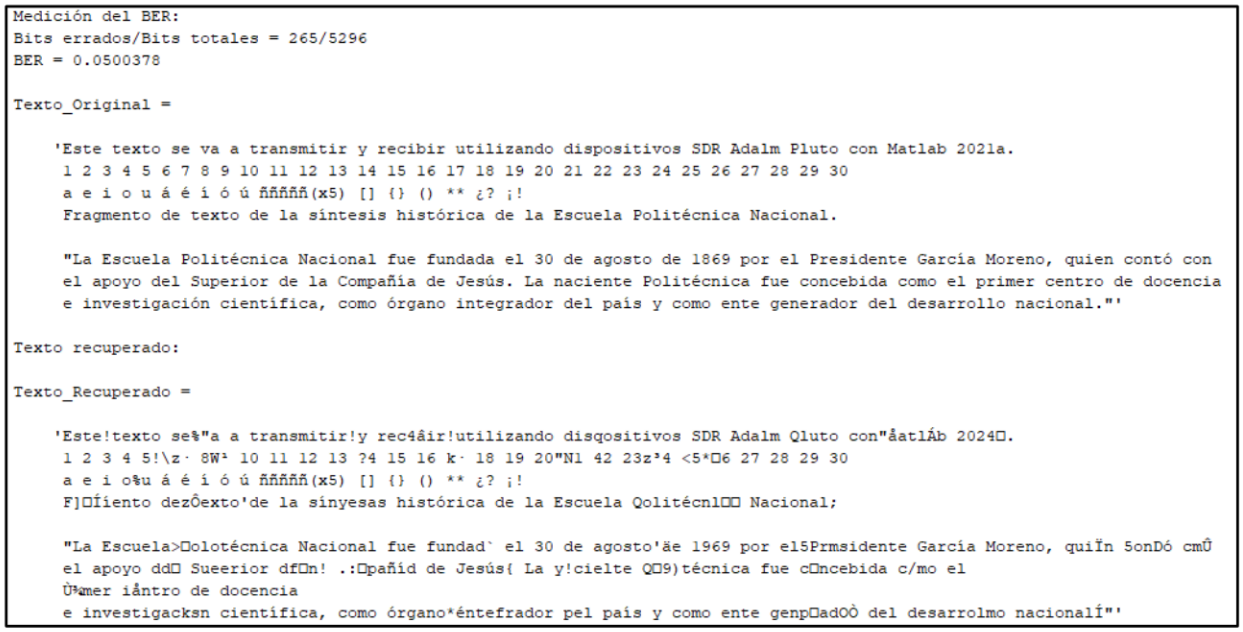


Figura 2.11 Segunda prueba: Texto Recuperado con Mapeo 16-QAM a una distancia de 3 metros con técnica de errores Convolucional.

Como se observa en la Figura 2.11 se obtuvo un valor de 265 bits errados, un valor distinto comparado con la primera prueba que fue un valor de 82. La diferencia entre la primera y segunda prueba si es un valor considerable, por eso es necesario la tercera prueba para ver qué valor se obtiene.

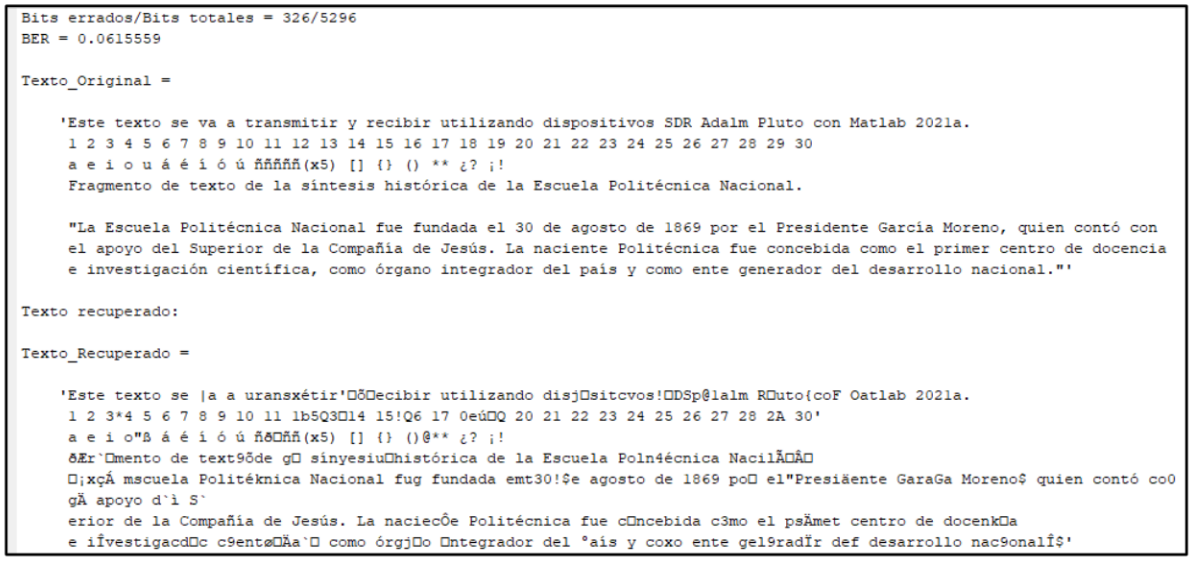


Figura 2.12 Tercera prueba: Texto Recuperado con Mapeo 16-QAM a una distancia de 3 metros con técnica de errores Convolucional.

Como se observa en la Figura 2.12 se obtuvo un valor de 326 bits errados, en comparación con las dos anteriores pruebas no se asemeja a ninguno y es más distante comparado con la prueba 1.

2.2.2. REED – SOLOMON

Los nombres de los programas a ejecutar son: *transmisor_ReedSolomon.m* que se encuentra en el Anexo VII y *receptor_ReedSolomon.m* que se encuentran en el Anexo VIII.

Obteniendo el siguiente resultado en la Figura 2.13

```
Corrección de errores implementado
Medición del BER:
Bits errados/Bits totales = 209/5296
BER = 0.0394637

Texto_Original =

    'Este texto se va a transmitir y recibir utilizando dispositivos SDR Adalm Pluto con Matlab 2021a.
     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
     a e i o u á é í ó ú ñññññ(x5) [] {} () ** ¿? ¡!
     Fragmento de texto de la síntesis histórica de la Escuela Politécnica Nacional.

     "La Escuela Politécnica Nacional fue fundada el 30 de agosto de 1869 por el Presidente García Moreno, quien contó con
     el apoyo del Superior de la Compañía de Jesús. La naciente Politécnica fue concebida como el primer centro de docencia
     e investigación científica, como órgano integrador del país y como ente generador del desarrollo nacional."'

Texto recuperado:

Texto_Recuperado =

    '□bpe texto$qe!va a`trajsìidir }avecibhv uuiliz`.do lispmsitivos SDV`□dalm P,eto con M`pl`b`r121a.
     1 2 3 4 5 6 7 8 9 0□ 11 12 13 □4415 16 q7$08 q9`20 3□$□□$23 24 25 2□027 28 3□ 1□ I
     a e i o u á é i ó ú ñññññ(x5) [I {} (+ **`ÿ? ¡1
     Fragmen|/ de texto de la síntesiq$èistówyca$le la Ewcueh  Pol)\écnic`eNccional.
     □]□"La Essuelq`Polmtéc.iaa Naci f`(0"we fundada el 30 de a'/qdo de 1869 por el P2$3)dáìpa Garcí  Moreno, qU)%n cont÷ gon ]
     el apoyo del Ru`erior dg la$Comp`ñía de Jesú3> □c nacientmapëlitécnica0fue concebiäc #om `el `rimer geo|rg0$- docenc(a
     e inver4igaciónaâèentíìica- col/ órg`n/ahêpågrador àål país y com/ %npã`ge.erador!àel desa2r/llo nag}onal."'
```

Figura 2.13 Primera prueba: Texto Recuperado con Mapeo 16-QAM a una distancia de 3 metros con técnica de errores Reed - Solomon.

En este caso como en el de convolucional igualmente se realizará dos pruebas más para ver que valores se obtiene.

```
Medición del BER:
Bits errados/Bits totales = 135/5296
BER = 0.0254909

Texto_Original =

    'Este texto se va a transmitir y recibir utilizando dispositivos SDR Adalm Pluto con Matlab 2021a.
     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
     a e i o u á é í ó ú ñññññ(x5) [] {} () ** ¿? ¡!
     Fragmento de texto de la síntesis histórica de la Escuela Politécnica Nacional.

     "La Escuela Politécnica Nacional fue fundada el 30 de agosto de 1869 por el Presidente García Moreno, quien contó con
     el apoyo del Superior de la Compañía de Jesús. La naciente Politécnica fue concebida como el primer centro de docencia
     e investigación científica, como órgano integrador del país y como ente generador del desarrollo nacional."'

Texto recuperado:

Texto_Recuperado =

    'Este texto se va a trans}itir y recibir utilizando dispositivos SDR Adal,□Pluto con Matlaf 2021a.
     1 64304 5 6 7 8 9 10 cs 12 57t14 55 16 u7418 giar4 s14330vb`24 25 26 27 28 29 30 □□a e y0o u á é í ó0þ ñññññ(x5) [] {ý$()
     Fraw}ento0le texto de la síntes)q histósyca de la Escuela Politécnica Nacio~ql.
     M□"Ha4Åsauela Poli|icnica Nacional fue fundadá el 30 de4qgosto de 1869 por el Tsusidmnte García Moreno, quien contó con
     el apoyo del Sutusinr de la Á/mp`ñia de Jesús. La naciente`politécnica`nue c?v!$b)da com 4el primer suntro de0docencia0M
     e investigacióo científæica,4s mo órgano integ2a$/r del país y como ente generador `el desarroìho nasiojad."'
```

Figura 2.14 Segunda prueba: Texto Recuperado con Mapeo 16-QAM a una distancia de 3 metros con técnica de errores Reed - Solomon.

Como se observa la Figura 2.14 se obtuvo un valor de 135 bits errados, en comparación con la primera prueba. Por eso se realiza una tercera prueba.

```
Medición del BER:
Bits errados/Bits totales = 54/5296
BER = 0.0101964

Texto_Original =

    'Este texto se va a transmitir y recibir utilizando dispositivos SDR Adalm Pluto con Matlab 2021a.
     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
     a e i o u á é í ó ú ñññññ(x5) [] {} () ** ¿? ¡!
     Fragmento de texto de la síntesis histórica de la Escuela Politécnica Nacional.

     "La Escuela Politécnica Nacional fue fundada el 30 de agosto de 1869 por el Presidente García Moreno, quien contó con
     el apoyo del Superior de la Compañía de Jesús. La naciente Politécnica fue concebida como el primer centro de docencia
     e investigación científica, como órgano integrador del país y como ente generador del desarrollo nacional."'

Texto recuperado:

Texto_Recuperado =

    'Este texto se va a transmitir0y rebibir utilizando dispositivos SDR Adalm Pluto con Matlab 2021a.
     1 2 3 4 5 6 ' 8 9 10 11 12 13 04 3% 16 17 !8 ') 60 21 22 23 24$25 2□ 2' 28 29 30
     a e i o u á é í ó ú ðõñññ(x5) [] {} () *>0¿?  !
     Fragmento de texto de la síntesis histórica de la Escuel` Politécnic! Nacional.
     □□"La Escuela Politécnica Nácional fue fundada el 30 de agost/ de 5 □9 por el Preri`ente Garcí! Mor$.o, qu)%n contó con
     el apoyo del Superior de la Compañía de Jesúq& La naciente Politécnica fue c ncdb)da com 0el primer auftro de docencia
     e investigación4sientífica, com/ órgano integrador del país y como ente generador deì desarrollo nacional."'
```

Figura 2.15 Tercera prueba: Texto Recuperado con Mapeo 16-QAM a una distancia de 3 metros con técnica de errores Reed - Solomon.

Como se observa en la Figura 2.15 se obtuvo un valor de 54 bits errados, un valor bien distinto comparado con la segunda prueba y con mayos diferencia comparado con la primera prueba.

2.2.3. TURBO CÓDIGOS

Los nombres de los programas a ejecutar son: *transmisor_Turbocodigo.m* que se encuentra en el ANEXO IX y *receptor_Turbocodigo.m* que se encuentran en el Anexo X.

Obteniendo el siguiente resultado en la Figura 2.16

```
Corrección de errores implementado
Medición del BER:
Bits errados/Bits totales = 0/5296
BER = 0

Texto_Original =

    'Este texto se va a transmitir y recibir utilizando dispositivos SDR Adalm Pluto con Matlab 2021a.
     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
     a e i o u á é í ó ú ñññññ(x5) [] {} () ** ¿? ¡!
     Fragmento de texto de la síntesis histórica de la Escuela Politécnica Nacional.

     "La Escuela Politécnica Nacional fue fundada el 30 de agosto de 1869 por el Presidente García Moreno, quien contó con
     el apoyo del Superior de la Compañía de Jesús. La naciente Politécnica fue concebida como el primer centro de docencia
     e investigación científica, como órgano integrador del país y como ente generador del desarrollo nacional."'

Texto recuperado:

Texto_Recuperado =

    'Este texto se va a transmitir y recibir utilizando dispositivos SDR Adalm Pluto con Matlab 2021a.
     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
     a e i o u á é í ó ú ñññññ(x5) [] {} () ** ¿? ¡!
     Fragmento de texto de la síntesis histórica de la Escuela Politécnica Nacional.

     "La Escuela Politécnica Nacional fue fundada el 30 de agosto de 1869 por el Presidente García Moreno, quien contó con
     el apoyo del Superior de la Compañía de Jesús. La naciente Politécnica fue concebida como el primer centro de docencia
     e investigación científica, como órgano integrador del país y como ente generador del desarrollo nacional."'
```

Figura 2.16 Primera prueba: Texto Recuperado con Mapeo 16-QAM a una distancia de 3 metros con técnica de errores Turbo códigos.

Se realizará dos pruebas más como en las anteriores técnicas para ver qué tanto de diferencia se obtiene la una de la otra.

```
Bits errados/Bits totales = 0/5296
BER = 0

Texto_Original =

    'Este texto se va a transmitir y recibir utilizando dispositivos SDR Adalm Pluto con Matlab 2021a.
     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
     a e i o u á é i ó ú ñññññ(x5) [] {} () ** ¿? ¡!
     Fragmento de texto de la síntesis histórica de la Escuela Politécnica Nacional.

     "La Escuela Politécnica Nacional fue fundada el 30 de agosto de 1869 por el Presidente García Moreno, quien contó con
     el apoyo del Superior de la Compañía de Jesús. La naciente Politécnica fue concebida como el primer centro de docencia
     e investigación científica, como órgano integrador del país y como ente generador del desarrollo nacional."'

Texto recuperado:

Texto_Recuperado =

    'Este texto se va a transmitir y recibir utilizando dispositivos SDR Adalm Pluto con Matlab 2021a.
     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
     a e i o u á é i ó ú ñññññ(x5) [] {} () ** ¿? ¡!
     Fragmento de texto de la síntesis histórica de la Escuela Politécnica Nacional.

     "La Escuela Politécnica Nacional fue fundada el 30 de agosto de 1869 por el Presidente García Moreno, quien contó con
     el apoyo del Superior de la Compañía de Jesús. La naciente Politécnica fue concebida como el primer centro de docencia
     e investigación científica, como órgano integrador del país y como ente generador del desarrollo nacional."'
```

Figura 2.17 Segunda prueba: Texto Recuperado con Mapeo 16-QAM a una distancia de 3 metros con técnica de errores Turbo códigos.

En la Figura 2.17 se observa que se obtuvo un valor de 0 bits errados, el mismo valor que en la primera prueba. Se realizo una tercera prueba en donde se obtuvo el mismo resultado. Es la primera técnica que se obtiene los mismos valores.

3. RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

3.1. RESULTADOS

En esta parte nos basamos en 4 distancias entre transmisor y receptor y obtenemos los siguientes resultados para las distintas técnicas de corrección de errores.

3.1.1. CONVOLUCIONAL

Ya que el canal es cambiante en el tiempo se ha tomado 6 mediciones del número de errores cometidos considerando el número total de bits enviados (5296).

Cabe recalcar que cuando se realizó la transmisión y recepción del tipo de Mapeo 64QAM a una distancia de 18 metros, no se obtuvo una respuesta por lo que se decidió solo en esa prueba cambiar los siguientes parámetros:

- Ganancia de recepción igual a 30 dB (antes estaba en 20 dB).
- Umbral de recepción igual a 0.0030 (antes estaba en 0.0040).

A continuación, se muestra todas las medidas de bits errados en todas las distancias:

- 18m
  - BPSK (0, 2, 9, 4, 19, 5)
  - QPSK (7, 11, 3, 9, 10, 5)
  - 16QAM (827, 609, 361, 717, 643, 524)
  - 64QAM (783, 844, 1706, 1200, 1466, 1993)
- 13 m
  - BPSK (9, 1, 3, 4, 2, 4)
  - QPSK (13, 3, 0, 1, 2, 9)
  - 16QAM (446, 397, 516, 463, 438, 486)
  - 64QAM (1063, 713, 788, 1254, 1005, 1245)
- 9m
  - BPSK (9, 0, 0, 0, 0, 0)
  - QPSK (0, 1, 6, 0, 1, 0)
  - 16QAM (44, 66, 77, 35, 107, 128)
  - 64QAM (823, 379, 715, 537, 695, 467)
- 5m
  - BPSK (1, 0, 0, 0, 0, 0)
  - QPSK (1, 7, 0, 0, 0, 1)
  - 16QAM (452, 704, 5, 286, 62, 70)
  - 64QAM (587, 1156, 1180, 1824, 666, 856)

El promedio de cada caso se muestra en la Tabla 3.1.

Tabla 3.1 Tabla promedio del BER en valor y fracción de la técnica de errores Convolucional.

CONVOLUCIONAL			
Esquema de Mapeo	**Distancia en metros**	**BER**	
		Valor	**Fracción**
BPSK	18	0.000122	6.5/5296
	13	0.000722	3.83/5296
	9	0.000283	1.5/5296
	5	0.000031	0.1667/5296
QPSK	18	0.001699	9/5296
	13	0.000881	4.667/5296
	9	0.000251	1.33/5296
	5	0.000283	1.5/5296
16-QAM	18	0.11584	613.5/5296
	13	0.08641	457.667/5296
	9	0.01438	76.1667/5296
	5	0.04969	263.1667/5296

64-QAM	18	0.25151	1332/5296
	13	0.19096	1011.33/5296
	9	0.11379	602.667/5296
	5	0.19728	1044.83/5296

3.1.2. REED - SOLOMON.

En este caso también se realizó la toma de 6 mediciones del número de errores cometidos considerando el número total de bits enviados (5296).

A continuación, se muestra todas las medidas de bits errados en todas las distancias:

- 18m
  - BPSK (3, 0, 0, 0, 0,1)
  - QPSK (0, 1, 2, 0, 0, 1)
  - 16QAM (386, 437, 343, 588, 484, 523)
  - 64QAM (568, 708, 791, 946, 868, 659)
- 13 m
  - BPSK (0, 0, 0, 1, 0, 0)
  - QPSK (0, 0, 0, 0, 0, 0)
  - 16QAM (287, 148, 478, 215, 197, 357)
  - 64QAM (539, 969, 682, 565, 711, 752)
- 9m
  - BPSK (3, 0, 0, 0, 11, 2)
  - QPSK (0, 0, 0, 0, 0, 0)
  - 16QAM (171, 291, 122, 312, 15, 88)
  - 64QAM (529, 253, 560, 626, 709, 528)
- 5m
  - BPSK (0, 0, 0, 0, 0, 0)
  - QPSK (0, 0, 0, 0, 0, 0)
  - 16QAM (167, 78, 74, 68, 402, 66)
  - 64QAM (290, 581, 300, 388, 442, 499)

El promedio de cada caso se muestra en la Tabla 3.2.

Tabla 3.2 Tabla promedio del BER en valor y fracción de la técnica de errores Reed - Solomon.

REED - SOLOMON			
Esquema de Mapeo	**Distancia en metros**	**BER**	
		Valor	**Fracción**
BPSK	18	0.000125	0.667/5296
	13	0.000031	0.1667/5296
	9	0.000503	2.667/5296
	5	0	0/5296

QPSK	18	0.000125	0.667/5296
	13	0	0/5296
	9	0	0/5296
	5	0	0/5296
16-QAM	18	0.08688	460.167/5296
	13	0.05293	280.333/5296
	9	0.03143	166.5/5296
	5	0.02690	142.5/5296
64-QAM	18	0.14287	756.667/5296
	13	0.13274	703/5296
	9	0.10086	534.167/5296
	5	0.07867	416.667/5296

3.1.3. TURBO CÓDIGOS

Como en los dos casos anteriores se realiza 6 mediciones del número de errores cometidos considerando el número total de bits enviados (5296).

A continuación, se muestra todas las medidas de bits errados en todas las distancias:

- 18m
  - BPSK (0, 0, 0, 0, 0, 0)
  - QPSK (0, 0, 0, 0, 0, 0)
  - 16QAM (2, 0, 0, 0, 22, 11)
  - 64QAM (9, 5, 16, 1, 0, 23)
- 13 m
  - BPSK (0, 0, 0, 0, 0, 0)
  - QPSK (0, 0, 0, 0, 0, 0)
  - 16QAM (0, 0, 0, 0, 0, 0)
  - 64QAM (32, 14, 73, 49, 6, 2)
- 9m
  - BPSK (0, 0, 0, 0, 0, 0)
  - QPSK (0, 0, 0, 0, 0, 0)
  - 16QAM (0, 0, 0, 0, 0, 0)
  - 64QAM (5, 0, 53, 0, 12, 2)
- 5m
  - BPSK (0, 0, 0, 0, 0, 0)
  - QPSK (0, 0, 0, 0, 0, 0)
  - 16QAM (0, 0, 0, 0, 0, 0)
  - 64QAM (17, 0, 77, 79, 1, 0)

El promedio de cada caso se muestra en la Tabla 3.3

Tabla 3.3 Tabla promedio del BER en valor y fracción de la técnica de errores Turbo códigos.

TURBO CÓDIGOS			
Esquema de Mapeo	**Distancia en metros**	**BER**	
		Valor	**Fracción**
BPSK	18	0	0/5296
	13	0	0/5296
	9	0	0/5296
	5	0	0/5296
QPSK	18	0	0/5296
	13	0	0/5296
	9	0	0/5296
	5	0	0/5296
16-QAM	18	0.00110	5.833/5296
	13	0	0/5296
	9	0	0/5296
	5	0	0/5296
64-QAM	18	0.00169	9/5296
	13	0.00553	29.33/5296
	9	0.02265	12/5296
	5	0.00547	29/5296

3.2. CONCLUSIONES

En las simulaciones realizadas, en donde se maneja un mapeo de BPSK y se aplica un valor de SNR igual a 6 dB, se obtuvo que la mejor técnica de corrección de errores resulta ser la de Turbo códigos con un valor de 0 bits errados, esto es de esperarse ya que dicha técnica en comparación con las tres es la más actual y por eso su uso en diferentes aplicaciones como lo son la telefonía móvil LTE.

En las pruebas prácticas realizadas se obtuvo que los esquemas de modulación digital más robustos frente a errores son BPSK y QPSK. Los valores de bits errados se mantienen en cero o cercanos a cero para el caso de los Turbo códigos, como se observa en las Tablas 3.1, 3.2 y 3.3. Cuando se cambia el mapeo a 16QAM se puede notar en la Tabla 3.1 y Tabla 3.2 que los valores de bits errados suben significativamente excepto en la Tabla 3.3 que corresponde a la técnica de Turbo códigos, que prácticamente se tiene valores de 0. Por lo anterior, la mejor técnica de corrección de errores es la de Turbo códigos.

Es importante mencionar que en la Tabla 3.2, que pertenece a Reed – Solomon con un mapeo de 16 – QAM, se obtiene valores ascendentes partiendo desde 5 metros con un valor de 142.5/5296, un BER de 166.5/5296 a 9 metros con un valor, un BER de 280.333/5296 a 13 metros y un BER de 460/5296 a 18 metros, comparándola con la técnica convolucional, la primera es la mejor opción.

3.3. RECOMENDACIONES

Para poder realizar una excelente transmisión y recepción en tiempo real, lo que primero se tiene que hacer es una buena simulación, conociendo todas las etapas que están en el transmisor y de igual manera en el receptor. En dicha simulación se debe obtener valores coherentes, acorde con la parte teórica consultada. No se puede obtener valores de bits errados similares en las tres diferentes técnicas que realizamos. De obtener valores similares, nos quiere decir que estamos realizando una mala simulación. Esto va acorde también, que cuando se realiza la simulación es preferible ir probando todo lo que se va programando, como cada transmisor y receptor tienen sus respectivos bloques que van concatenados el uno con el otro, se debe ir probando cada uno de ellos.

Cuando se realiza las pruebas en tiempo real es recomendable que los equipos SDR Adalm Pluto estén a una distancia considerable con respecto al piso, en este proyecto se trabajó con una distancia de entre 1.5 metros a 1.8 metros y se obtuvo excelentes resultados. Mientras más alto estén los dispositivos mejor información se emitirá y se receptará.

Es preferible que las antenas estén bien ajustadas a los SDR Adalm Pluto para que la señal que se transmita se emita de la mejor manera y que el receptor pueda recibir la mayor cantidad de información y así poder obtener los mejores valores de BER.

En todos los programas se maneja 4 tipos de Mapeo, los cuales si se desean cambiar se debe modificar en el programa que se requiera ejecutar.

4. MANUAL DE USUARIO

4.1. SIMULACIÓN

4.1.1. CONVOLUCIONAL

- Se realiza lo expresado en la Figura 1.1
- Se abre el archivo *Simulacion_Convolucional.m* como se muestra en la Figura 4.1,

Figura 4.1 Archivo Simulacion_Convolucional.m

- Dentro del programa *Simulacion_Convolucional.m* se escoge el tipo de mapeo a utilizar, como se muestra en la Figura 4.2

```
15      % Para elegir el tipo de mapeo que se pretende utilizar la
16      % variable seleccionMapeo, en donde si se seleccionamos BPSK se debe poner
17      % el valor igual a 1, para QPSK igual a 2, para 16-QAM igual a 3 y
18      % para 64-QAM igual a 4:
19 -    seleccionMapeo=3;
```

Figura 4.2 Elección del tipo de mapeo

- Se ejecuta el archivo *Simulacion_Convolucional.m,*
- Se escoge el archivo ,txt a transmitir, en este caso se llama *ArchivoTextoRT3.txt,* como se muestra en la Figura 4.3.

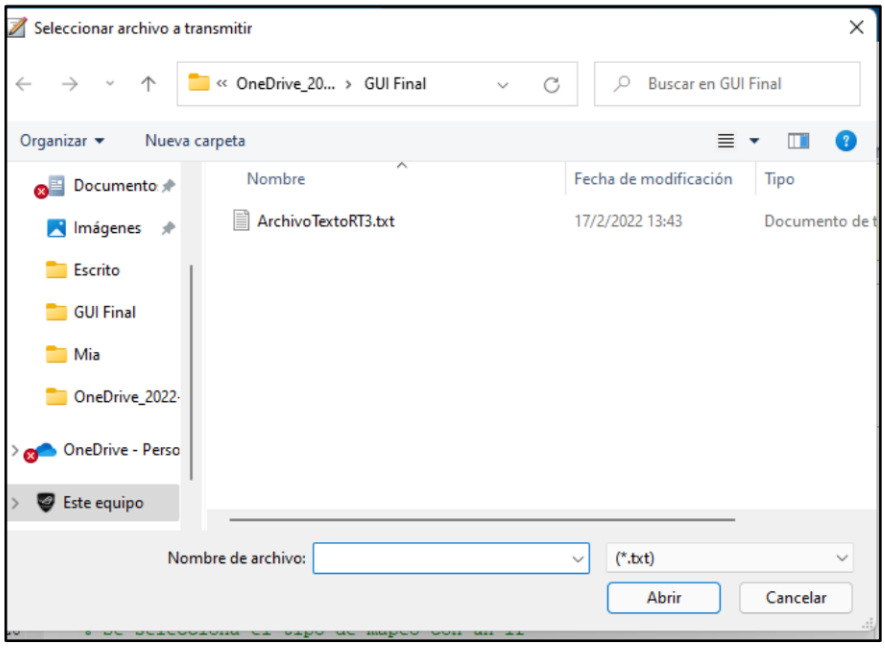


Figura 4.3 Archivo *ArchivoTextoRT3.txt*

- Se obtiene el resultado como se muestra en la Figura 4.4

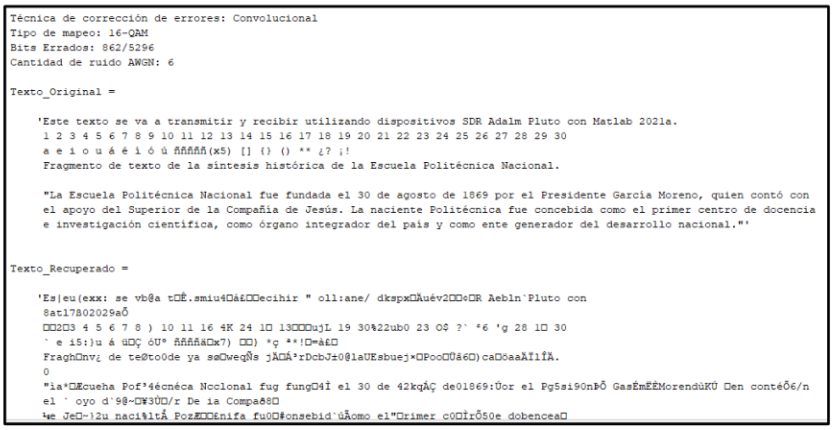


Figura 4.4 Resultado de simulación convolucional.

4.1.2. REED – SOLOMON

- Se realiza lo expresado en la Figura 1.1
- Se abre el archivo *Simulacion_ReedSolomon.m* como se muestra en la Figura 4.5,

Figura 4.5 Archivo *Simulacion_ReedSolomon.m*

- Dentro del programa *Simulacion_ReedSolomon.m* se escoge el tipo de mapeo a utilizar, como se muestra en la Figura 4.6

```
62      %% Mapeo
63      % Con la siguiente variable se selecciona el tipo de Mapeo:
64 -    seleccionMapeo=3;
65      % Para elegir el tipo de mapeo que se pretende utilizar la
66      % variable seleccionMapeo, en donde se selecciona 1 para BPSK, 2 para QPSK,
67      % 3 para 16-QAM y 4 para 64-QAM.
68      % Con un lazo if se discrimina el mapeo a utilizar
```

Figura 4.6 Elección del tipo de mapeo

- Se ejecuta el archivo *Simulacion_ReedSolomon.m,*
- Se escoge el archivo ,txt a transmitir, en este caso se llama *ArchivoTextoRT3.txt,* como se muestra en la Figura 4.7.

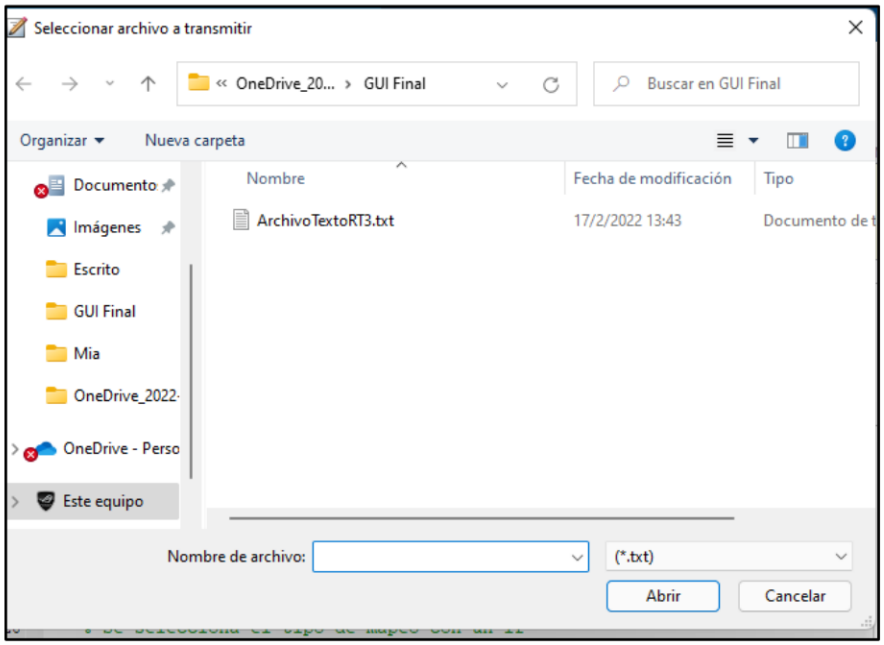


Figura 4.7 Archivo *ArchivoTextoRT3.txt*

- Se obtiene el resultado como se muestra en la Figura 4.8

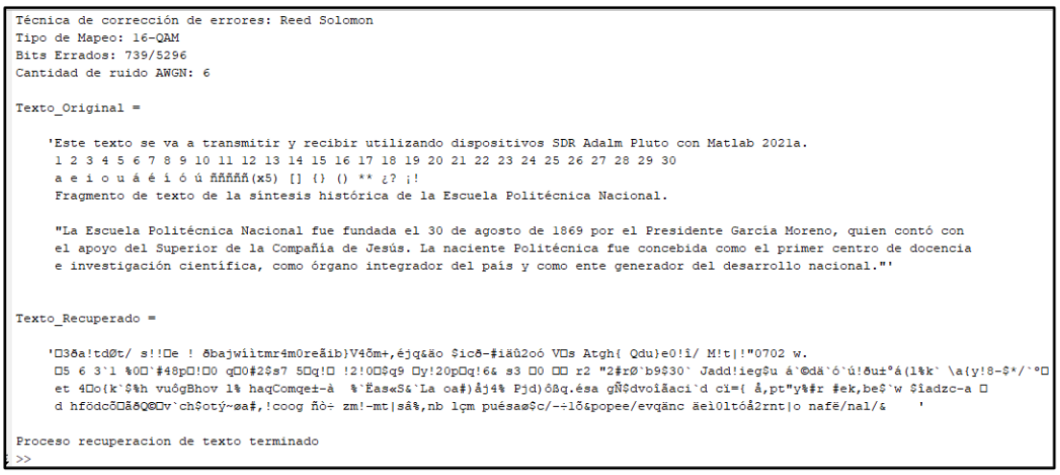


Figura 4.8 Resultado de simulación Reed - Solomon.

4.1.3. TURBO CÓDIGOS

- Se realiza lo expresado en la Figura 1.1
- Se abre el archivo *Simulacion_Turbocodigo.m* como se muestra en la Figura 4.9,

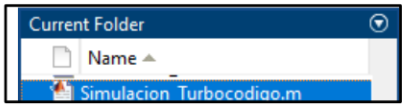


Figura 4.9 Archivo *Simulacion_Turbocodigo.m*

- Dentro del programa *Simulacion_Turbocodigo.m* se escoge el tipo de mapeo a utilizar, como se muestra en la Figura 4.10

```
13      % Para elegir el tipo de mapeo que se pretende utilizar la
14      % variable seleccionMapeo, en donde si se seleccionamos BPSK se debe poner
15      % el valor igual a 1, para QPSK igual a 2, para 16-QAM igual a 3 y
16      % para 64-QAM igual a 4
17 -    seleccionMapeo=3;
```

Figura 4.10 Elección del tipo de mapeo

- Se ejecuta el archivo *Simulacion_Turbocodigo.m,*
- Se escoge el archivo ,txt a transmitir, en este caso se llama *ArchivoTextoRT3.txt,* como se muestra en la Figura 4.11.

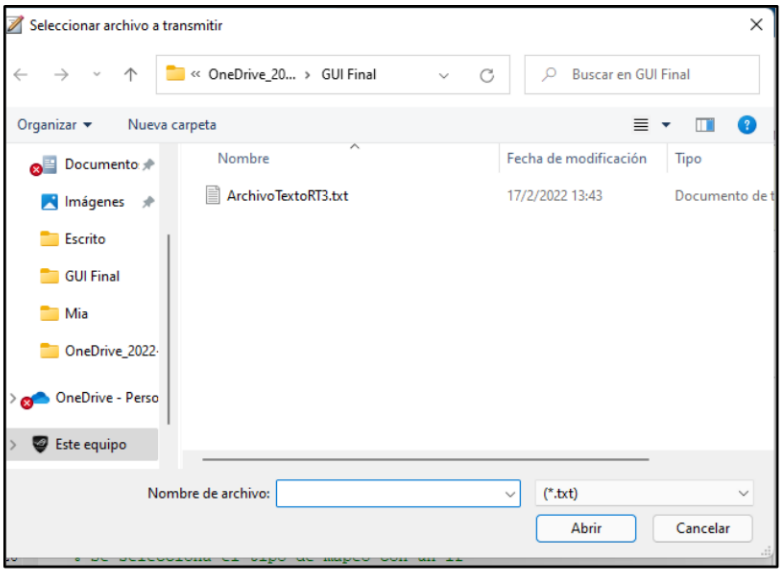


Figura 4.11 Archivo *ArchivoTextoRT3.txt*

- Se obtiene el resultado como se muestra en la Figura 4.12

```
Técnica de corrección de errores: Turbo código
Tipo de Mapeo: 16-QAM
Bits Errados: 0/5296
Cantidad de ruido AWGN: 6

Texto_Original =

    'Este texto se va a transmitir y recibir utilizando dispositivos SDR Adalm Pluto con Matlab 2021a.
     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
     a e i o u á é í ó ú ñÑñÑñ(x5) [] {} () ** ¿? ¡!
     Fragmento de texto de la sintesis histórica de la Escuela Politécnica Nacional.

     "La Escuela Politécnica Nacional fue fundada el 30 de agosto de 1869 por el Presidente García Moreno, quien contó con
     el apoyo del Superior de la Compañía de Jesús. La naciente Politécnica fue concebida como el primer centro de docencia
     e investigación científica, como órgano integrador del país y como ente generador del desarrollo nacional."'


Texto_Recuperado =

    'Este texto se va a transmitir y recibir utilizando dispositivos SDR Adalm Pluto con Matlab 2021a.
     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
     a e i o u á é í ó ú ñÑñÑñ(x5) [] {} () ** ¿? ¡!
     Fragmento de texto de la sintesis histórica de la Escuela Politécnica Nacional.

     "La Escuela Politécnica Nacional fue fundada el 30 de agosto de 1869 por el Presidente García Moreno, quien contó con
     el apoyo del Superior de la Compañía de Jesús. La naciente Politécnica fue concebida como el primer centro de docencia
     e investigación científica, como órgano integrador del país y como ente generador del desarrollo nacional."'
```

Figura 4.12 Resultado de simulación Turbo códigos.

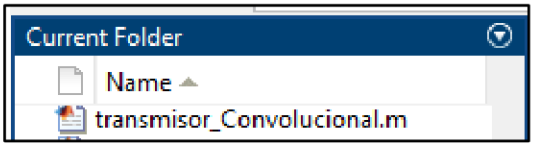
4.2. IMPLEMENTACIÓN PRÁCTICA

4.2.1. CONVOLUCIONAL

- Se realiza lo expresado en la Figura 1.2
- Se abre y se ejecuta el archivo *configurarAdalmPluto.m* en nuestras dos laptops como se muestra en la Figura 4.13. Dentro de este programa se encuentra las configuraciones de los SDR Adalm Pluto.

Figura 4.13 Archivo *configurarAdalmPluto.m*

- En la parte del transmisor se ejecuta el programa *transmisor_Convolucional.m* como se muestra en la Figura 4.14.



- **Figura 4.14** Archivo *transmisor_Convolucional.m*

- Dentro del programa *transmisor_Convolucional.m* se escoge el tipo de mapeo a utilizar, como se muestra en la Figura 4.15

```
%% TRANSMISIÓN CONVOLUCIONAL
%% Parámetros para seleccionar algoritmos y controlar la transmisión
seleccionMapeo=3; % Variable para seleccionar el tipo de mapeo a utilizar,
% BPSK (1), QPSK (2), 16-QAM(3), 64-QAM(4)
```

Figura 4.15 Elección del tipo de mapeo

- Una vez ejecutado el archivo *transmisor_Convolucional.m,* se escoge el archivo ,txt a transmitir, en este caso se llama *ArchivoTextoRT3.txt,* como se muestra en la Figura 4.16

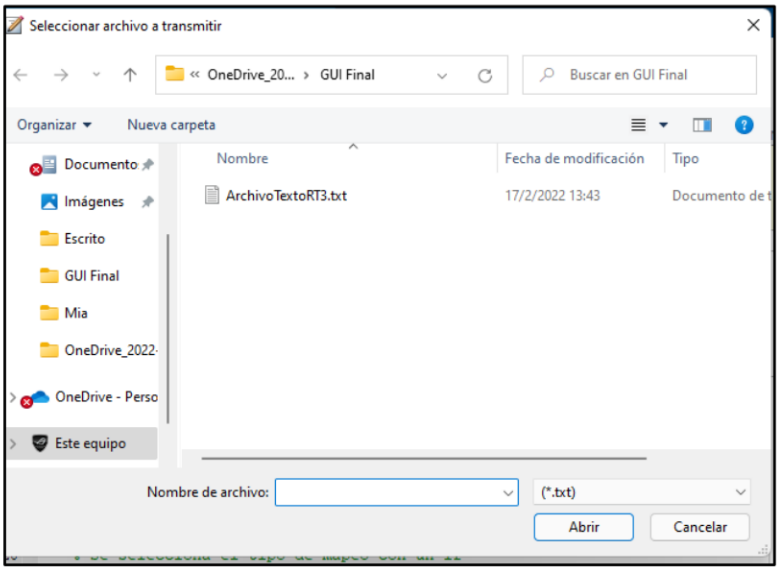


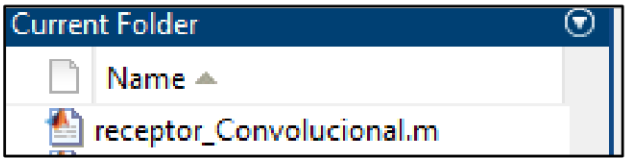
Figura 4.16 Archivo *ArchivoTextoRT3.txt*

- Se empieza a transmitir como se muestra en la Figura 4.17

```
Dispositivo configurado
>> transmisor_Convolucional
Datos Serializados
Codificación Convolucional Agregada
Técnica de corrección de errores: Convolucional
Tipo de mapeo: 16-QAM
Símbolos OFDM creados
IFFT implementada
Prefijo cíclico (PC) añadido
Preámbulo 802.11a añadido
Datos Procesados
## Establishing connection to hardware. This process can take several seconds.
Datos transmitidos...
Datos transmitidos...
Datos transmitidos...
```

Figura 4.17 Datos transmitidos Transmisor Convolucional

- En la parte del receptor se selecciona el programa *receptor_Convolucional.m* como se muestra en la Figura 4.18.



- **Figura 4.18** Archivo *receptor_Convolucional.m*

- Dentro del programa *receptor_Convolucional.m* se escoge el tipo de mapeo a utilizar, como se muestra en la Figura 4.19

```
3 -     seleccionMapeo=3; % Variable para seleccionar el tipo de mapeo a utilizar,
4       % BPSK (1), QPSK (2), 16-QAM(3), 64-QAM(4)
```

Figura 4.19 Elección del tipo de mapeo

- Una vez ejecutado el archivo *receptor_Convolucional.m,* se escoge el archivo ,txt a transmitir, en este caso se llama *ArchivoTextoRT3.txt,* como se muestra en la Figura 4.20

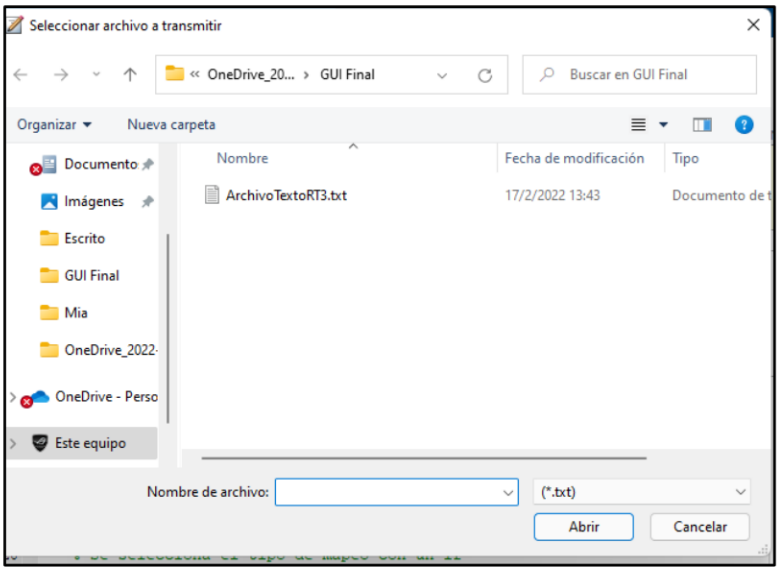


Figura 4.20 Archivo *ArchivoTextoRT3.txt*

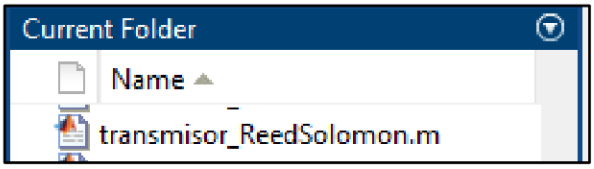
- Se empieza a receptar la información como se muestra en la Figura 4.21

```
Corrección de errores implementado
Medición del BER:
Bits errados/Bits totales = 82/5296
BER = 0.0154834

Texto_Original =

    'Este texto se va a transmitir y recibir utilizando dispositivos SDR Adalm Pluto con Matlab 2021a.
     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
     a e i o u á é í ó ú ñÑñÑñ(x5) [] {} () ** ¿? ¡!
     Fragmento de texto de la síntesis histórica de la Escuela Politécnica Nacional.

     "La Escuela Politécnica Nacional fue fundada el 30 de agosto de 1869 por el Presidente García Moreno, quien contó con
     el apoyo del Superior de la Compañía de Jesús. La naciente Politécnica fue concebida como el primer centro de docencia
     e investigación científica, como órgano integrador del país y como ente generador del desarrollo nacional."'

Texto recuperado:

Texto_Recuperado =

    'Mste texto se ~a a transmitir y reÉébir utilizando dispositivos SDR Adalm Plutmúcon Matlab 2021a.
     □ 2 3 4 5 3 7 8 9 10 11 12 13 14 15 16 17 18 19 20"21 22 23 24 25 26 27 28 29 30!
     a e i nuu á é í óuú ñÑñÑñ(x5) [] {} () ** ¿? ¡!
     Fragmento□de texto%de la síngesis histórica de la EscÊ%la Rolitécnica Nacional.

     "La Escuela Politécnica Nacional fue fundada gl 30 de a}osto de 1869 por el!Presidente García Moreno, quien coÎtó con
     eD ap□yo del Superior de la Compañia de Jesús/ La naciente Politécnica fue cÍncebida aomo el primer centro de docenaia
     e invmstigacióc científica, como órgano intmgrador del país y como en$e gen`2ador del desarrol□o naÃional."'
```

Figura 4.21 Datos recibidos Receptor Convolucional

4.2.2. REED – SOLOMON

- Se realiza lo expresado en la Figura 1.2
- Se abre y se ejecuta el archivo *configurarAdalmPluto.m* en nuestras dos laptops como se mostró en la Figura 4.13. Dentro de este programa se encuentra las configuraciones de los SDR Adalm Pluto.
- En la parte del transmisor se ejecuta el programa *transmisor_ReedSolomon.m* como se muestra en la Figura 4.22.



- **Figura 4.22** Archivo *transmisor_ReedSolomon.m*

- Dentro del programa *transmisor_ReedSolomon.m* se escoge el tipo de mapeo a utilizar, como se muestra en la Figura 4.23

```
3 -    seleccionMapeo=3; % Variable para seleccionar el tipo de mapeo a utilizar,
4      % BPSK (1), QPSK (2), 16-QAM(3), 64-QAM(4)
```

Figura 4.23 Elección del tipo de mapeo

- Una vez ejecutado el archivo *transmisor_ReedSolomon.m,* se escoge el archivo ,txt a transmitir, en este caso se llama *ArchivoTextoRT3.txt,* como se muestra en la Figura 4.24

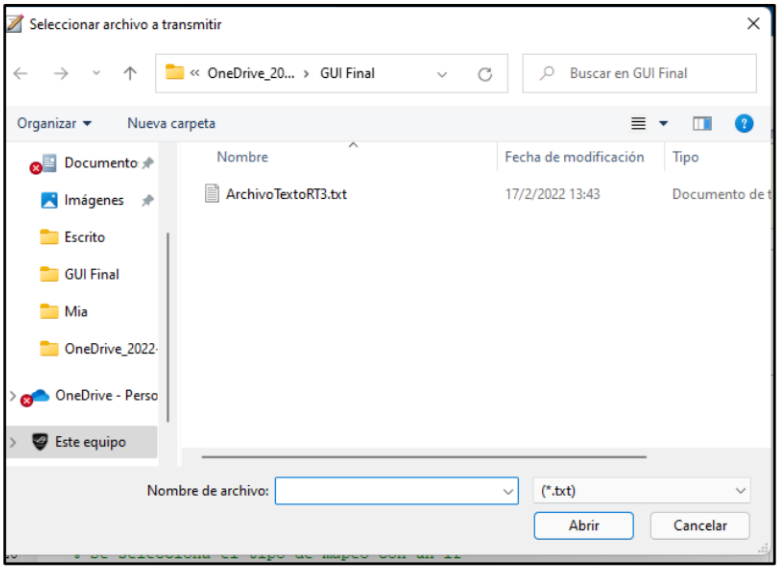


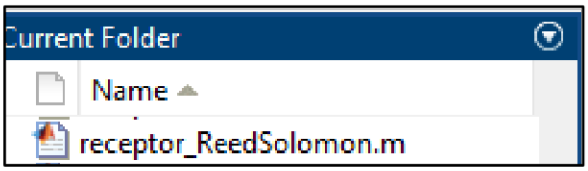
Figura 4.24 Archivo *ArchivoTextoRT3.txt*

- Se empieza a transmitir como se muestra en la Figura 4.25

```
>> transmisor_ReedSolomon
Datos Serializados
Codificación Reed Solomon Agregada
Técnica de corrección de errores: Reed Solomon
Tipo de Mapeo: 64-QAM
Símbolos OFDM creados
IFFT implementada
Prefijo cíclico (PC) añadido
Preámbulo 802.11a añadido
Datos Procesados
Datos transmitidos...
Datos transmitidos...
Datos transmitidos...
```

Figura 4.25 Datos transmitidos Transmisor Reed - Solomon

- En la parte del receptor se selecciona el programa *receptor_ReedSolomon.m* como se muestra en la Figura 4.26.



- **Figura 4.26** Archivo *receptor_ReedSolomon.m*

- Dentro del programa *receptor_ReedSolomon.m* se escoge el tipo de mapeo a utilizar, como se muestra en la Figura 4.27

```
3 -    seleccionMapeo=3; % Variable para seleccionar el tipo de mapeo a utilizar,
4      % BPSK (1), QPSK (2), 16-QAM(3), 64-QAM(4)
```

Figura 4.27 Elección del tipo de mapeo

- Una vez ejecutado el archivo *receptor_ReedSolomon.m,* se escoge el archivo ,txt a transmitir, en este caso se llama *ArchivoTextoRT3.txt,* como se muestra en la Figura 4.28

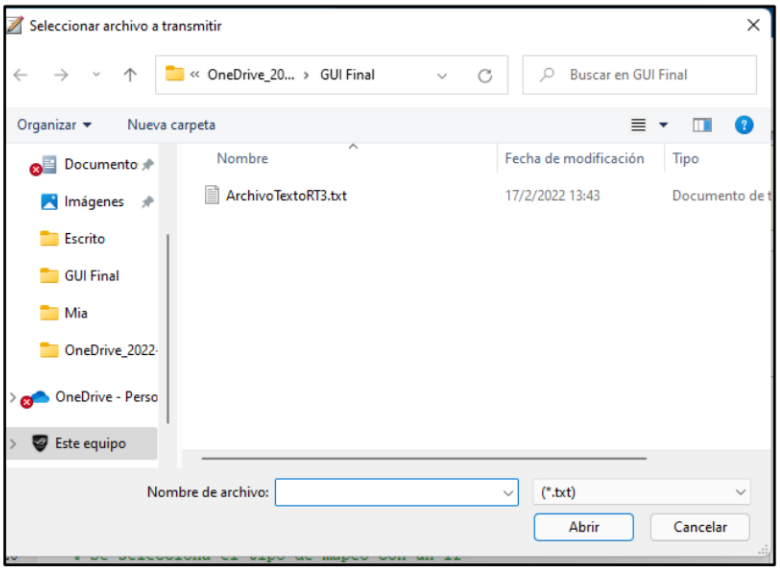


Figura 4.28 Archivo *ArchivoTextoRT3.txt*

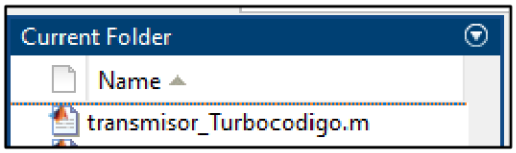
- Se empieza a receptar la información como se muestra en la Figura 4.29

```
Corrección de errores implementado
Medición del BER:
Bits errados/Bits totales = 209/5296
BER = 0.0394637

Texto_Original =

    'Este texto se va a transmitir y recibir utilizando dispositivos SDR Adalm Pluto con Matlab 2021a.
     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
     a e i o u á é í ó ú ñÑñÑñ(x5) [] {} () ** ¿? ¡!
     Fragmento de texto de la síntesis histórica de la Escuela Politécnica Nacional.

     "La Escuela Politécnica Nacional fue fundada el 30 de agosto de 1869 por el Presidente García Moreno, quien contó con
     el apoyo del Superior de la Compañía de Jesús. La naciente Politécnica fue concebida como el primer centro de docencia
     e investigación científica, como órgano integrador del país y como ente generador del desarrollo nacional."'

Texto recuperado:

Texto_Recuperado =

    '□bpe texto$qe!va a`trajslidir }avecibhv uuiliz`.do lispmsitivos SDV`□dalm P,eto con M`pl`b`rl2la.
     1 2 3 4 5 6 7 8 9 0□ 11 12 13 □4415 16 q7$08 q9`20 3□$□□$23 24 25 2□027 28 3□ 1□ I
     a e i o u á é í ó ú ñÑñÑñ(x5) [I {} (+ **`ÿ? ¡l
     Fragmen|/ de texto de la síntesiq$èistówyca$le la Ewcueh  Pol)\écnic`eNccional.
     □]□"La Essuelq`Polmtéc.iaa Naci f`(0"we fundada el 30 de a'/qdo de 1869 por el P2$3)dáîpa Garcí  Moreno, qU)%n cont÷ gon ]
     el apoyo del Ru`erior dg la$Comp`ñía de Jesú3> □c nacientmapëlitécnica0fue concebiäc #om `el `rimer geo|rg0$- docenc(a
     e inver4igaciónaâèentíiica- col/ órg`n/ahêpâgrador àål país y com/ %npã`ge.erador!àel desa2r/llo nag}onal."'
```

Figura 4.29 Datos recibidos Receptor Reed - Solomon

4.2.3. TURBO CÓDIGOS

- Se realiza lo expresado en la Figura 1.2
- Se abre y se ejecuta el archivo *configurarAdalmPluto.m* en nuestras dos laptops como se mostró en la Figura 4.13. Dentro de este programa se encuentra las configuraciones de los SDR Adalm Pluto.
- En la parte del transmisor se ejecuta el programa *transmisor_Turbocodigo.m* como se muestra en la Figura 4.30



- **Figura 4.30** Archivo *transmisor_Turbocodigo.m*

- Dentro del programa *transmisor_Turbocodigo.m* se escoge el tipo de mapeo a utilizar, como se muestra en la Figura 4.31

```
3 -    seleccionMapeo=3; % Variable para seleccionar el tipo de mapeo a utilizar,
4      % BPSK (1), QPSK (2), 16-QAM(3), 64-QAM(4)
```

Figura 4.31 Elección del tipo de mapeo

- Una vez ejecutado el archivo *transmisor_Turbocodigo.m,* se escoge el archivo ,txt a transmitir, en este caso se llama *ArchivoTextoRT3.txt,* como se muestra en la Figura 4.32

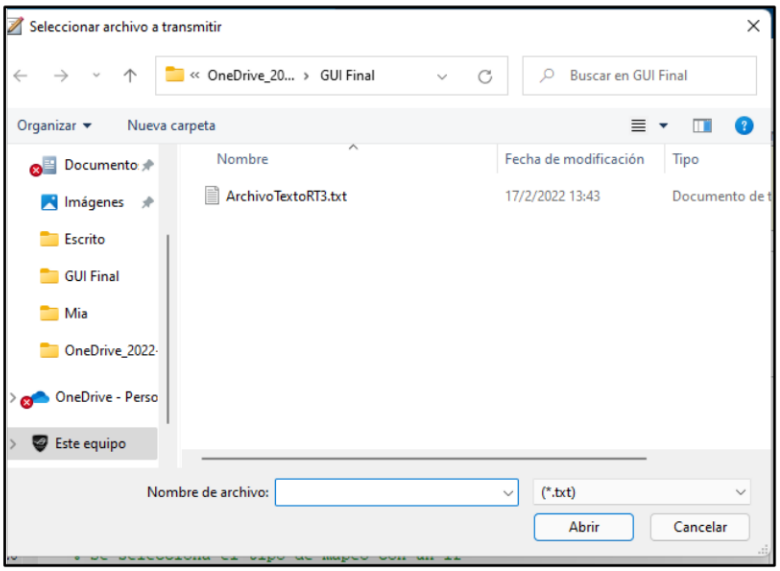


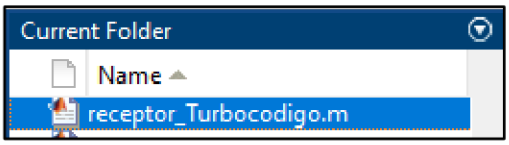
Figura 4.32 Archivo *ArchivoTextoRT3.txt*

- Se empieza a transmitir como se muestra en la Figura 4.33

```
Tipo de Mapeo: 64-QAM
Símbolos OFDM creados
IFFT implementada
Prefijo cíclico (PC) añadido
Preámbulo 802.11a añadido
Datos Procesados
Datos transmitidos...
Datos transmitidos...
Datos transmitidos...
```

Figura 4.33 Datos transmitidos Transmisor Reed - Solomon

- En la parte del receptor se selecciona el programa *receptor_Turbocodigo.m* como se muestra en la Figura 4.34.



- **Figura 4.34** Archivo *receptor_Turbocodigo.m*

- Dentro del programa *receptor_Convolucional.m* se escoge el tipo de mapeo a utilizar, como se muestra en la Figura 4.35

```
3 -     seleccionMapeo=3; % Variable para seleccionar el tipo de mapeo a utilizar,
4       % BPSK (1), QPSK (2), 16-QAM(3), 64-QAM(4)
```

Figura 4.35 Elección del tipo de mapeo

- Una vez ejecutado el archivo *receptor_Turbocodigo.m,* se escoge el archivo ,txt a transmitir, en este caso se llama *ArchivoTextoRT3.txt,* como se muestra en la Figura 4.36

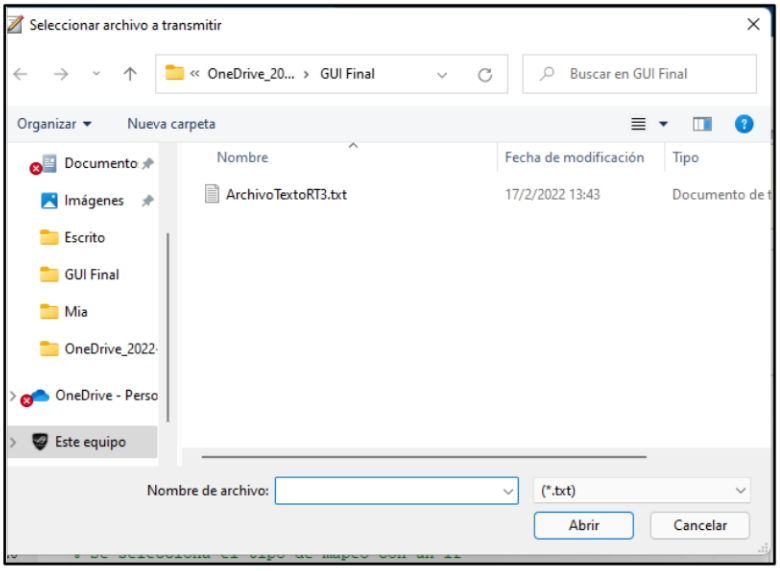


Figura 4.36 Archivo *ArchivoTextoRT3.txt*

- Se empieza a receptar la información como se muestra en la Figura 4.37

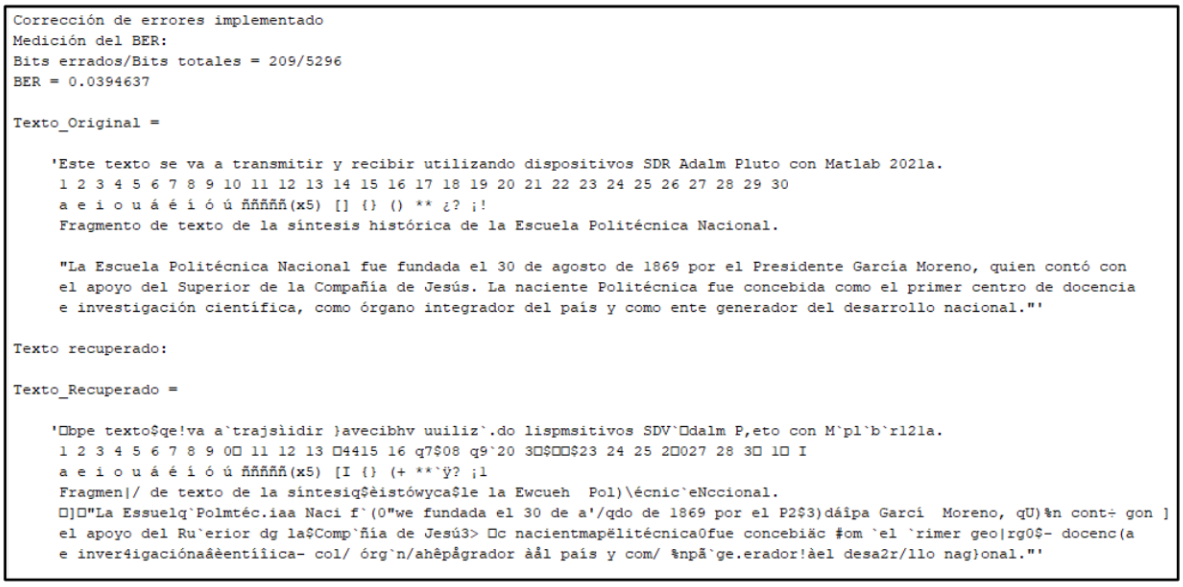


Figura 4.37 Datos recibidos Receptor Turbo códigos

5. REFERENCIAS BIBLIOGRÁFICAS

[1] Analog Devices, “ADALM-PLUTO SDR Active Learning Module”. 2021. Accedido: 16 de febrero de 2022. [En línea]. Disponible en: https://www.analog.com/media/en/news-marketing-collateral/product-highlight/ADALM-PLUTO-Product-Highlight.pdf

[2] T.F. Collins, R. Getz, D. Pu y A. M. Wyglinski, “Software Defined Radio for Engineers”, 2018. [En línea]. Disponible en: https://www.analog.com/media/en/training-seminars/design-handbooks/Software-Defined-Radio-for-Engineers-2018/SDR4Engineers.pdf

[3] MathWorks, “Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio”. Accedido: 17 de febrero de 2022. [En línea]. Disponible en: https://www.mathworks.com/help/supportpkg/plutoradio/index.html?s_cid=doc_ftr

[4] Madelin Gabriela Gordón Enríquez, Francisco Javier Hidalgo Pazmiño, “Desarrollo de un prototipo de comunicación inalámbrica de uso didáctico utilizando equipos SDR, incluyendo la técnica OFDM y el servicio de confidencialidad”, EPN, octubre 2020, Quito-Ecuador.

[5] A. Bensky, "*Short-range Wireless Communications"*, 3era ed. Reino Unido: Elsevier Inc., 2019.

[6] O. Quilumbango, “MATERIAL DIDÁCTICO PARA EL ANÁLISIS Y SIMULACIÓN DEL DESEMPEÑO DE LA CAPA FÍSICA DE WIMAX/IEEE 802.16, USANDO LA

INTERFAZ GRÁFICA DE MATLAB (GUI)", Escuela Politécnica Nacional, Quito-Ecuador, 2011.

[7] C. Espín, "ANÁLISIS, SIMULACIÓN E IMPLEMENTACIÓN CON CARÁCTER DIDÁCTICO DEL MÉTODO DE DETECCIÓN Y CORRECCIÓN DE ERRORES CONVOLUCIONAL Y SU DECODIFICACIÓN USANDO EL ALGORITMO DE VITERVI", Escuela Politécnica Nacional, Quito-Ecuador, 2017.

[8] Ingeniería en Electrónica y Telecomunicaciones, "Diagrama de Trellis". Accedido: 19 de febrero de 2022. [En línea]. Disponible en: https:// www.studocu.com/ec/document/escuela-politecnica-nacional/comunicacion-digital/diagrama-de-trellis/5239448

[9] MathWorks, "comm.SDRTxPluto". 2021. Accedido: 16 de febrero de 2022. [En línea]. Disponible en: https://www.mathworks.com/help/supportpkg/plutoradio/ref/comm.sdrtxpluto-system-object.html

[10] MathWorks, "comm.SDRRxPluto". 2021. Accedido: 19 de febrero de 2022. [En línea]. Disponible en: https://www.mathworks.com/help/supportpkg/plutoradio/ref/comm.sdrrxpluto-system-object.html

[11] Sandoval E Fedón A., "CODIFICADOR Y DECODIFICADOR DIGITAL REED-SOLOMON PROGRAMADOS PARA HARDWARE RECONFIGURABLE". 2007. Accedido: 19 de febrero de 2022. [En línea]. Disponible en: https:// www.redalyc.org/pdf/477/47711102.pdf

[12] Wikioes, "Código turbo - Turbo code". 2021. Accedido: 19 de febrero de 2022. [En línea]. Disponible en: https:// wikioes.icu/wiki/Turbo_code.html

[13] MathWorks, "poly2trllis". 2021. Accedido: 19 de febrero de 2022. [En línea]. Disponible en: https://la.mathworks.com/help/comm/ref/poly2trellis.html

[14] MathWorks, "vitdec". 2021. Accedido: 8 de enero de 2022. [En línea]. Disponible en: https://www.mathworks.com/help/comm/ref/vitdec.html

[15] MathWorks, "gf". 2021. Accedido: 16 de febrero de 2022. [En línea]. Disponible en: https://www.

mathworks.com/help/comm/ref/gf.html?searchHighlight=gf&s_tid=srchtitle_gf_1.html

[16] MathWorks, “rsenc”. 2021. Accedido: 17 de febrero de 2022. [En línea]. Disponible en: https://la.mathworks.com/help/comm/ref/rsenc.html?searchHighlight=rsenc&s_tid =srchtitle_rsenc_1.html

[17] MathWorks, “comm.TurboEncoder”. 2021. Accedido: 19 de febrero de 2022. [En línea]. Disponible en: https://www.mathworks.com/help/comm/ref/comm.turboencoder-system-object.html?searchHighlight=comm.TurboEncoder%20&s_tid=srchtitle_comm.TurboEncoder%20_1#d123e230944.html

[18] MathWorks, “comm.TurboDecoder”. Accedido: 17 de febrero de 2022. [En línea]. Disponible en: https:// la.mathworks.com/help/comm/ref/comm.turbodecoder-system-object.html?searchHighlight=comm.TurboDecoder&s_tid=srchtitle_comm.TurboDecoder_1

[19] Tarrés F Cabrera M., “Codificación de canal II: códigos convolucionales”. 2021. Accedido: 19 de febrero de 2022. [En línea]. Disponible en: Teoría de la codificación y modulaciones avanzadas, septiembre 2012 (uoc.edu).html

[20 MathWorks, “uigetfile”. Accedido: 17 de febrero de 2022. [En línea]. Disponible en: https:// la.mathworks.com/help/matlab/ref/uigetfile.html?searchHighlight=uigetfile%20&s_tid=srchtitle_uigetfile%20_1

[21] MathWorks, “Unicode and ASCII Values”. 2021. Accedido: 14 de septiembre de 2021. [En línea]. Disponible en: https://www.mathworks.com/help/matlab/matlab_prog/unicode-and-ascii-values.html

[22] “ASCII table , ascii codes”. Accedido: 14 de septiembre de 2021. [En línea]. Disponible en: https://theasciicode.com.ar/

[23] T. Collins, R. Getz, D. Pu, y A. Wyglinski, Software-Defined Radio for Engineers. U.S.A.: Analog Devices, 2018. Accedido: 30 de septiembre de 2020. [En línea].

Disponible en: https://www.analog.com/media/en/training-seminars/design-handbooks/Software-Defined-Radio-for-Engineers-2018/SDR4Engineers.pdf

[24] S. Jiménez y D. Panchi, “DISEÑO E IMPLEMENTACIÓN DE UN MODULADOR Y UN DEMODULADOR N-QAM EMPLEANDO XILINX ISE, SYSTEM GENERATOR Y SIMULINK SOBRE UNA TARJETA DE ENTRENAM.

[25] L. Cheuk, “Method of Synchronization using IEEE 802.1 la OFDM Training Structure for Indoor Wireless Applications”, SIMON FRASER, California, 2004.

6. ANEXOS

ANEXO I

Simulacion_Convolucional.m

```
%% Simulación Codificación Convolucional.
clc
clear all;
close all;
%% TRANSMISOR
%% Datos de entrada (texto)
msg_abc=uigetfile('*.txt;','Seleccionar archivo a transmitir'); %Selecciona
% el archivo .txt a codificar}
fID = fopen(msg_abc); % Abrir el archivo
datosAbiertos=fread(fID,'*char'); % Lee los caracteres
msg=reshape(dec2bin(datosAbiertos,8).'-'0',1,[]);   %Transforma a binario
%% Selección de Mapeo
% Primero se define los valores para seleccionar el Mapeo
NPdatos=40; % Número de portadoras de datos por símbolo OFDM
% Para elegir el tipo de mapeo que se pretende utilizar la
% variable seleccionMapeo, en donde si se seleccionamos BPSK se debe poner
% el valor igual a 1, para QPSK igual a 2, para 16-QAM igual a 3 y
% para 64-QAM igual a 4:
seleccionMapeo=1;
% Se selecciona el tipo de mapeo con un if
if (seleccionMapeo==1)
   % BPSK
   M = 2; %2 para BPSK
elseif (seleccionMapeo==2)
   % QPSK
   M = 4; %4 para QPSK
elseif (seleccionMapeo==3)
   % 16-QAM
   M = 16; %16 para 16-QAM
elseif (seleccionMapeo==4)
   % 64-QAM
   M = 64; %64 para 64-QAM
end
Nbps_M=log2(M); % Se calcula los bits por símbolo según la
% opción de mapeo que se seleccionó
%% Codificación Convolucional
% Parámetros del codificador convolucional: (2, 1, 3)
nc=2; % n es la longitud por cada ingreso de k bits
kc=1; % k es los bits de entrada
mc=3; % m es el tamaño de registro de la memoria
g1=5; % Polinomio generador x2+1=5(oct)
g2=7; % Polinomio generador x2+x+1=7(oct)
Rcc=kc/nc; % tasa del codificador convolucional
% Cálculo del número de bits serializados que se tiene por cada símbolo
% OFDM:
Nbits=Nbps_M*NPdatos*Rcc;
% Cálculo para rellenar el vector de entrada, y así obtener un vector
% que sea múltiplo de Nbits:
NumSimb=ceil(length(msg)/Nbits); % Número de simbolos OFDM
%ceil=redondea la longitud al entero inmediato superior
longitudPadd=Nbits*NumSimb-length(msg); % Longitud de relleno
vector_padd=zeros(1,longitudPadd); % vector de relleno de ceros
datosConPadd=[msg vector_padd]; % Datos serializados con relleno
```

```
% Creción de la estructura de Trellis:
P_Trellis = poly2trellis(mc, [g1 g2]);
% Codificador:
datosCodConv=[]; % se inicializa la variable que va a almacenar los
% datos codificados y serializados
% Se recorre los símbolos OFDM con un lazo for
for i=1:NumSimb
   auxDatosCc = datosConPadd(1,(Nbits*(i-1)+1):Nbits*i); % variable
   % auxiliar para almacenar los datos a codificar
   % Codificación convolucional:
   CodConv = convenc(auxDatosCc, P_Trellis);
   %Codifica auxDatosCc con la estructura de trellis P_Trellis
   datosCodConv=[datosCodConv CodConv]; % Datos de salida
end
datosBinario=datosCodConv; % Copiar los datos de salida en otra variable
%% Mapeo
% Con un if se discrimina el mapeo a utilizar
if(seleccionMapeo==1)
   % Mapeo BPSK
   datosColumna=datosBinario.'; % Se transpone la matriz que contiene
   % los datos de entrada para obtener un vector columna
   % Se recorre los cada bit de entrada y se asigna un símbolo
   for i=1:length(datosColumna)
      % Con el if discrimino si es igual a 1 y se asigna el
      % valor 0.707+0.707i, de no ser así es 0 y se asigna el
      % valor de -0.707-0.707i
      if(datosColumna(i)==1)
         datosMapeados(i,1)=complex(0.707,0.707);
      else
         datosMapeados(i,1)=complex(-0.707,-0.707);
      end
   end
   disp('Técnica de corrección de errores: Convolucional')
   disp('Tipo de mapeo: BPSK')
elseif(seleccionMapeo==2)
   % Mapeo QPSK
   M = 4; % Orden de modulación
   Nb=log2(M); % Bits por símbolo, se utiliza 2 bits por símbolo
   % Hay que asegurarse que la longitud de los datos de entrada sea
   % múltiplo de Nb, por lo que se rellena de ceros cuando no es
   % múltiplo de Nb.
   % Se calcula la longitud de relleno y se almacena en long_padd0
   long_padd0=ceil(length(datosBinario)/Nb)*Nb-length(datosBinario);
   %ceil=redondea la longitud al entero inmediato superior
   paddQ=zeros(1,long_padd0); % Vector de relleno con ceros
   datosbin=[datosBinario paddQ]; % Vector con relleno
   %Pasar de binario a símbolos decimales:
   dataMatriz=reshape(datosbin,Nb,[]).'; % Matriz de Nb columnas y
   % cada fila corresponde a un símbolo
   dataSimb=bi2de(dataMatriz,'left-msb'); % Símbolos a modular
   %bi2de convertir de binario a decimal
   %left-msb Indica la columna izquierda (o primera) de la salida binaria,
   %como el bit más significativo(o dígito de mayor orden)
   % Modulador:
   % Factor de normalización = 1/sqrt(2)
   datosMapeados = (1/sqrt(2))*pskmod(dataSimb,M,pi/4,'gray'); % Datos
   % mapeados con QPSK
   %pskmod=modulación por desplazamiento de fase de la entrada dataSimb
   %con una orden de modulación M, y una fase inial igual a pi/4
```

```
    %Gray significa el orden de símbolos, que esta codificada en orden gray
    disp('Técnica de corrección de errores: Convolucional')
    disp('Tipo de mapeo: QPSK')
elseif (seleccionMapeo==3)
    % Mapeo 16-QAM
    M = 16; % orden de modulación
    Nb=log2(M); % Bits por símbolo, se usan 4 bits por símbolo
    % Hay que asegurarse que la longitud de los datos de entrada sea
    % múltiplo de Nb, por lo que se rellena de ceros cuando no es
    % múltiplo de Nb.
    % Se calcula la longitud de relleno y se almacena en long_padd0
    long_padd0=ceil(length(datosBinario)/Nb)*Nb-length(datosBinario);
    %ceil=redondea la longitud al entero inmediato superior
    paddQ=zeros(1,long_padd0); % Vector de relleno con ceros
    datosbin=[datosBinario paddQ]; % Vector con relleno
    %Pasar de binario a símbolos decimales:
    dataMatriz=reshape(datosbin,Nb,[]).'; % Matriz de Nb columnas y
    % cada fila corresponde a un símbolo
    dataSimb=bi2de(dataMatriz,'left-msb'); % Símbolos a modular
    %bi2de convertir de binario a decimal
    %left-msb Indica la columna izquierda (o primera) de la salida binaria,
    %como el bit más significativo(o dígito de mayor orden).
    % Modulador:
    % Factor de normalización = 1/sqrt(10)
    datosMapeados = (1/sqrt(10))*qammod(dataSimb,M); % Modulación 16-QAM
    %qammod=modula la señal dataSimb con QAM con el orden de modulación M
    disp('Técnica de corrección de errores: Convolucional')
    disp('Tipo de mapeo: 16-QAM')
elseif (seleccionMapeo==4)
    % Mapeo 64-QAM
    M = 64; % Puntos en la constelación (símbolos),orden de modulación
    Nb=log2(M); % Bits por símbolo, se usan 6 bits por símbolo
    % Hay que asegurarse que la longitud de los datos de entrada sea
    % múltiplo de Nb, por lo que se rellena de ceros cuando no es
    % múltiplo de Nb.
    % Se calcula la longitud de relleno y se almacena en long_padd0
    long_padd0=ceil(length(datosBinario)/Nb)*Nb-length(datosBinario);
    %ceil=redondea la longitud al entero inmediato superior
    paddQ=zeros(1,long_padd0); % Vector de relleno con ceros
    datosbin=[datosBinario paddQ]; % Vector con relleno
    %Pasar de binario a símbolos decimales:
    dataMatriz=reshape(datosbin,Nb,[]).'; % Matriz de Nb columnas y
    % cada fila corresponde a un símbolo
    dataSimb=bi2de(dataMatriz,'left-msb'); % Símbolos a modular
    %bi2de convertir de binario a decimal
    %left-msb Indica la columna izquierda (o primera) de la salida binaria,
    %como el bit más significativo(o dígito de mayor orden).
    % Modulador:
    % Factor de normalización = 1/sqrt(42)
    datosMapeados = (1/sqrt(42))*(qammod(dataSimb,M)); % Datos
    % mapeados con 64-QAM
    %qammod=modula la señal dataSimb con QAM con el orden de modulación M
    disp('Técnica de corrección de errores: Convolucional')
    disp('Tipo de mapeo: 64-QAM')
end
%% Añadiendo ruido AWGN
ruidoy=6;
datosMapeo = awgn(datosMapeados,ruidoy,'measured'); % Señal con ruido,
% tal que awgn(entrada,SNR,'measured')
```

```
%measured para que la función awgn mida la potencia de datosMapeados antes
%de agregar ruido.
% datosMapeo=datosMapeados; % Si se desea probar sin ruido
%% RECEPTOR
%% Demapeo de datos
% Con un if se discrimina el demapeo a utilizar
if(seleccionMapeo==1)
   % Demapeo BPSK
   M=2; % Orden de modulación
   Nb=log2(M); % Bits por símbolo
   % Los datos filtrados no se normalizan
   longitud=length(datosMapeo); % Longitud datos de entrada
   datosDemod=zeros(longitud,1); % Inicializar vector
   % Implementar el Demapeo BPSK con el lazo for
   for i=1:longitud
      if(real(datosMapeo(i))>=0 && imag(datosMapeo(i))>=0)
         datosDemod(i)=1;
      elseif(real(datosMapeo(i))<0 && imag(datosMapeo(i))<0)
         datosDemod(i)=0;
      end
   end
elseif(seleccionMapeo==2)
   % Demapeo QPSK
   M = 4; % Orden de modulación
   Nb=log2(M); % Bits por símbolo
   long_pM=ceil(length(datosMapeo)/Nb)*Nb-length(datosMapeo);
   %ceil=redondea la longitud al entero inmediato superior
   paddM=zeros(long_pM,1); % relleno con ceros
   % Los datos filtrados solo se multiplican por sqrt(2)
   ModDat=(sqrt(2))*[datosMapeo;paddM]; %Datos con relleno
   % Demodulación QPSK:
   datosDemodS=pskdemod(ModDat,M,pi/4,'gray');
   %Demodula ModDat con una orden de modulación M y fase inicial de pi/4
   %Gray significa el orden de símbolos, que esta codificada en orden gray
   % Pasar de símbolos a bits:
   demodMatriz=de2bi(datosDemodS,Nb,'left-msb');
   %de2bi convertir de decimal a binario
   %left-msb Indica la columna izquierda (o primera) de la salida binaria,
   %como el bit más significativo(o dígito de mayor orden).
   datosDemod=reshape(demodMatriz.',1,[]).'; % Datos
   % recuperados (bits Serializados)
elseif(seleccionMapeo==3)
   % Demapeo 16-QAM
   M = 16; % orden de modulación
   Nb = log2(M); % Bits por símbolo
   long_pM=ceil(length(datosMapeo)/Nb)*Nb-length(datosMapeo);
   %ceil=redondea la longitud al entero inmediato superior
   paddM=zeros(long_pM,1); % relleno con ceros
   % Los datos filtrados solo se multiplican por sqrt(10)
   ModDat=(sqrt(10))*[datosMapeo;paddM]; % Datos con relleno
   % Demodulación 16-QAM:
   datosDemodS=qamdemod(ModDat,M);
   %qamdemod=demodula la señal ModDat con QAM con el orden de modulación M
   % Pasar de símbolos a bits:
   demodMatriz=de2bi(datosDemodS,Nb,'left-msb');
   %de2bi convertir de decimal a binario
   %left-msb Indica la columna izquierda (o primera) de la salida binaria,
   %como el bit más significativo(o dígito de mayor orden).
   datosDemod=reshape(demodMatriz.',1,[]).'; %Datos
```

```
    % recuperados (bits Serializados)
elseif(seleccionMapeo==4)
    % Demapeo 64-QAM
    M = 64; % Orden de modulación
    Nb = log2(M); % Bits por símbolo
    long_pM=ceil(length(datosMapeo)/Nb)*Nb-length(datosMapeo);
    %ceil=redondea la longitud al entero inmediato superior
    paddM=zeros(long_pM,1); % relleno con ceros
    % Los datos filtrados solo se multiplican por sqrt(42)
    ModDat=(sqrt(42))*[datosMapeo;paddM]; % Datos con relleno
    % Demodulación 64-QAM:
    datosDemodS=qamdemod(ModDat,M);
    %qamdemod=demodula la señal ModDat con QAM con el orden de modulación M
    % Pasar de símbolos a bits:
    demodMatriz=de2bi(datosDemodS,Nb,'left-msb');
    %de2bi convertir de decimal a binario
    %left-msb Indica la columna izquierda (o primera) de la salida binaria,
    %como el bit más significativo(o dígito de mayor orden).
    datosDemod=reshape(demodMatriz.',1,[]).'; %Datos
    % recuperados (bits Serializados)
end
%% Decodificador convolucional
nBc=Nb*NPdatos; % Número de bits codificados por simb OFDM
% Relleno de bits para que el vector de entrada sea multiplo de nBc:
nGrupos=ceil(length(datosDemod)/nBc);
%ceil=redondea la longitud al entero inmediato superior
longiPadd=nBc*nGrupos-length(datosDemod); % Longitud de relleno
v_relleno=zeros(longiPadd,1); % relleno de ceros
datosDemodRell=[datosDemod;v_relleno]; % Datos serializados con
% relleno de ceros
% La estructura de trellis y los parámetros se definen en
%la primera sección
tb=nBc*Rcc; % La profundidad de rastreo debe ser igual al número de
% bits que se tuvo a la entrada del codificador
datosDecodConv=[]; % Inicilizar vector que almacena los datos
% decodificados
% Con un for se recorre los grupos de bits a ser decodificados
for i=1:nGrupos
    auxDecod = datosDemodRell((nBc*(i-1)+1):nBc*i,1); % Se toma el
    % grupo correspondiente
    DecodVit= vitdec(auxDecod,P_Trellis,tb,'trunc','hard'); % Se
    % decodificación binaria convolucional de auxDecod con la estructura de
    %Trellis y profundidad de rastreo tb
    %trunc significa el modo de operación
    %hard significa el tipo de decodificación
    datosDecodConv=[datosDecodConv;DecodVit]; % Almacenar los bits
    % decodificados
end
%% Medición del BER
% Recortar datos de entrada para coincidir la longitud
msgErr=datosDecodConv(1:length(msg));%error de los datos decodificados con
%la longitud del msg de entrada
Nerrores=biterr(msg.',msgErr);
%biterr realiza el número de errores entre nuestra entrada msg y los datos
%recibidos, tasa de bits erroneos BER
fprintf('Bits Errados: %.0f/%.0f \n',Nerrores,length(msg))
fprintf('Cantidad de ruido AWGN: %.0f \n',ruidoy)
%% Recuperación del texto
    % Relleno de ceros:
```

```
bPc=8; % Bits por cada caracter
numC=ceil(length(datosDecodConv)/8); %Número caracteres
%ceil=redondea la longitud al entero inmediato superior
LongRellnumC=bPc*numC-length(datosDecodConv); % Longitud
% de relleno para hacer múltiplo de 8
rellenoC=zeros(LongRellnumC,1); % Vector de relleno ceros
datosSerializados=[datosDecodConv ; rellenoC]; %Rellenar
% en caso de ser necesario
% Pasar de los bits serializados a una matriz, en donde las
% filas contiene a los caracteres en binario:
datosMtz=reshape(datosSerializados,[8 numC]).'; % numC filas
% x 8 columnas
% Pasar de binario a decimal, los datos de cada fila:
datosTexto=[]; % Inicializar vector
for i=1:numC
   % Se pasa cada fila a decimal y se almacena en datosTexto
   datosTexto(1,i)=double(bi2de(datosMtz(i,:),'left-msb'));
   %bi2de convertir de binario a decimal
%left-msb Indica la columna izquierda (o primera) de la salida binaria,
%como el bit más significativo(o dígito de mayor orden).
end
%Texto Original
Texto_Original=transpose(datosAbiertos)
%Texto Recuperado
% Pasar de decimal a caracteres:
Texto_Recuperado=char(datosTexto) % Texto recuperado
disp('Proceso recuperacion de texto terminado');
```

ANEXO II

Simulacion_ReedSolomon.m

```
%% Simulación Codificación Reed-Solomon
clc
clear all;
close all;
%% TRANSMISOR
%% Datos de entrada (texto)
% Código corrector-detector colocarlo despues de Protocolo
% Esquema RS (5,3,1) (n,k,t)
% Se debe tener una matriz de Galois
%Abrir texto y que se haga bits.
msg_abc=uigetfile('*.txt;','Seleccionar archivo a transmitir'); %Selecciona
% el archivo .txt a codificar}
fID = fopen(msg_abc); % Abrir el archivo
   datosAbiertos=fread(fID,'*char'); % Leer los caracteres
msg=reshape(dec2bin(datosAbiertos,8).'-'0',1,[]);    %Transformo a binario
%% Selección de Mapeo
NPdatos=40; % número de portadora de datos por símbolo OFDM
NRS=5;      %Longitud del codificador
KRS=3;      %Símbolos originales
tasaRS=KRS/NRS;
Nbits=tasaRS*NPdatos; % bits de datos de entrada por cada símbolo OFDM
% Rellenar los datos:
m=8; % tamaño de símbolo (bits)
nS=ceil(length(msg)/Nbits); % número de símbolos (filas)
longitudPadd=Nbits*nS-length(msg); % Longitud de relleno
vector_padd=zeros(1,longitudPadd); % relleno de ceros
auxDatos=[msg vector_padd]; % Datos serializados con padd de ceros
```

```
% El vector datosDecimal puede tomarse directo de los símbolos decimales
% del archivo de texto
MatrizBin=reshape(auxDatos,m,[]).'; % Matriz de m columnas y cada fila
%corresponde a un símbolo
datosDecimal=bi2de(MatrizBin,'left-msb').'; % matriz con representación decimal
%bi2de convertir de binario a decimal
%left-msb Indica la columna izquierda (o primera) de la salida binaria,
%como el bit más significativo(o dígito de mayor orden)
datosBinario=[]; %Inicializo vector que almacena los datos Binarios
LsimbolosDec=length(datosDecimal); % longitud del vector de datos decimal
%% Codificador Reed - Solomon
%Se recorre en multiplos de KRS símbolos:
for i=1:KRS:LsimbolosDec
   auxDecimal=datosDecimal(i:i+KRS-1); % Tomar de KSR en KRS símbolos decim
   datosGalois=gf(auxDecimal,m); % Campo de Galois de la matriz auxDecimal
   %con el tamaño de símbolo para relleno de datos, m tiene que ser entero
   %de 1 a 16
   tic
   CodRS = rsenc(datosGalois,NRS,KRS); % codificador Reed-Solomon
   %Codifica la matriz datosGalois con la longitud del codificador  y los
   %símbolos originales y los guarda en CodRS
   toc
   CodRSd=double(CodRS.x); % Extraer los datos decimales de CodRS, estos
   % son los símbolos decimales de info mas los de paridad
   %simbolos a bits
   bitRS=de2bi(CodRSd.',m,'left-msb');
   %de2bi convertir de decimal a binario
   %left-msb Indica la columna izquierda (o primera) de la salida binaria,
   %como el bit más significativo(o dígito de mayor orden)
   bitRSs=reshape(bitRS.',1,[]); % vector con bits serializados
   %Reshape cambia el tamaño y la forma de bitRS
   datosBinario=[datosBinario bitRSs]; % Vector con bits
   % serializados acumulados
end
%% Mapeo
% Con la siguiente variable se selecciona el tipo de Mapeo:
seleccionMapeo=1;
% Para elegir el tipo de mapeo que se pretende utilizar la
% variable seleccionMapeo, en donde se selecciona 1 para BPSK, 2 para QPSK,
% 3 para 16-QAM y 4 para 64-QAM.
% Con un lazo if se discrimina el mapeo a utilizar
if(seleccionMapeo==1)
   % Mapeo BPSK
   datosColumna=datosBinario.'; % Se transpone la matriz que contiene
   % los datos de entrada para obtener un vector columna
   % Se recorre los cada bit de entrada y se asigna un símbolo
   for i=1:length(datosColumna)
      % Con if se discrimina si es igual a 1 y se asigna el
      % valor 0.707+0.707i, caso contrario es 0 y se asigna el
      % valor de -0.707-0.707i
      if(datosColumna(i)==1)
         datosMapeados(i,1)=complex(0.707,0.707);
      else
         datosMapeados(i,1)=complex(-0.707,-0.707);
      end
   end
   % Factor de normalización = 1
   disp('Técnica de corrección de errores: Reed Solomon')
   disp('Tipo de Mapeo: BPSK')
```

```
elseif(seleccionMapeo==2)
    % Mapeo QPSK
    M = 4; % Orden de modulación
    Nb=log2(M); % Bits por símbolo, se utiliza 2 bits por símbolo
    % Hay que asegurarse que la longitud de los datos de entrada sea
    % múltiplo de Nb, por lo que se rellena de ceros cuando no es
    % múltiplo de Nb.
    % Se calcula la longitud de relleno y se almacena en long_padd0
    long_padd0=ceil(length(datosBinario)/Nb)*Nb-length(datosBinario);
    %ceil=redondea la longitud al entero inmediato superior
    paddQ=zeros(1,long_padd0); % Vector de relleno con ceros
    datosbin=[datosBinario paddQ]; % Vector con relleno
    %Pasar de binario a símbolos decimales:
    dataMatriz=reshape(datosbin,Nb,[]).'; % Matriz de Nb columnas y
    % cada fila corresponde a un símbolo
    dataSimb=bi2de(dataMatriz,'left-msb'); % Símbolos a modular
    %bi2de cambio de binario a decimal dataMatriz
    %left-msb Indica la columna izquierda (o primera) de la salida binaria,
    %como el bit más significativo(o dígito de mayor orden)
    % Factor de normalización = 1/sqrt(2)
    datosMapeados = (1/sqrt(2))*pskmod(dataSimb,M,pi/4,'gray'); % Datos
    % mapeados con QPSK
    %pskmod modula por desplazamiento de fase dataSimb con un orden de
    %modulación M y una fase inicial igual a pi/4
    %Gray significa el orden de símbolos, que esta codificada en orden gray
    disp('Técnica de corrección de errores: Reed Solomon')
    disp('Tipo de Mapeo: QPSK')
elseif (seleccionMapeo==3)
    % Mapeo 16-QAM
    M = 16; % orden de modulación
    Nb=log2(M); % Bits por símbolo, se usan 4 bits por símbolo
    % Hay que asegurarse que la longitud de los datos de entrada sea
    % múltiplo de Nb, por lo que se rellena de ceros cuando no es
    % múltiplo de Nb.
    % Se calcula la longitud de relleno y se almacena en long_padd0
    long_padd0=ceil(length(datosBinario)/Nb)*Nb-length(datosBinario);
    %ceil=redondea la longitud al entero inmediato superior
    paddQ=zeros(1,long_padd0); % Vector de relleno con ceros
    datosbin=[datosBinario paddQ]; % Vector con relleno
    %Pasar de binario a símbolos decimales:
    dataMatriz=reshape(datosbin,Nb,[]).'; % Matriz de Nb columnas y
    % cada fila corresponde a un símbolo
    dataSimb=bi2de(dataMatriz,'left-msb');
    %bi2de cambio de binario a decimal
    %left-msb Indica la columna izquierda (o primera) de la salida binaria,
    %como el bit más significativo(o dígito de mayor orden)
    % Factor de normalización = 1/sqrt(10)
    datosMapeados = (1/sqrt(10))*qammod(dataSimb,M); % Modulación 16-QAM
    %qammod Modula la señal dataSimb en amplitud de cuadraturacon la
    %orden de modulación M, la salida es la señal modulada
    disp('Técnica de corrección de errores: Reed Solomon')
    disp('Tipo de Mapeo: 16-QAM')
elseif (seleccionMapeo==4)
    % Mapeo 64-QAM
    M = 64; % Orden de modulación
    Nb=log2(M); % Bits por símbolo, se usan 6 bits por símbolo
    % Hay que asegurarse que la longitud de los datos de entrada sea
    % múltiplo de Nb, por lo que se rellena de ceros cuando no es
    % múltiplo de Nb.
```

```
    % Se calcula la longitud de relleno y se almacena en long_padd0
    long_padd0=ceil(length(datosBinario)/Nb)*Nb-length(datosBinario);
    %ceil=redondea la longitud al entero inmediato superior
    paddQ=zeros(1,long_padd0); % Vector de relleno con ceros
    datosbin=[datosBinario paddQ]; % Vector con relleno
    %Pasar de binario a símbolos decimales:
    dataMatriz=reshape(datosbin,Nb,[]).'; % Matriz de Nb columnas y
    % cada fila corresponde a un símbolo
    dataSimb=bi2de(dataMatriz,'left-msb'); % Símbolos a modular
    % bi2de, cambio de binario a decimal
    %left-msb Indica la columna izquierda (o primera) de la salida binaria,
    %como el bit más significativo(o dígito de mayor orden)
    % Factor de normalización = 1/sqrt(42)
    datosMapeados = (1/sqrt(42))*(qammod(dataSimb,M)); % Datos
    % mapeados con 64-QAM
    %qammod Modula la señal dataSimb en amplitud de cuadraturacon la
    %orden de modulación M, la salida es la señal modulada
    disp('Técnica de corrección de errores: Reed Solomon')
    disp('Tipo de Mapeo: 64-QAM')
end

%% Añadiendo ruido AWGN
ruidoy=6;
datosMapeo = awgn(datosMapeados,ruidoy,'measured'); % Señal con ruido,
% tal que awgn(entrada,SNR,'measured')
%measured para que la función awgn mida la potencia de datosMapeados antes
%de agregar ruido.
% datosMapeo=datosMapeados; % Si se desea probar sin ruido
%% RECEPTOR
%% Demapeo de datos
% Con un lazo if se discrimina el demapeo a utilizar
if(seleccionMapeo==1)
    % Demapeo BPSK
    % Los datos filtrados no se normalizan
    longitud=length(datosMapeo); % Longitud datos de entrada
    datosDemod=zeros(longitud,1); % Inicializar vector
    % Implementar el Demapeo BPSK con el lazo for
    for i=1:longitud
        if(real(datosMapeo(i))>=0 && imag(datosMapeo(i))>=0)
            datosDemod(i)=1;
        elseif(real(datosMapeo(i))<0 && imag(datosMapeo(i))<0)
            datosDemod(i)=0;
        end
    end
elseif(seleccionMapeo==2)
    % Demapeo QPSK
    M = 4; % orden de modulación
    Nb=log2(M); % Bits por símbolo
    % Relleno de ceros:
    long_pM=ceil(length(datosMapeo)/Nb)*Nb-length(datosMapeo);
    %ceil=redondea la longitud al entero inmediato superior
    paddM=zeros(long_pM,1); % relleno con ceros
    % Los datos filtrados solo se multiplican por sqrt(2)
    ModDat=(sqrt(2))*[datosMapeo;paddM]; %Datos con relleno
    % Demodulación QPSK:
    datosDemodS=pskdemod(ModDat,M,pi/4,'gray');
    %pskmod modula por desplazamiento de fase dataSimb con un orden de
    %modulación M y una fase inicial igual a pi/4
    %Gray significa el orden de símbolos, que esta codificada en orden gray
```

```
    demodMatriz=de2bi(datosDemodS,Nb,'left-msb');
    %de2bi pasar de decimal a binario
    %left-msb Indica la columna izquierda (o primera) de la salida binaria,
    %como el bit más significativo(o dígito de mayor orden)
    datosDemod=reshape(demodMatriz.',1,[]).'; % Datos
    % recuperados (bits Serializados)
elseif(seleccionMapeo==3)
    % Demapeo 16-QAM
    M = 16; % Orden de modulación
    Nb = log2(M); % Bits por símbolo
    % Relleno de ceros:
    long_pM=ceil(length(datosMapeo)/Nb)*Nb-length(datosMapeo);
    %ceil=redondea la longitud al entero inmediato superior
    paddM=zeros(long_pM,1); % relleno con ceros
    % Los datos filtrados solo se multiplican por sqrt(10)
    ModDat=(sqrt(10))*[datosMapeo;paddM]; % Datos con relleno
    % Demodulación 16-QAM:
    datosDemodS=qamdemod(ModDat,M);
    %qamdemod, demodulación Qam de ModDat con un orden de modulación M
    % Pasar de símbolos a bits:
    demodMatriz=de2bi(datosDemodS,Nb,'left-msb');
    %left-msb Indica la columna izquierda (o primera) de la salida binaria,
    %como el bit más significativo(o dígito de mayor orden)
    datosDemod=reshape(demodMatriz.',1,[]).'; %Datos
    % recuperados (bits Serializados)
elseif(seleccionMapeo==4)
    % Demapeo 64-QAM
    M = 64; % Puntos en la constelación (símbolos)
    Nb = log2(M); % Bits por símbolo
    % Relleno de ceros:
    long_pM=ceil(length(datosMapeo)/Nb)*Nb-length(datosMapeo);
    %ceil=redondea la longitud de(datosMapeo)/Nb)*Nb-length(datosMapeo)
    paddM=zeros(long_pM,1); % relleno con ceros
    % Los datos filtrados solo se multiplican por sqrt(42)
    ModDat=(sqrt(42))*[datosMapeo;paddM]; % Datos con relleno
    % Demodulación 64-QAM:
    datosDemodS=qamdemod(ModDat,M);
    %qamdemod, demodulación Qam de ModDat con un orden de modulación M
    % Pasar de símbolos a bits:
    demodMatriz=de2bi(datosDemodS,Nb,'left-msb');
    %de2bi, pasar de decimal a binario
    %left-msb Indica la columna izquierda (o primera) de la salida binaria,
    %como el bit más significativo(o dígito de mayor orden)
    datosDemod=reshape(demodMatriz.',1,[]).'; %Datos
    % recuperados (bits Serializados)
end
%% Decodificador Reed-Solomon
% Relleno de ceros para hacer múltiplo de 8 y 5 (m y NRS)
k=ceil(length(datosDemod)/(m*NRS)); % Calcular el número de grupos
% de bits a decodificar
%ceil=redondea la longitud al inmediato superior entero
longiPadd=m*NRS*k-length(datosDemod); % Longitud de relleno
v_relleno=zeros(longiPadd,1); % relleno de ceros
datosRell=[datosDemod;v_relleno]; % Datos serializados con
% relleno de ceros
datlNbin=reshape(datosRell,m,[]).';
datDec=bi2de(datlNbin,'left-msb').'; % Datos decimales codificados
%bi2de, paso de binario a decimal
LdatCod=length(datDec);%obtengo longitud de datDec
```

```
datDecodReed=[];
for i=1:NRS:LdatCod
   auxD=datDec(i:i+NRS-1); % Tomar de NSR en NRS símbolos decim
   datGalois=gf(auxD,m);
   % Campo de Galois de la matriz auxDecimal con el tamaño de símbolo
   %para relleno de datos, m tiene que ser entero de 1 a 16
   decRS=rsdec(datGalois,NRS,KRS); % Decodificar
   %rsdec, decodificador Reed Solomon, lo hace con datGalois, con su
   %respectiva longitud del codificador y símbolos originales
   decRSd=double(decRS.x); % Extraer los datos decodificados del campo gf
   % y pasar a decimal
   %Símbolos a bits
   bitdRS=de2bi(decRSd.',m,'left-msb');
   %de2bi, paso de decimal a binario
   bitdRSs=reshape(bitdRS.',1,[]); % vector en bits serializado
   datDecodReed=[datDecodReed bitdRSs]; % bits serializados acumulados
   %Agrupar y poner en byte, reshape y obtener el texto
end

%% Medir BER
% Recortar datos de entrada para coincidir la longitud
msgErr=datDecodReed(1:length(msg)); %Ontengo los datos recibidos con la
%lontigud del mensaje original
Nerrores=biterr(msg,msgErr);
%biterr realiza el número de errores entre nuestra entrada msg y los datos
%recibidos, tasa de bits erroneos BER
fprintf('Bits Errados: %.0f/%.0f \n',Nerrores,length(msg))
fprintf('Cantidad de ruido AWGN: %.0f \n',ruidoy)
%% Recuperar Texto
   % Relleno de ceros:
   bPc=8; % Bits por cada caracter
   numC=ceil(length(datDecodReed)/8); %Número caracteres
   %ceil, obtengo el inmediato superior entero
   LongRellnumC=bPc*numC-length(datDecodReed); % Longitud
   % de relleno para hacer múltiplo de 8
   rellenoC=zeros(LongRellnumC,1); % Vector de relleno ceros
   datosSerializados=[datDecodReed ; rellenoC]; %Rellenar
   % en caso de ser necesario
   % Pasar de los bits serializados a una matriz, en donde las
   % filas contiene a los caracteres en binario:
   datosMtz=reshape(datosSerializados,[8 numC]).'; % numC filas
   % x 8 columnas
   % Pasar de binario a decimal, los datos de cada fila:
   datosTexto=[]; % Inicializar vector
   for i=1:numC
      % Se pasa cada fila a decimal y se almacena en datosTexto
      datosTexto(1,i)=double(bi2de(datosMtz(i,:),'left-msb'));
   end
   %Texto Original
   Texto_Original=transpose(datosAbiertos)
   % Texto recuperado
   % Pasar de decimal a caracteres:
   Texto_Recuperado=char(datosTexto) % Texto recuperado
   disp('Proceso recuperacion de texto terminado');
```

ANEXO III

Simulacion_Turbocodigo.m

```
%% Simulación Codificación Turbo código.
clc
clear all;
close all;
%% TRANSMISOR
%% Datos de entrada (texto)
msg_abc=uigetfile('*.txt;','Seleccionar archivo a transmitir'); %Selecciona
% el archivo .txt a codificar}
fID = fopen(msg_abc); % Abrir el archivo
   datosAbiertos=fread(fID,'*char'); % Leer los caracteres
msg=reshape(dec2bin(datosAbiertos,8).'-'0',1,[]);       %Transformo a binario
% msg = randi([0 1],1,40); %
% Para elegir el tipo de mapeo que se pretende utilizar la
% variable seleccionMapeo, en donde si se seleccionamos BPSK se debe poner
% el valor igual a 1, para QPSK igual a 2, para 16-QAM igual a 3 y
% para 64-QAM igual a 4
seleccionMapeo=1;
%% Codificador Turbo código
% Primero se define los valores a considerar.
mc=4; % m es el tamaño de registro de la memoria
g_p=[13 15 17]; % Polinomios generadores
Nbits=40; % número de bits serializados a codificar
% Creación de la estructura de Trellis:
P_Trellis = poly2trellis(mc, g_p);
% Creación del codificador para turbo códigos
intrlvrIndices = Nbits:-1:1; %Permutación de números enteros.
%Sentencia codificador Turbo código
turboenc = comm.TurboEncoder(P_Trellis,intrlvrIndices);
%comm.TurboEncoder= codifica la señal utilizando un esquema de codificación
%concatenado paralelo, se maneja con la estructura de Trellis e
%intrlvrIndices que es un índice de entrelazado especificado como un
%vector de columna de enteros
% Cálculo para rellenar el vector de entrada, y así obtener un vector
% que sea múltiplo de Nbits:
NumSimb=ceil(length(msg)/Nbits); % Número de simbolos OFDM
longitudPadd=Nbits*NumSimb-length(msg); % Longitud de relleno
vector_padd=zeros(1,longitudPadd); % vector de relleno de ceros
datosConPadd=[msg vector_padd]; % Datos serializados con relleno
% Codificador:
datosCodConv=[]; % se inicializa la variable que va a almacenar los
% datos codificados y serializados
% Se recorre todos los símbolos OFDM con un lazo for
for i=1:NumSimb
   auxDatosCc = datosConPadd(1,(Nbits*(i-1)+1):Nbits*i); % variable
   % auxiliar para almacenar los datos a codificar
   %Codificador Turbo código;
   CodConv = turboenc(transpose(auxDatosCc));
   datosCodConv=[datosCodConv;CodConv]; % Datos de salida
end
datosBinario=datosCodConv; % Copiar los datos de salida en una variable
%% Mapeo
% Con un if se discrimina el mapeo a utilizar
if(seleccionMapeo==1)
   % Mapeo BPSK
   % los datos de entrada es un vector columna
   modBPSK=comm.BPSKModulator(pi/4);
   %comm.BPSKModulator es una modulación usando método BPSK donde usamos
   %un valor de pi/4 para la fase del punto cero de constelación
   datosMapeados=modBPSK(datosBinario);
```

```
    disp('Técnica de corrección de errores: Turbo código')
    disp('Tipo de Mapeo: BPSK')
elseif(seleccionMapeo==2)
    % Mapeo QPSK
    M = 4; % orden de modulación
    Nb=log2(M); % Bits por símbolo, se utiliza 2 bits por símbolo
    % Hay que asegurarse que la longitud de los datos de entrada sea
    % múltiplo de Nb, por lo que se rellena de ceros cuando no es
    % múltiplo de Nb.
    % Se calcula la longitud de relleno y se almacena en long_padd0
    long_padd0=ceil(length(datosBinario)/Nb)*Nb-length(datosBinario);
    %ceil=redondea la longitud al entero inmediato superior
    paddQ=zeros(1,long_padd0); % Vector de relleno con ceros
    datosbin=[datosBinario paddQ]; % Vector con relleno
    %Pasar de binario a símbolos decimales:
    dataMatriz=reshape(datosbin,Nb,[]).'; % Matriz de Nb columnas y
    % cada fila corresponde a un símbolo
    dataSimb=bi2de(dataMatriz,'left-msb'); % Símbolos a modular
    % bi2de, transforma binario a decimal
    % Factor de normalización = 1/sqrt(2)
    datosMapeados = (1/sqrt(2))*pskmod(dataSimb,M,pi/4,'gray'); % Datos
    % mapeados con QPSK
    %pskmod=modulación por desplazamiento de fase de la entrada dataSimb
    %con una orden de modulación M, y una fase inial igual a pi/4
    %Gray significa el orden de símbolos, que esta codificada en orden gray
    disp('Técnica de corrección de errores: Turbo código')
    disp('Tipo de Mapeo: QPSK')
elseif (seleccionMapeo==3)
    % Mapeo 16-QAM
    M = 16; % Puntos en la constelación (símbolos)
    Nb=log2(M); % Bits por símbolo, se usan 4 bits por símbolo
    % Hay que asegurarse que la longitud de los datos de entrada sea
    % múltiplo de Nb, por lo que se rellena de ceros cuando no es
    % múltiplo de Nb.
    % Se calcula la longitud de relleno y se almacena en long_padd0
    long_padd0=ceil(length(datosBinario)/Nb)*Nb-length(datosBinario);
    %ceil=redondea la longitud al entero inmediato superior
    paddQ=zeros(long_padd0,1); % Vector de relleno con ceros
    datosbin=[datosBinario;paddQ]; % Vector con relleno
    %Pasar de binario a símbolos decimales:
    dataMatriz=reshape(datosbin,Nb,[]).'; % Matriz de Nb columnas y
    % cada fila corresponde a un símbolo
    dataSimb=bi2de(dataMatriz,'left-msb'); % Símbolos a modular
    % Modulador:
    % Factor de normalización = 1/sqrt(10)
    datosMapeados = (1/sqrt(10))*qammod(dataSimb,M); % Modulación 16-QAM
    %qammod=modula la señal dataSimb con QAM con el orden de modulación M
    disp('Técnica de corrección de errores: Turbo código')
    disp('Tipo de Mapeo: 16-QAM')
elseif (seleccionMapeo==4)
    % Mapeo 64-QAM
    M = 64; % orden de modulación
    Nb=log2(M); % Bits por símbolo, se usan 6 bits por símbolo
    % Hay que asegurarse que la longitud de los datos de entrada sea
    % múltiplo de Nb, por lo que se rellena de ceros cuando no es
    % múltiplo de Nb.
    % Se calcula la longitud de relleno y se almacena en long_padd0
    long_padd0=ceil(length(datosBinario)/Nb)*Nb-length(datosBinario);
    %ceil=redondea la longitud al entero inmediato superior
```

```
    paddQ=zeros(long_padd0,1); % Vector de relleno con ceros
    datosbin=[datosBinario;paddQ]; % Vector con relleno
    %Pasar de binario a símbolos decimales:
    dataMatriz=reshape(datosbin,Nb,[]).'; % Matriz de Nb columnas y
    % cada fila corresponde a un símbolo
    dataSimb=bi2de(dataMatriz,'left-msb'); % Símbolos a modular
    % bi2de, binario a decimal
    % Factor de normalización = 1/sqrt(42)
    datosMapeados = (1/sqrt(42))*(qammod(dataSimb,M)); % Datos
    % mapeados con 64-QAM
    %qammod=modula la señal dataSimb con QAM con el orden de modulación M
    disp('Técnica de corrección de errores: Turbo código')
    disp('Tipo de Mapeo: 64-QAM')
end
%% Añadiendo ruido AWGN
ruidoy=6;
datosMapeo = awgn(datosMapeados,ruidoy,'measured'); % Señal con ruido,
% tal que awgn(entrada,SNR,'measured')
%measured para que la función awgn mida la potencia de datosMapeados antes
%de agregar ruido.
% datosMapeo=datosMapeados; % Si se desea probar sin ruido
%% RECEPTOR
%% Demapeo de datos
% Con un lazo if se discrimina el demapeo a utilizar
if(seleccionMapeo==1)
    % Demapeo BPSK
    demodBPSK=comm.BPSKDemodulator(pi/4,'DecisionMethod','Log-likelihood ratio'); % Demodulador
    %comm.BPSKDemodulator es un demodulador usando método BPSK, donde se
    %maneja una fase del punto cero de constelación con valor de pi/4,
    %donde DecisionMethod para manejar los valores de simple o double y
    %Log-likelihood ratio se utiliza para especificar que es demodulación
    datosDemod=demodBPSK(datosMapeo); % Demodulación QPSK, bits suaves
    % determinados mediante un método LLR.
elseif(seleccionMapeo==2)
    % Demapeo QPSK
    ModDat=(sqrt(2))*datosMapeo; % Desnormalizar datos
    demodQPSK=comm.QPSKDemodulator('BitOutput',true,...
        'DecisionMethod','Log-likelihood ratio'); % Demodulados
    % Demodulación QPSK:
    %comm.QPSKDemodulator , demodulador con método QPSK, BitOutput
    %significa que los datos de salida salen
    %en forma de bits, al poner true genera un vector de columna de valores
    %de bits con una longitud igual al doble de la cantidad de
    %símbolos demodulados, DecisionMethod por default es hard y
    %Log-likelihood ratio para que demodule nuestro sistema
    datosDemod=demodQPSK(ModDat); % bits suaves determinados mediante
    %un método LLR
elseif(seleccionMapeo==3)
    % Demapeo 16-QAM
    M = 16; % Puntos en la constelación (símbolos)
    % Los datos filtrados solo se multiplican por sqrt(10)
    ModDat=(sqrt(10))*datosMapeo; % Datos con relleno
    % Demodulación 16-QAM:
    datosDemod=qamdemod(ModDat,M,'OutputType','llr');
    % Pasar de símbolos a bits:
    %qamdemod, demodula ModDat a una orden de modulación M, OutputType
    %especifica el tipo de salida en este caso LLR
elseif(seleccionMapeo==4)
    % Demapeo 64-QAM
```

```
    M = 64; % Puntos en la constelación (símbolos)
    ModDat=(sqrt(42))*datosMapeo; % Datos con relleno
    % Demodulación 64-QAM:
    datosDemod=qamdemod(ModDat,M,'OutputType','llr');
    %qamdemod, demodula ModDat a una orden de modulación M, OutputType
    %especifica el tipo de salida en este caso LLR
end
%% Decodificador Turbo código
% La estructura de trellis se definió en la parte de codificación (tx)
% Creación del decodificador para turbo códigos:
n = log2(P_Trellis.numOutputSymbols); %Obteniendo el logaritmo entre
%Trellis y el número de símbolos de salida del decodificador.
numTails = log2(P_Trellis.numStates)*n; %Obteniedo el logaritmo entre
%Trellis y el número de estados en el decodificador
L_sal = Nbits*(2*n - 1) + 2*numTails; %Longitud del paquete de palabra código de salida.
intrlvrIndices = Nbits:-1:1; %Permutación de números enteros.
numiter=4; %Asignando un valor de 4 para el decodificador
turbodec = comm.TurboDecoder(P_Trellis,intrlvrIndices,numiter); % Decodificador
%comm.TurboDecoder Decodificador Turbo código que utiliza la estructura
%de Trellis, intrlvrIndices es un índice de entrelazado que por defecto
%realiza vector columna de enteros, numiter es un valor de número de
%iteraciones
% Relleno de bits para que el vector de entrada sea multiplo de nBc:
nGrupos=ceil(length(datosDemod)/L_sal); % Calcular el número de grupos
% de bits a decodificar
%%ceil=redondea la longitud al entero inmediato superior
longiPadd=L_sal*nGrupos-length(datosDemod); % Longitud de relleno
v_relleno=zeros(longiPadd,1); % relleno de ceros
datosDemodRell=[datosDemod;v_relleno]; % Datos serializados con
% relleno de ceros
% Decodificación por turbo códigos
datosDecodConv=[]; % Inicilizar vector que almacena los datos
% decodificados
% Con un lazo for se recorre los grupos de bits a ser decodificados
for i=1:nGrupos
    auxDecod = datosDemodRell((L_sal*(i-1)+1):L_sal*i,1); % Se toma el
    % grupo correspondiente
    %Turbo código
    DecodVit = turbodec(-auxDecod); %Obtención de la decodificación
    datosDecodConv=[datosDecodConv;DecodVit]; % Almacenar los bits
    % decodificados
end
%% Medir BER
% Recortar datos de entrada para coincidir la longitud
msgErr=datosDecodConv(1:length(msg));%error de los datos decodificados con
%la longitud del msg de entrada
Nerrores=biterr(msg.',msgErr);
%biterr realiza el número de errores entre nuestra entrada msg y los datos
%recibidos, tasa de bits erroneos BER
fprintf('Bits Errados: %.0f/%.0f \n',Nerrores,length(msg))
fprintf('Cantidad de ruido AWGN: %.0f \n',ruidoy)
%% Recuperar Texto
    % Relleno de ceros:
    bPc=8; % Bits por cada caracter
    numC=ceil(length(msgErr)/8); %Número caracteres
    %ceil=redondea la longitud al entero inmediato superior
    LongRellnumC=bPc*numC-length(msgErr); % Longitud
    % de relleno para hacer múltiplo de 8
    rellenoC=zeros(LongRellnumC,1); % Vector de relleno ceros
```

```
datosSerializados=[msgErr ; rellenoC]; %Rellenar
% en caso de ser necesario
% Pasar de los bits serializados a una matriz, en donde las
% filas contiene a los caracteres en binario:
datosMtz=reshape(datosSerializados,[8 numC]).'; % numC filas
% x 8 columnas
% Pasar de binario a decimal, los datos de cada fila:
datosTexto=[]; % Inicializar vector
for i=1:numC
    % Se pasa cada fila a decimal y se almacena en datosTexto
    datosTexto(1,i)=double(bi2de(datosMtz(i,:),'left-msb'));
end
%Texto Original
Texto_Original=transpose(datosAbiertos)
% Texto recuperado
% Pasar de decimal a caracteres:
Texto_Recuperado=char(datosTexto) % Texto recuperado
disp('Proceso recuperacion de texto terminado');
```

ANEXO IV

configurarAdalmPluto.m

```
% Configurar equipos SDR Adalm Pluto para transmisión y recepción
% Parámetros de configuración:
% Propiedades comunes para tx (txAdalm) y rx (rxAdalm):
frecuenciaCentral=860e6; % Máximo alcance de las antenas del SDR
frecuenciaMuestreo=2000e3;% Valor donde se tiene menor BER
usbB='usb:0'; % ID del puerto USB donde está conectado el SDR
% Propiedades tx (txAdalm):
tamanioBloque=800000; %Tamaño de bloque a enviar en transmisión.
Gaintx=-2; % Ganancia en transmisión de -89.75 a 0 dB
%Estaba en -2
% Propiedades rx (rxAdalm):
GainSource='Manual'; % Fuente de ganancia manual
Gainrx=20;% ganancia en recepción de -4 a 71
%Estaba en 20
OutputDataType='single'; % Puede tomar los valores de
% int16(16bits) single(32bits) double(64bits)
SamplesPerFrame=tamanioBloque;%Número de muestras recibidas del transmisor
ShowAP=1; % mostrar las propiedades avanzadas
FrequencyCorrection=0; % Corrección de frec. en ppm (-200 a 200)
% Buscar dispositivo SDR:
% findPlutoRadio busca el RadioID y el número de serie del dispositivo
% Si no se está conectado un dispositivo la estructura esta vacía
encontrarSDR=findPlutoRadio;
numeroEncontrado=length(encontrarSDR); % 0 si no hay ningun dispositivo
numeroEncontrado=1;
if(numeroEncontrado==0) % No está conectado ningún dispositivo
    disp('Dispositivo no encontrado')
else  % El dispositivo esta conectado
    % Se configuran los objetos globales para tx y rx:
    % Objeto global para la tx:
    txAdalm=sdrtx('Pluto',...
        'RadioID',usbB,...
        'CenterFrequency',frecuenciaCentral,...
        'Gain',Gaintx,...
        'BasebandSampleRate',frecuenciaMuestreo)
    % Objeto global para la rx:
```

```
    rxAdalm = sdrrx('Pluto',...
        'RadioID',usbB,...
        'CenterFrequency',frecuenciaCentral,...
        'GainSource',GainSource,...
        'Gain',Gainrx,...
        'BasebandSampleRate',frecuenciaMuestreo,...
        'OutputDataType',OutputDataType,...
        'SamplesPerFrame',SamplesPerFrame,...
        'ShowAdvancedProperties',ShowAP,...
        'FrequencyCorrection',FrequencyCorrection)
    disp('Dispositivo configurado')
end
```

ANEXO V

transmisor_Convolucional.m

```
%% TRANSMISIÓN CONVOLUCIONAL
%% Parámetros para seleccionar algoritmos y controlar la transmisión
seleccionMapeo=3; % Variable para seleccionar el tipo de mapeo a utilizar,
% BPSK (1), QPSK (2), 16-QAM(3), 64-QAM(4)
correccionErrores=20; % Variable para activar (20) o desactivar (10) la
% corrección de errores
intTime=3; %Intervalo de tiempo en segundos, para esperar entre
% transmisiones
txRealTime=20; % Se setea la variable para ingresar al lazo while de
% transmisión repetitiva cada intTime segundos
%% Seleccionar archivo de texto y procesarlo para transmitir
nombreArchivo=uigetfile('*.txt;','Seleccionar archivo a transmitir');
% La variable nombreArchivo toma el valor de 0 cuando no se ha
% seleccionado ningun archivo y toma el nombre del archivo cuando
% si se ha seleccionado un archivo
if nombreArchivo==0
    % No se ha seleccionado ningun archivo
    disp('Archivo no seleccionado'); % Se utiliza para informar
else
    %% %%%%%%%%%%%%%%%%%% ABRIR ARCHIVO DE TEXTO
%%%%%%%%%%%%%%%%%%%%%%
    % Se ha seleccionado un archivo y se procede a abrir el mismo
    fID = fopen(nombreArchivo); % Abrir el archivo
    datosAbiertos=fread(fID,'*char'); % Leer los caracteres
    fclose(fID); % Cerrar el archivo
    %% %%%%%%%%%%%%%%%%%%%% SERIALIZAR DATOS
%%%%%%%%%%%%%%%%%%%%%%%%%%
    textoBinario = dec2bin(datosAbiertos,8); % representacion de cada
    % caracter con 8 bits
    % Para serializar los datos se utiliza dos lazos for(). El primer
    % lazo for() recorre cada fila, length(textoBinario) representa el
    % número de filas. El segundo lazo for() en cada fila recorre todos
    % los elementos (8 columnas). Y con una variable auxiliar (aux) se
    % almacena los datos en la posición que dicte la variable aux.
    aux=1; % Inicializar variable auxiliar
    for i = 1:length(textoBinario)
        for j=1:8
            textoBitsSerial(aux)=textoBinario(i,j); % tipo char
            aux=aux+1; % Sumar 1 en la variable auxiliar
        end
    end
```

```
% Para convertir los datos del tipo char al tipo str se utiliza un
% lazo for para recorrer todos los elementos del vector serializado
for aux=1:length(textoBitsSerial)
    % Se utiliza una variable auxiliar para almacenar los datos
    datosSerializados(aux)=str2double(textoBitsSerial(aux));
end
disp('Datos Serializados') % Mostrar un msj en el Command Window
%% %%%%%%%% CODIFICADOR CONVOLUCIONAL (Corrección de errores)%%%%%%%%%
datosBina=datosSerializados; % Copiar los datos serializados en una
% nueva variable
% Se utiliza un lazo condicional if() para determinar si se utiliza la
% correción de errores a nivel de bit
if correccionErrores==10
    % No se ha seleccionado la corrección de errores
    datosBinario=datosBina; % Se copia los datos serializados en la
    % variable de salida de esta etapa y por ende la entrada de la
    % siguiente etapa
elseif correccionErrores==20
    % Se ha seleccionado la corrección de errores.
    % Se define los valores a considerar de las etapas posteriores:
    NPdatos=40; % número de portadora de datos por símbolo OFDM
    % Dependiendo de la opción de mapeo es el orden de modulación
    if (seleccionMapeo==1)
        % BPSK
        M = 2; %2 para BPSK
    elseif (seleccionMapeo==2)
        % QPSK
        M = 4; %4 para QSPK
    elseif (seleccionMapeo==3)
        % 16-QAM
        M = 16; %16 para 16-QAM
    elseif (seleccionMapeo==4)
        % 64-QAM
        M = 64; %64 para 64-QAM
    end
    Nbps_M=log2(M); % Se calcula los bits por símbolo en según la
    % opción de mapeo que se ha escogido
    % Parámetros del codificador convolucional (2, 1, 3):
    nc=2; % n es la longitud por cada ingreso de k bits
    kc=1; % k es los bits de entrada
    mc=3; % m es el tamaño de registro de la memoria
    g1=5; % Polinomio generador x2+1=5(oct)
    g2=7; % Polinomio generador x2+x+1=7(oct)
    Rcc=kc/nc; % tasa del codificador convolucional
    % Cálculo del número de bits serializados que se tiene por cada símbolo
    % OFDM:
    Nbits=Nbps_M*NPdatos*Rcc;
    % Cálculo para rellenar el vector de entrada, y así obtener un vector
    % que sea múltiplo de Nbits:
    NumSimb=ceil(length(msg)/Nbits); % Número de simbolos OFDM
    %ceil=redondea la longitud al entero inmediato superior
    longitudPadd=Nbits*NumSimb-length(msg); % Longitud de relleno
    vector_padd=zeros(1,longitudPadd); % vector de relleno de ceros
    datosConPadd=[msg vector_padd]; % Datos serializados con relleno
    % Creción de la estructura de Trellis:
    P_Trellis = poly2trellis(mc, [g1 g2]);
    % Codificador:
    datosCodConv=[]; % se inicializa la variable que va a almacenar los
    % datos codificados y serializados
```

```
        % Se recorre los símbolos OFDM con un lazo for
        for i=1:NumSimb
            auxDatosCc = datosConPadd(1,(Nbits*(i-1)+1):Nbits*i); % variable
            % auxiliar para almacenar los datos a codificar
            % Codificación convolucional:
            CodConv = convenc(auxDatosCc, P_Trellis);
            %Codifica auxDatosCc con la estructura de trellis P_Trellis
            datosCodConv=[datosCodConv CodConv]; % Datos de salida
        end
        datosBinario=datosCodConv; % Copiar los datos codificados en la
        % variable de salida
        disp('Codificación Convolucional Agregada') % Mensaje informativo
    end
    %% %%%%%%%%%%%%%%%%%%%%%%%%%%%% MAPEO
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % El vector de entrada de esta etapa es datosBinario y el de salida es
    % datosMapeados
    % El valor en la variable global seleccionMapeo define el tipo de
    % mapeo a utilizar BPSK (1), QPSK (2), 16-QAM(3) o 64-QAM(4)
    if(seleccionMapeo==1)
    % Mapeo BPSK
    datosColumna=datosBinario.'; % Se transpone la matriz que contiene
    % los datos de entrada para obtener un vector columna
    % Se recorre los cada bit de entrada y se asigna un símbolo
    for i=1:length(datosColumna)
        % Con el if discrimino si es igual a 1 y se asigna el
        % valor 0.707+0.707i, de no ser así es 0 y se asigna el
        % valor de -0.707-0.707i
        if(datosColumna(i)==1)
            datosMapeados(i,1)=complex(0.707,0.707);
        else
            datosMapeados(i,1)=complex(-0.707,-0.707);
        end
    end
    disp('Técnica de corrección de errores: Convolucional')
    disp('Tipo de mapeo: BPSK')
elseif(seleccionMapeo==2)
    % Mapeo QPSK
    M = 4; % Orden de modulación
    Nb=log2(M); % Bits por símbolo, se utiliza 2 bits por símbolo
    % Hay que asegurarse que la longitud de los datos de entrada sea
    % múltiplo de Nb, por lo que se rellena de ceros cuando no es
    % múltiplo de Nb.
    % Se calcula la longitud de relleno y se almacena en long_padd0
    long_padd0=ceil(length(datosBinario)/Nb)*Nb-length(datosBinario);
    %ceil=redondea la longitud al entero inmediato superior
    paddQ=zeros(1,long_padd0); % Vector de relleno con ceros
    datosbin=[datosBinario paddQ]; % Vector con relleno
    %Pasar de binario a símbolos decimales:
    dataMatriz=reshape(datosbin,Nb,[]).'; % Matriz de Nb columnas y
    % cada fila corresponde a un símbolo
    dataSimb=bi2de(dataMatriz,'left-msb'); % Símbolos a modular
    %bi2de convertir de binario a decimal
    %left-msb Indica la columna izquierda (o primera) de la salida binaria,
    %como el bit más significativo(o dígito de mayor orden)
    % Modulador:
    % Factor de normalización = 1/sqrt(2)
    datosMapeados = (1/sqrt(2))*pskmod(dataSimb,M,pi/4,'gray'); % Datos
    % mapeados con QPSK
```

```
    %pskmod=modulación por desplazamiento de fase de la entrada dataSimb
    %con una orden de modulación M, y una fase inial igual a pi/4
    %Gray significa el orden de símbolos, que esta codificada en orden gray
    disp('Técnica de corrección de errores: Convolucional')
    disp('Tipo de mapeo: QPSK')
elseif (seleccionMapeo==3)
    % Mapeo 16-QAM
    M = 16; % orden de modulación
    Nb=log2(M); % Bits por símbolo, se usan 4 bits por símbolo
    % Hay que asegurarse que la longitud de los datos de entrada sea
    % múltiplo de Nb, por lo que se rellena de ceros cuando no es
    % múltiplo de Nb.
    % Se calcula la longitud de relleno y se almacena en long_padd0
    long_padd0=ceil(length(datosBinario)/Nb)*Nb-length(datosBinario);
    %ceil=redondea la longitud al entero inmediato superior
    paddQ=zeros(1,long_padd0); % Vector de relleno con ceros
    datosbin=[datosBinario paddQ]; % Vector con relleno
    %Pasar de binario a símbolos decimales:
    dataMatriz=reshape(datosbin,Nb,[]).'; % Matriz de Nb columnas y
    % cada fila corresponde a un símbolo
    dataSimb=bi2de(dataMatriz,'left-msb'); % Símbolos a modular
    %bi2de convertir de binario a decimal
    %left-msb Indica la columna izquierda (o primera) de la salida binaria,
    %como el bit más significativo(o dígito de mayor orden).
    % Modulador:
    % Factor de normalización = 1/sqrt(10)
    datosMapeados = (1/sqrt(10))*qammod(dataSimb,M); % Modulación 16-QAM
    %qammod=modula la señal dataSimb con QAM con el orden de modulación M
    disp('Técnica de corrección de errores: Convolucional')
    disp('Tipo de mapeo: 16-QAM')
elseif (seleccionMapeo==4)
    % Mapeo 64-QAM
    M = 64; % Puntos en la constelación (símbolos),orden de modulación
    Nb=log2(M); % Bits por símbolo, se usan 6 bits por símbolo
    % Hay que asegurarse que la longitud de los datos de entrada sea
    % múltiplo de Nb, por lo que se rellena de ceros cuando no es
    % múltiplo de Nb.
    % Se calcula la longitud de relleno y se almacena en long_padd0
    long_padd0=ceil(length(datosBinario)/Nb)*Nb-length(datosBinario);
    %ceil=redondea la longitud al entero inmediato superior
    paddQ=zeros(1,long_padd0); % Vector de relleno con ceros
    datosbin=[datosBinario paddQ]; % Vector con relleno
    %Pasar de binario a símbolos decimales:
    dataMatriz=reshape(datosbin,Nb,[]).'; % Matriz de Nb columnas y
    % cada fila corresponde a un símbolo
    dataSimb=bi2de(dataMatriz,'left-msb'); % Símbolos a modular
    %bi2de convertir de binario a decimal
    %left-msb Indica la columna izquierda (o primera) de la salida binaria,
    %como el bit más significativo(o dígito de mayor orden).
    % Modulador:
    % Factor de normalización = 1/sqrt(42)
    datosMapeados = (1/sqrt(42))*(qammod(dataSimb,M)); % Datos
    % mapeados con 64-QAM
    %qammod=modula la señal dataSimb con QAM con el orden de modulación M
    disp('Técnica de corrección de errores: Convolucional')
    disp('Tipo de mapeo: 64-QAM')
end
    %% %%%%%%%%%%%%%%%%%%%%%%%% CREAR SIMBOLO OFDM
%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% El vector de entrada de esta etapa es datosMapeados, el cual es un
% vector tipo columna.
% El vector de salida es simbolosOFDMCreados
% La estructura del símbolo OFDM consta de una subportadora DC,
% 4 pilotos, 19 de guarda y 40 de datos.
nPdatos=40; % Número de portadoras de datos
% El vector de entrada debe ser múltiplo de nPdatos, cuando no se
% se cumple dicha condición se realiza un relleno con los datos del
% inicio al final de vector. Esto con el objetivo de hacer que el
% vector sea múltiplo de nPdatos:
longDatos=length(datosMapeados); % Longitud del vector de entrada
nSimbOFDM=ceil(longDatos/nPdatos); % Calcular el número de símbolos
longPadd=nSimbOFDM*nPdatos-longDatos; % Calcular la longitud del
% relleno
datosEntradaPadd=datosMapeados; % Copio los datos de entrada en una
% nueva variable auxiliar
% Si la longitud de relleno es cero no se realiza ninguna operación
% adicional, si es mayor que cero se realiza el relleno
if(longPadd>0)
   % Se hace el relleno con los datos del inicio del vector de entrada
   datosEntradaPadd(end+1:end+longPadd)=datosMapeados(1:longPadd);
end
% Se procede a crear los simbolos OFDM recorriendo en múltiplos de
% nPdatos, mediante un lazo for:
numeroSimbolos=1; % Inicializo la variable que sirve para almacenar los
% símbolos OFDM creados, en una matriz.
longDatosPadd=length(datosEntradaPadd); % Longitud de los datos de
% entrada con relleno
for i=1:nPdatos:longDatosPadd
   datosAux=datosEntradaPadd(i:i+nPdatos-1); %Copio los datos nPdatos,
   % es decir los 40 datos a introducirse dentro del símbolo OFDM
   DC=0;% Valor para la subportadora DC
   % Las variables p1, p2, p3, p4 son los valores que toman las
   % subportadoras piloto, usadas para poder hacer una estimación del
   % canal en recepción
   p1=1;
   p2=1;
   p3=1;
   p4=1;
   simbOFDM=zeros(64,1); % Inicializo símbolo OFDM, con todos los
   % valores en cero
   simbOFDM(1)=DC; % Portadora DC
   % Asignación del valor a cada subportadora de datos, y a las
   % subportadoras piloto:
   simbOFDM(2:7)=datosAux(21:26);
   simbOFDM(8)=p1; % Piloto
   simbOFDM(9:21)=datosAux(27:39);
   simbOFDM(22)=p2; % Piloto
   simbOFDM(23)=datosAux(40);
   % simbOFDM de 24 a 42 son de guarda
   simbOFDM(43)=datosAux(1);
   simbOFDM(44)=p3;% Piloto
   simbOFDM(45:57)=datosAux(2:14);
   simbOFDM(58)=p4; % Piloto
   simbOFDM(59:64)=datosAux(15:20);
   simbolosOFDMCreados(1:64,numeroSimbolos)=simbOFDM; % Matriz de
   % salida que almacena el símbolo OFDM completo, según la columna
   % que dicte la variable numeroSimbolo
   numeroSimbolos=numeroSimbolos+1; % Aumento en uno la variable
```

```
    end
    numeroSimbolos=numeroSimbolos-1; % Disminuyo en uno el valor de la
    % variable para que concuerde con la cantidad total de símbolos OFDM
    disp('Símbolos OFDM creados');
    %% %%%%%%%%%%%%%%%%%%%%%%%%%%%% IFFT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % En esta sección se realiza la modulación de los símbolos OFDM
    % mediante la ifft. La matriz de entrada es simbolosOFDMCreados y la de
    % salida es simbolosOFDM.
    % La función ifft admite como entrada un vector columna, pero tambien
    % una matriz. En la matriz aplica la ifft a cada columna. Como la
    % matriz de entrada contiene un spimbolo en cada columna se puede
    % aplicar directamente la función ifft
    simbolosOFDM=ifft(simbolosOFDMCreados); % matriz de salida
    disp('IFFT implementada');
    %% %%%%%%%%%%%%%%%%%%%%%%%%%%%% AÑADIR PC
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % El prefijo cíclico (PC) se añade copiando la parte final de cada
    % símbolo, al inicio del mismo.
    % La matriz de entrada es simbolosOFDM y a la salida se tiene una
    % matriz llamada simbolosOFDMpc con los símbolos en cada columna y
    % un vector contodos los símbolos serializados llamado datosEnviar
    % Con un lazo for se recorre cada símbolo modulado, se añade el PC
    % y se serializa los datos
    datosEnviar=[]; % Variable que almacenara los datos serializados
    for auxPC=1:numeroSimbolos
        % Añadir el prefijo cíclico, de tamaño igual a la cuarta parte del
        % símbolo:
        simbolosOFDMpc(1:80,auxPC)=[simbolosOFDM(49:64,auxPC);simbolosOFDM(:,auxPC)];
        datosEnviar=[datosEnviar;simbolosOFDMpc(:,auxPC)]; % Datos
        % serializados
    end
    disp('Prefijo cíclico (PC) añadido');
    %% %%%%%%%%%%%%%%%%%%%%%%%%%%%% AÑADIR PREAMBULO
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Añadimos el preambulo dado por el estandar 802.11a antes de los
    % símbolos OFDM serializados que contiene datosEnviar
    % Se inicia por definir parte de la sección corta y larga en el dominio
    % de la frecuencia para luego obtener el preambulo en el dominio del
    % tiempo:
    % 4 secuencias cortas (parte de LSTF) en el dominio de la frecuencia
    parteCorta64=[0;0;0;0;-1.4720-1.4720i;0;0;0
        -1.4720-1.4720i;0;0;0;1.4720+1.4720i;0;0;0
        1.4720+1.4720i;0;0;0;1.4720+1.4720i;0;0;0
        1.4720+1.4720i;0;0;0;0;0;0;0;0;0;0;0;0;0;0
        1.4720+1.4720i;0;0;0;-1.4720-1.4720i;0;0;0
        1.4720+1.4720i;0;0;0;-1.4720-1.4720i;0;0;0
        -1.4720-1.4720i;0;0;0;1.4720+1.4720i;0;0;0];
    % Secuencia larga (parte de LLTF) en el dominio de la frecuencia
    parteLarga=[...
        0;1;-1;-1;1;1;-1;1;-1;1;-1;-1;-1;-1;-1;1
        1;-1;-1;1;-1;1;-1;1;1;1;1;0;0;0;0;0
        0;0;0;0;0;0;1;1;-1;-1;1;1;-1;1;-1;1
        1;1;1;1;1;-1;-1;1;1;-1;1;-1;1;1;1;1];
    % Conversión del dominio de la frecuencia al tiempo, y creación de la
    % sección corta completa (LSTF):
    parteCorta64Time=ifft(parteCorta64); % pasar al dominio del tiempo
    parteCorta80Time=parteCorta64Time(49:64);% copiar las ultimas
    % 16 muestras al inicio de la variable
```

```
    parteCorta80Time(17:80)=parteCorta64Time; % 5 secuencias cortas
    preambulo=parteCorta80Time; % copiar las 5 secuencias cortas
    preambulo(81:160)=parteCorta80Time; % añadir las otras 5 secuencias
    % cortas para obtener la sección corta (LSTF)
    % Conversión del dominio de la frecuencia al tiempo, y adición de PC
    % de la sección larga:
    parteLargaTime=ifft(parteLarga); % conversión al dominio del tiempo la
    % secuencia larga
    preambulo(161:192)=parteLargaTime(33:64); % PC en el preámbulo
    preambulo(193:256)=parteLargaTime; % primera secuencia larga
    preambulo(257:320)=parteLargaTime; % segunda secuecia larga y se obtiene
    % el preámbulo completo.
    % Unir el preambulo a los símbolos OFDM serializados:
    datosConPre=[preambulo;datosEnviar]; % Preambulo con símbolos OFDM
    % creados previamente
    disp('Preámbulo 802.11a añadido');
    % Inicializar con ceros una variable auxiliar, que tenga el tamaño del
    % bloque a enviar:
    auxDat=zeros(tamanioBloque,1); % inicializar variable
    [longDat,~]=size(datosConPre); % longitud de los datos a enviar
    auxDat(1:longDat,1)=datosConPre; % copiar el preambulo con los símbolos
    % OFDM y los demás datos del bloque se mantienen en cero
    datosEnviarBloque=auxDat;% Datos de salida a enviar
    disp('Datos Procesados')
    %% %%%%%%%%%%%%%%%% NORMALIZAR LOS DATOS A TRANSMITIR %%%%%%%%%%%%%%%%
    % Normalizar los datos a transmitir:
    % Se evalua la parte real y la parte imaginaria de los vectores, para
    % encontrar el módulo máximo de la parte real e imaginaria.
    v_maxR=max(abs(real(datosEnviarBloque))); % Valor máximo, parte real
    v_maxI=max(abs(imag(datosEnviarBloque))); % Valor máximo, parte imag
    % Con un condicional se evalúa si la parte imaginaria o la parte real
    % tiene el valor máximo para normalizar los datos
    if(v_maxR<v_maxI)
        valorMax=v_maxI; % La parte imaginaria tiene el máximo valor abs
    else
        valorMax=v_maxR; % La parte real tiene el máximo valor absoluto
    end
    % Normalizo y almaceno los datos a transmitir en una variable auxiliar
    aux_TxNorm=datosEnviarBloque/valorMax;
    % Convertimos los datos a un tipo single que emplea 32 bits, mientras
    % que double emplea 64 bits para la representacion de cada valor:
    auxTx=single(aux_TxNorm);
    %% %%%%%%%%%%%%%%%%%%%%%%%% TRANSMITIR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Se ingresa al lazo para transmitir cada intervalo intTime:
    while (txRealTime==20)
        % se procede a transmitir el bloque de datos OFDM utilizando el
        % objeto de transmisión txAdalm()
        txAdalm(auxTx(:,1));
        % mostramos un mensaje de transmision terminada
        disp('Datos transmitidos...');
        pause(intTime)
    end
end
```

ANEXO VI

receptor_Convolucional.m

```
%% RECEPCIÓN CONVOLUCIONAL
%% Parámetros para seleccionar algoritmos y controlar la recepción
seleccionMapeo=4; % Variable para seleccionar el tipo de mapeo a utilizar,
% BPSK (1), QPSK (2), 16-QAM(3), 64-QAM(4)
correccionErrores=20; % Variable para activar (20) o desactivar (10) la
% corrección de errores
umbralRx=0.0040; % Umbral para la recepción, tiene que ser mayor a 0.0026
intTimeR=10; %Intervalo de tiempo en segundos, para esperar cuando se haya
% recuperado el texto
rxRealT=20; % Se setea la variable para ingresar al lazo while de recepción
% en tiempo real
%% %%%%%%%%%%%%% SELECCIÓN DE ARCHIVO PARA MEDIR BER
%%%%%%%%%%%%%%%%%%
nombreArchivo=uigetfile('*.txt;','Seleccionar archivo a transmitir');
% La variable nombreArchivo toma el valor de 0 cuando no se ha
% seleccionado ningun archivo y toma el nombre del archivo cuando
% si se ha seleccionado un archivo
if nombreArchivo==0
   % No se ha seleccionado ningun archivo
   disp('Archivo no seleccionado'); % Se utiliza para informar
   actBER=10; % Variable para activar o desactivar(10) la medición del BER
else
   % Se ha seleccionado un archivo y se procede a abrir el mismo
   fid =fopen(nombreArchivo);
   x=fread(fid,'*char'); % Datos Originales
   fclose(fid);
   textoBinario = dec2bin(x,8);% representar cada caracter con 8 bits
   % Serializar:
   tamanioTexto=length(textoBinario);
   aux=1;
   for i = 1:tamanioTexto
      for j=1:8
         textoBitsSerializado(aux)=textoBinario(i,j);
         aux=aux+1;
      end
   end
   datosSerializadosTxOrig=textoBitsSerializado;
   % De char a double:
   for aux=1:length(datosSerializadosTxOrig)
      % Datos originales en bits:
      datosMedirBER(aux)=str2double(datosSerializadosTxOrig(aux));
   end
   longDatosBER=length(datosMedirBER); % longitud de los datos abiertos
   % para obtener el BER
   actBER=20; % Variable para activar(20) o desactivar la medición del BER
end
%% Recepción en tiempo real:
% En el lazo while se implementa todos los procesos de recepción en tiempo
% real
while (rxRealT==20)
   clearvars datosRecibidos; % Limpiar la variable
   %% %%%%%%%%%%%%%%%%%%%%%%    RECIBIR DATOS
%%%%%%%%%%%%%%%%%%%%%%%%%
   for i=0:6
      vaciar=rxAdalm(); % Este vector no se utiliza, solo muestrea
      % el canal varias veces para que en caso de existir algún
      %buffer este se elimine
   end
   condAux=0; % Condición auxiliar, para estar dentro del while
```

```
%Recepcion de los datos:
while(condAux==0)
    datosRxComplejos= rxAdalm(); % Guarda la señal recuperada del
    % canal inalámbrico
    datosRecibidos=double(datosRxComplejos); % de single a double
    v_max=max(abs(datosRecibidos)); % Máximo valor de la magnitud
    % de cada dato recibido
    % Imprimir la magnitu máxima:
    fprintf('Valor máximo de las muestras: %8.5f\n',v_max)
    % Verificar si se supera el umbral establecido:
    if(v_max>umbralRx)
        datosRxAux=rxAdalm(); % Guarda la señal recuperada
        datosRec2=double(datosRxAux); % De single a double
        % Verificar si se supera el umbral:
        if (max(abs(datosRec2))>umbralRx)
            % Guardar en el mismo vector
            datosRecibidos=[datosRecibidos;datosRec2];
        end
        condAux=1; % Cambiar el valor para salir del lazo while
    end
end % Final del bucle while
disp('Datos recibidos del canal inalámbrico')
%%    %%%%%%%%%%%%%%%%%%%%%%    RECORTAR DATOS
%%%%%%%%%%%%%%%%%%%%%%%%%%
tic % Para medir el tiempo de procesamiento
% Se recorre el vector de entrada y  se obtiene la posición del
% primer valor por arriba del umbral:
pos_inic=0; % Inicializar variable
for i=1:length(datosRecibidos)
    if(max(abs(datosRecibidos(i)))>umbralRx)
        pos_inic=i; % Primera posición arriba del umbral
        break;
    end
end
% Guardar desde 400 datos antes de superar el umbral, hasta el final:
if(pos_inic>400)
    datosRec=datosRecibidos(pos_inic-400:end); % Datos recortados
else
    % El umbral se superó antes de las primeras 400 posiciones, solo
    % se copia los datos
    datosRec=datosRecibidos;
end
datosInv=flip(datosRec); % Invierte el vector, tal que los datos
% finales queden al inicio, con esto se puede repetir el proceso
% anterior.
% Se recorre el vector de invertido y  se obtiene la posición del
% primer valor por arriba del umbral:
pos_fin=0; % Inicializar variable
for i=1:length(datosRec)
    if (max(abs(datosInv(i)))>umbralRx)
        pos_fin=i; % Primera posición arriba del umbral
        break
    end
end
% Guardar desde 200 datos antes de superar el umbral, hasta el final:
if(pos_fin>200)
    datosInvRec=datosInv(pos_fin-200:end); % Recortar datos
else
    % El umbral se superó antes de las primeras 200 posiciones, solo
```

```
    % se copia los datos
    datosInvRec=datosInv;
end
datosRecortados=flip(datosInvRec); % Invertir nuevamente el vector
disp('Datos recortados')
%% %%%%%%%%%%%%%%%%%%%%    DETECCIÓN TRAMA OFDM     %%%%%%%%%%%%%%%%%
datosRX=datosRecortados; % Copiar los datos en una nueva variable
if length(datosRX)>(SamplesPerFrame+600)
    disp('***** Muestras recibidas inválidas *****');
    continue
end
% El preambulo se encuentra al inicio de cada paquete y por tanto
% se trabaja con las primeras muestras, con esto se evita
% procesamiento innecesario
Mmues=1400; % Muestras para calcular la métrica
if(length(datosRX)>Mmues)
    r=datosRX(1:Mmues);
else
    % Si es menor a Mmues se trabaja con todas las muestras
    r=datosRX;
end
L = 48; % Múltiplo dela longitud de una secuencia corta (16*x)
inMax = length(r)-2*L+1; % k+m puede tomar valores de hasta 2L-1
% el indice maximo a evaluar es la longitud de r menos 2L.
m=L-1; % Máximo valor de m
M = zeros(Mmues,1); % Inicializar la variable para mayor velocidad
% Determinar la métrica M(k) para comprobar si es un paquete OFDM:
for k=1:inMax
    Numerador = (r(k:k+m)'*r(k+L:k+m+L)); % Sumatorio del numerador
    Denom = sum((abs(r(k+L:k+m+L))).^2); % Sumatorio del denominador
    M(k)=Numerador/Denom; % Métrica
end
M2=(abs(M)).^2; % Cuadrado de la magnitud de cada valor
umbral=0.60; % Umbral para estimar si existio un paquete OFDM, debe
% ser mayor a 0.40
tramaEncontrada=0; % 0 trama no encontrada
auxET=0; % Inicializar variable que almacena la posición en donde
% se superera el umbral de la métrica
for i=1:length(M2)
    %Se obtiene la posición del primer valor por arriba del umbral:
    if(M2(i)> umbral)
        auxET=i;
        tramaEncontrada=1; % Trama encontrada
        break;
    end
end
datosOFDM=[]; % Iniciar variable que contiene la trama OFDM
if(tramaEncontrada==1)
    % Para el caso de que se haya encontrado una paquete OFDM
    datosOFDM(:,1)=datosRX(auxET:end);
else
    % Si no existe un paquete OFDM se empieza de nuevo el bucle while
    disp('***** Trama OFDM NO encontrada *****');
    pause(0.1)
    continue
end
% datosOFDM son los datos de salida
disp('Trama OFDM detectada');
```

```
    %% %%%%%%%%%%%%%%%%%%%%    CORRECCIÓN CFO
%%%%%%%%%%%%%%%%%%%%%%%%%%
    r = datosOFDM; % Copiar los datos en r para coincidir con la ecuación
    Fs=frecuenciaMuestreo; % Frecuencia de muestreo de los datos recibidos
    N = 128; % máximo valor del indice k
    L=16; % longitud d euna secuencia corta
    m=L-1; % Máxima longitud que puede tomar m
    fEst = zeros(N,1); % Iniciar vector que almacena la estimación de freq
    % Lazo que recorre de k igual a 1 hasta N
    for k=1:N
        P = r(k:k+m)'*r(k+L:k+m+L); % Sumarotio p(k)
        fEst(k) = (Fs/(2*pi*L))*(angle(P)); % Calcular la estimación
        % de frecuencia en la posición k
    end
    estFreq=-mean(fEst); % Negativo del promedio del CFO estimado
    %Corrección de frecuencia con la estimación realizada:
    despFreqObj = comm.PhaseFrequencyOffset('FrequencyOffset',...
        estFreq,'SampleRate',Fs);
    % Aplicar la estimacion de CFO a las muestras recibidas para
    % corregir el CFO:
    datosCorrCFO=despFreqObj(datosOFDM); % Datos corregidos el CFO
    %datosCorrCFO=datosOFDM; % Esta linea se usa para evitar corregir en
    %frecuencia
    disp('Correccion CFO terminada');
    %% %%%%%%%%%%%%%%%%%%%% SINCRONIZACIÓN DE SÍMBOLO
%%%%%%%%%%%%%%%%%%
    if length(datosCorrCFO)>5000
        datSinCFO=datosCorrCFO(1:5000); % Se toma dolos primeros 5000 datos
    % para reducir el tiempo de procesamiento
    else
        datSinCFO=datosCorrCFO;
    end
    % Crear la parte larga del preámbulo 802.11a:
    % Secuencia larga (parte de LLTF) en el dominio de la frecuencia
    parteLarga=[...
        0;1;-1;-1;1;1;-1;1;-1;1;-1;-1;-1;-1;-1;1
        1;-1;-1;1;-1;1;-1;1;1;1;1;0;0;0;0;0
        0;0;0;0;0;0;1;1;-1;-1;1;1;-1;1;-1;1
        1;1;1;1;1;-1;-1;1;1;-1;1;-1;1;1;1;1];
    % Conversión del dominio de la frecuencia al tiempo, y adición de PC
    % de la sección larga:
    parteLargaTime=ifft(parteLarga); % conversión al dominio del tiempo
    LLTF=[parteLargaTime(33:64);parteLargaTime;parteLargaTime]; % LLTF
    secLLTF80=LLTF(1:80); % Primeras 80 muestras de la LLTF
    % Cuadrado del valor absoluto de la orrelacíón cruzada entre
    % la señal recibida y secLLTF80:
    datosCorrRell=(abs(xcorr(datSinCFO,secLLTF80))).^2;
    % Se corrige en tiempo los datos correlacionados, para que
    % coincidan con las posiciones de los datos de entrada:
    datosCorr=datosCorrRell(length(datSinCFO):end);
    % Almacenar la posición de los dos picos:
    [~,posPico1]=max(datosCorr); % Posición del primer pico
    datosCorr(posPico1)=0; % Eliminar el primer pico para poder
    % encontrar el segundo
    [~,posPico2]=max(datosCorr); % Posición del segundo pico
    % Se comprueba que la distancia entre los dos picos sea 64
    if abs(posPico2-posPico1)==64
        % Se encuentra el inicio real del LTTF y de LSTF
        inicioLLTF=min([posPico1 posPico2]); % Se toma la posición
```

```
        % menor y esa es la ubicación del primer dato del LLTF
        inicioLSTF=inicioLLTF-160; % Determinar el inicio del preambulo
        % Comprobar que la posición inicial sea mayor a 0
        if (inicioLSTF>0)
            % Recortar los datos
            datosSincronizados=datosCorrCFO(inicioLSTF:end); % Datos
            % de salida
        else
            disp('Falla en la sincronizacion de simbolo (2)')
            continue
        end
    else
        disp('Falla de sincronización de símbolo')
        continue
    end
    % datosSincronizados es los datos de salida
    disp('Sincronización de símbolo terminado');
    %% %%%%%%%%%%%%%%%%%%%%    CORRECCIÓN DE FASE
%%%%%%%%%%%%%%%%%%%%%%%%
    % Pasar las dos secuencias largas del LLTF del dominio del tiempo al
    % dominio de la frecuencia:
    a=fft(datosSincronizados(257:320)); % segunda sección larga de LLTF
    b=fft(datosSincronizados(193:256)); % Primera sección larga de LLTF
    %  Valores de las secuencias largas a calcular el ángulo:
    angulosAux=[a(2) a(5) a(6) a(8) a(10) a(16) a(17) a(20) ...
        a(53) a(56) a(57) a(59) a(61) a(62) a(63) a(64) ...
        b(2) b(5) b(6) b(8) b(10) b(16) b(17) b(20) ...
        b(53) b(56) b(57) b(59) b(61) b(62) b(63) b(64)];
    angulos=(angle(angulosAux)*180)/pi; % Calcular el ángulo y pasar de
    % radianes a grados
    anguloProm=-mean(angulos); % Negativo del promedio de los ángulos
        % Corregir el desfase
    desFaseObj = comm.PhaseFrequencyOffset('PhaseOffset',anguloProm,...
        'SampleRate',Fs);
    datFase=desFaseObj(datosSincronizados);
    % Se verifica que los datos corregidos sean multiplo de 80,
    % caso contrario se realiza un padding de las ultimas muestras
    longDatF=length(datFase);
    padd80=ceil(longDatF/80)*80-longDatF;
    if(padd80>=1)
        datosCorrFase=[datFase;datFase(end-padd80+1:end)];
    else
        datosCorrFase=datFase;
    end
    % El vector de salida es datosCorrFase
    disp('Corrección de fase terminado');
    %% %%%%%%%%%%%%%%%%%%%%%%%%    ELIMINAR PC
%%%%%%%%%%%%%%%%%%%%%%%%
    clearvars simbolosSinPC;
    %Eliminacion del Prefijo ciclico
    nSimOFDM=0;
    tramaDatos=datosCorrFase(321:end,1); %Quitar el preambulo
    for j=1:80:length(tramaDatos)
        nSimOFDM=nSimOFDM+1;
        simbolosSinPC(:,nSimOFDM)=tramaDatos(j+16:j+79);
    end
    disp('Proceso Eliminación PC terminado');
    %% %%%%%%%%%%%%%%%%%%%%%%%%    FFT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Demodulación OFDM (FFT)
% La función fft aplica la DFT a cada columna de la matriz
% simbolosSinPC
simbolosfft=fft(simbolosSinPC); % Demodular los símbolos sin PC
disp('Proceso FFT terminado');
%% %%%%%%%%%%    ESTIMACIÓN DE CANAL Y ECUALIZADOR  %%%%%%%%%%%%%%
umbralF=2; % Definir umbral
simbolosEcualizados=[];
for auxnSimb=1:nSimOFDM
   % Almacenamos cada simbolo OFDM de 64 muestras en una variable
   simbF=simbolosfft(:,auxnSimb); % Símbolo en frecuencia
   % Determinamos el comportamiento del canal sobre las portadoras
   % piloto
   v1=[simbF(8) simbF(22)]; % Valores de pilotos rx
   v2=[simbF(44) simbF(58)]; % Valores de pilotos rx (2)
   % Realizamos el proceso de interpolacion lineal a los datos
   % anteriores obtenidos
   x1=[8 22]; % Puntos de la muestra
   xq1=(2:23);      % Puntos a estimar
   x2=[44 58];% Puntos de la muestra
   xq2=(43:64);     % Puntos a estimar
   val_interp1 = (interp1(x1,v1,xq1,'linear','extrap')).';
   val_interp2 = (interp1(x2,v2,xq2,'linear','extrap')).';
   % Se aplica la interpolación a los datos
   datEc1=simbF(2:23)./val_interp1;
   datEc2=simbF(43:64)./val_interp2;
   % Almacenamos los datos ecualizados en una nueva variable
   DatosEc=zeros(64,1);
   DatosEc(2:23)=datEc1;
   DatosEc(43:64)=datEc2;
   simbolosEcualizados(:,auxnSimb)=DatosEc;
end
nSimOFDM=auxnSimb; % número de símbolos OFDM
disp('Ecualizacion terminada');
%% %%%%%%%%%%%%% RECUPERAR DATOS DEL SÍMBOLO OFDM    %%%%%%%%%%%%%%
nPdatos=40; % Número de portadoras de datos
datosMapeo=zeros(nPdatos*nSimOFDM,1); % Variable que contiene los
% datos serializados
for aux=1:nSimOFDM
   % obtener los datos OFDM
   simAux=simbolosEcualizados(:,aux); % copiar en una var. auxiliar
   % Desamblaje del símbolo OFDM:
   datosAux=zeros(40,1); % Inicializar en 0 un vector auxiliar
   %             simAux(1) es Portadora DC
   datosAux(21:26)=simAux(2:7);
   %             simAux(8) es la primera piloto
   datosAux(27:39)=simAux(9:21);
   %             simAux(22) es la segunda piloto
   datosAux(40)=simAux(23);
   %           simAux(24:42) son las portadoras de guarda
   datosAux(1)=simAux(43);
   %          simAux(44) es la tercera piloto
   datosAux(2:14)=simAux(45:57);
   %            simAux(58) es la cuarta piloto
   datosAux(15:20)=simAux(59:64);
   % Serializar los datos:
   datosMapeo((aux-1)*nPdatos+1:aux*nPdatos)=datosAux;
end
disp('Proceso desamblaje OFDM terminado')
```

```
    %% %%%%%%%%%%%%%%%%%%%%%%%%%    DEMAPEO
%%%%%%%%%%%%%%%%%%%%%%%%%%
    if(seleccionMapeo==1)
    % Demapeo BPSK
    M=2; % Orden de modulación
    Nb=log2(M); % Bits por símbolo
    % Los datos filtrados no se normalizan
    longitud=length(datosMapeo); % Longitud datos de entrada
    datosDemod=zeros(longitud,1); % Inicializar vector
    % Implementar el Demapeo BPSK con el lazo for
    for i=1:longitud
        if(real(datosMapeo(i))>=0 && imag(datosMapeo(i))>=0)
            datosDemod(i)=1;
        elseif(real(datosMapeo(i))<0 && imag(datosMapeo(i))<0)
            datosDemod(i)=0;
        end
    end
elseif(seleccionMapeo==2)
    % Demapeo QPSK
    M = 4; % Orden de modulación
    Nb=log2(M); % Bits por símbolo
    long_pM=ceil(length(datosMapeo)/Nb)*Nb-length(datosMapeo);
    %ceil=redondea la longitud al entero inmediato superior
    paddM=zeros(long_pM,1); % relleno con ceros
    % Los datos filtrados solo se multiplican por sqrt(2)
    ModDat=(sqrt(2))*[datosMapeo;paddM]; %Datos con relleno
    % Demodulación QPSK:
    datosDemodS=pskdemod(ModDat,M,pi/4,'gray');
    %Demodula ModDat con una orden de modulación M y fase inicial de pi/4
    %Gray significa el orden de símbolos, que esta codificada en orden gray
    % Pasar de símbolos a bits:
    demodMatriz=de2bi(datosDemodS,Nb,'left-msb');
    %de2bi convertir de decimal a binario
    %left-msb Indica la columna izquierda (o primera) de la salida binaria,
    %como el bit más significativo(o dígito de mayor orden).
    datosDemod=reshape(demodMatriz.',1,[]).'; % Datos
    % recuperados (bits Serializados)
elseif(seleccionMapeo==3)
    % Demapeo 16-QAM
    M = 16; % orden de modulación
    Nb = log2(M); % Bits por símbolo
    long_pM=ceil(length(datosMapeo)/Nb)*Nb-length(datosMapeo);
    %ceil=redondea la longitud al entero inmediato superior
    paddM=zeros(long_pM,1); % relleno con ceros
    % Los datos filtrados solo se multiplican por sqrt(10)
    ModDat=(sqrt(10))*[datosMapeo;paddM]; % Datos con relleno
    % Demodulación 16-QAM:
    datosDemodS=qamdemod(ModDat,M);
    %qamdemod=demodula la señal ModDat con QAM con el orden de modulación M
    % Pasar de símbolos a bits:
    demodMatriz=de2bi(datosDemodS,Nb,'left-msb');
    %de2bi convertir de decimal a binario
    %left-msb Indica la columna izquierda (o primera) de la salida binaria,
    %como el bit más significativo(o dígito de mayor orden).
    datosDemod=reshape(demodMatriz.',1,[]).'; %Datos
    % recuperados (bits Serializados)
elseif(seleccionMapeo==4)
    % Demapeo 64-QAM
    M = 64; % Orden de modulación
```

```
    Nb = log2(M); % Bits por símbolo
    long_pM=ceil(length(datosMapeo)/Nb)*Nb-length(datosMapeo);
    %ceil=redondea la longitud al entero inmediato superior
    paddM=zeros(long_pM,1); % relleno con ceros
    % Los datos filtrados solo se multiplican por sqrt(42)
    ModDat=(sqrt(42))*[datosMapeo;paddM]; % Datos con relleno
    % Demodulación 64-QAM:
    datosDemodS=qamdemod(ModDat,M);
    %qamdemod=demodula la señal ModDat con QAM con el orden de modulación M
    % Pasar de símbolos a bits:
    demodMatriz=de2bi(datosDemodS,Nb,'left-msb');
    %de2bi convertir de decimal a binario
    %left-msb Indica la columna izquierda (o primera) de la salida binaria,
    %como el bit más significativo(o dígito de mayor orden).
    datosDemod=reshape(demodMatriz.',1,[]).'; %Datos
    % recuperados (bits Serializados)
    else
        % La variable selección mapeo tiene otro valor por
        % lo que se regresa al inicio del lazo while
        disp('Mapeo no realizado')
        continue
    end
    % La variable datosDemod es el vector de salida
    disp('Datos Demapeados')
    %% %%%%%%%%%%%%%%%%%% DECODIFICADOR CONVOLUCIONAL
%%%%%%%%%%%%%%%%%%%%
    if correccionErrores==20
        % Calcular el número de bits codificados por símbolo:
        NPdatos=40; % Portadoras de datos por símbolo OFDM
        % Nb es en número de bits por símbolo mapeado (definido
        % en la etapa demapeo)
        nBc=Nb*NPdatos; % Número de bits codificados por simb OFDM
        % Relleno de bits para que el vector de entrada sea multiplo de nBc:
        nGrupos=ceil(length(datosDemod)/nBc);
        %ceil=redondea la longitud al entero inmediato superior
        longiPadd=nBc*nGrupos-length(datosDemod); % Longitud de relleno
        v_relleno=zeros(longiPadd,1); % relleno de ceros
        datosDemodRell=[datosDemod;v_relleno]; % Datos serializados con
        % relleno de ceros
        % Parámetros del codificador convolucional: (2, 1, 3)
        nc=2; % n es la longitud por cada ingreso de k bits
        kc=1; % k es los bits de entrada
        mc=3; % m es el tamaño de registro de la memoria
        g1=5; % Polinomio generador x2+1=5(oct)
        g2=7; % Polinomio generador x2+x+1=7(oct)
        Rcc=kc/nc; % tasa del codificador convolucional
        P_Trellis = poly2trellis(mc, [g1 g2]); %Estructura de Trellis
        tb=nBc*Rcc; % La profundidad de rastreo debe ser igual al número de
        % bits que se tuvo a la entrada del codificador
        datosDecodConv=[]; % Inicilizar vector que almacena los datos
        % decodificados
        % Con un for se recorre los grupos de bits a ser decodificados
        for i=1:nGrupos
            auxDecod = datosDemodRell((nBc*(i-1)+1):nBc*i,1); % Se toma el
            % grupo correspondiente
            DecodVit= vitdec(auxDecod,P_Trellis,tb,'trunc','hard'); % Se
            % decodificación binaria convolucional de auxDecod con la estructura de
            %Trellis y profundidad de rastreo tb
            %trunc significa el modo de operación
```

```
            %hard significa el tipo de decodificación
            datosDecodConv=[datosDecodConv;DecodVit]; % Almacenar los bits
            % decodificados
        end
            % El vector de salida es datosDecodConv
            disp('Corrección de errores implementado')
        else
        datosDecodConv=datosDemod; % Copiar los datos
    end
    %% %%%%%%%%%%%%%%%%%%%%%%%     MEDIR BER
%%%%%%%%%%%%%%%%%%%%%%%%%
    if(actBER==20)
        % Medición del BER activado
        LongDatDecod=length(datosDecodConv); % longitud del archivo
        if (longDatosBER<=LongDatDecod)
            % Longitud correcta y se puede recortar los datos y medir el
            % BER
            datosSerializadosRx=datosDecodConv(1:longDatosBER);
            bitsErrado=0;
            % la variable longDatosBER es la longitud del archivo y se
            % define en el botón Abrir Archivo
            for aux=1:longDatosBER
                if(datosSerializadosRx(aux)~=datosMedirBER(aux))
                    bitsErrado=bitsErrado+1;
                end
            end
            disp('Medición del BER:');
            % Imprimir la cantidad de bits errados sobre la longitud total:
            v_BER=bitsErrado/longDatosBER; % Valor de BER
            fprintf('Bits errados/Bits totales = %g/%g \n',...
            bitsErrado,longDatosBER);
            fprintf('BER = %g \n',v_BER);

        else
            disp('Archivo Recuperado, BER no medido')
            disp('Longitud del archivo recuperdado menor al original');
        end
    else
        % Copiar los datos de entrada en el vector de salida
        datosSerializadosRx=datosDecodConv;
    end
    % El vector de salida es datosSerializadosRx
    %% %%%%%%%%%%%%%%%%%%%%%%%   RECUPERAR TEXTO
%%%%%%%%%%%%%%%%%%%%%%%
    % Relleno de ceros:
    bPc=8; % Bits por cada caracter
    numC=ceil(length(datosSerializadosRx)/8); %Número caracteres
    LongRellnumC=bPc*numC-length(datosSerializadosRx); % Longitud
    % de relleno para hacer múltiplo de 8
    rellenoC=zeros(LongRellnumC,1); % Vector de relleno ceros
    datosSerializados=[datosSerializadosRx ; rellenoC]; %Rellenar
    % en caso de ser necesario
    % Pasar de los bits serializados a una matriz, en donde las
    % filas contiene a los caracteres en binario:
    datosMtz=reshape(datosSerializados,[8 numC]).'; % numC filas
    % x 8 columnas
    % Pasar de binario a decimal, los datos de cada fila:
    datosTexto=[]; % Inicializar vector
    for i=1:numC
```

```
        % Se pasa cada fila a decimal y se almacena en datosTexto
        datosTexto(1,i)=double(bi2de(datosMtz(i,:),'left-msb'));
    end
    %Texto Original
    Texto_Original=transpose(x)
    % Pasar de decimal a caracteres:
    disp('Texto recuperado: ');
    Texto_Recuperado=char(datosTexto) % Texto recuperado
    disp('Proceso recuperacion de texto terminado');
    toc % Fin de medir el tiempo de procesamiento
    pause(intTimeR) % Pause para esperar un momento antes de volver a
    % recibir
end
```

ANEXO VII

transmisor_ReedSolomon.m

```
%% TRANSMISOR REED - SOLOMON
%% Parámetros para seleccionar algoritmos y controlar la transmisión
seleccionMapeo=4; % Variable para seleccionar el tipo de mapeo a utilizar,
% BPSK (1), QPSK (2), 16-QAM(3), 64-QAM(4)
correccionErrores=20; % Variable para activar (20) o desactivar (10) la
% corrección de errores
intTime=3; %Intervalo de tiempo en segundos, para esperar entre
% transmisiones
txRealTime=20; % Se setea la variable para ingresar al lazo while de
% transmisión repetitiva cada intTime segundos
%% Seleccionar archivo de texto y procesarlo para transmitir
nombreArchivo=uigetfile('*.txt;','Seleccionar archivo a transmitir');
% La variable nombreArchivo toma el valor de 0 cuando no se ha
% seleccionado ningun archivo y toma el nombre del archivo cuando
% si se ha seleccionado un archivo
if nombreArchivo==0
    % No se ha seleccionado ningun archivo
    disp('Archivo no seleccionado'); % Se utiliza para informar
else
    %% %%%%%%%%%%%%%%%%%%% ABRIR ARCHIVO DE TEXTO
%%%%%%%%%%%%%%%%%%%%
    % Se ha seleccionado un archivo y se procede a abrir el mismo
    fID = fopen(nombreArchivo); % Abrir el archivo
    datosAbiertos=fread(fID,'*char'); % Leer los caracteres
    fclose(fID); % Cerrar el archivo
    %% %%%%%%%%%%%%%%%%%%%% SERIALIZAR DATOS
%%%%%%%%%%%%%%%%%%%%%%%%
    clearvars textoBitsSerial datosSerializados
    textoBinario = dec2bin(datosAbiertos,8); % representacion de cada
    % caracter con 8 bits
    % Para serializar los datos se utiliza dos lazos for(). El primer
    % lazo for() recorre cada fila, length(textoBinario) representa el
    % número de filas. El segundo lazo for() en cada fila recorre todos
    % los elementos (8 columnas). Y con una variable auxiliar (aux) se
    % almacena los datos en la posición que dicte la variable aux.
    aux=1; % Inicializar variable auxiliar
    for i = 1:length(textoBinario)
        for j=1:8
            textoBitsSerial(aux)=textoBinario(i,j); % tipo char
            aux=aux+1; % Sumar 1 en la variable auxiliar
        end
```

```
    end
    % Para convertir los datos del tipo char al tipo str se utiliza un
    % lazo for para recorrer todos los elementos del vector serializado
    for aux=1:length(textoBitsSerial)
        % Se utiliza una variable auxiliar para almacenar los datos
        datosSerializados(aux)=str2double(textoBitsSerial(aux));
    end
    disp('Datos Serializados') % Mostrar un msj en el Command Window
    %% Codificador Reed - Solomon
    datosBina=datosSerializados; % Copiar los datos serializados en una
    % nueva variable
    % Se utiliza un lazo condicional if() para determinar si se utiliza la
    % correción de errores
    if correccionErrores==20
        % Se ha seleccionado la corrección de errores. (Reed Solomon)
        % Se define los valores a considerar de las etapas posteriores:
        % Esquema RS (5,3,1) (n,k,t)
        NPdatos=40; % número de portadora de datos por símbolo OFDM
NRS=5;      %Longitud del codificador
KRS=3;      %Símbolos originales
tasaRS=KRS/NRS;
Nbits=tasaRS*NPdatos; % bits de datos de entrada por cada símbolo OFDM
% Rellenar los datos:
m=8; % tamaño de símbolo (bits)
nS=ceil(length(msg)/Nbits); % número de símbolos (filas)
longitudPadd=Nbits*nS-length(msg); % Longitud de relleno
vector_padd=zeros(1,longitudPadd); % relleno de ceros
auxDatos=[msg vector_padd]; % Datos serializados con padd de ceros
% El vector datosDecimal puede tomarse directo de los símbolos decimales
% del archivo de texto
MatrizBin=reshape(auxDatos,m,[]).'; % Matriz de m columnas y cada fila
%corresponde a un símbolo
datosDecimal=bi2de(MatrizBin,'left-msb').'; % matriz con representación decimal
%bi2de convertir de binario a decimal
%left-msb Indica la columna izquierda (o primera) de la salida binaria,
%como el bit más significativo(o dígito de mayor orden)
datosBinario=[]; %Inicializo vector que almacena los datos Binarios
LsimbolosDec=length(datosDecimal); % longitud del vector de datos decimal
%Se recorre en multiplos de KRS símbolos:
for i=1:KRS:LsimbolosDec
    auxDecimal=datosDecimal(i:i+KRS-1); % Tomar de KSR en KRS símbolos decim
    datosGalois=gf(auxDecimal,m); % Campo de Galois de la matriz auxDecimal
    %con el tamaño de símbolo para relleno de datos, m tiene que ser entero
    %de 1 a 16
    %tic
    CodRS = rsenc(datosGalois,NRS,KRS); % codificador Reed-Solomon
    %Codifica la matriz datosGalois con la longitud del codificador  y los
    %símbolos originales y los guarda en CodRS
    %toc
    CodRSd=double(CodRS.x); % Extraer los datos decimales de CodRS, estos
    % son los símbolos decimales de info mas los de paridad
    %simbolos a bits
    bitRS=de2bi(CodRSd.',m,'left-msb');
    %de2bi convertir de decimal a binario
    %left-msb Indica la columna izquierda (o primera) de la salida binaria,
    %como el bit más significativo(o dígito de mayor orden)
    bitRSs=reshape(bitRS.',1,[]); % vector con bits serializados
    %Reshape cambia el tamaño y la forma de bitRS
    datosBinario=[datosBinario bitRSs]; % Vector con bits
```

```
    % serializados acumulados
end
        disp('Codificación Reed Solomon Agregada') % Mensaje informativo
    else
        % No se ha seleccionado la corrección de errores
        datosBinario=datosBina; % Se copia los datos serializados en la
        % variable de salida de esta etapa y por ende la entrada de la
        % siguiente etapa
    end

    %% %%%%%%%%%%%%%%%%%%%%%%%%%%%% MAPEO
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % El vector de entrada de esta etapa es datosBinario y el de salida es
    % datosMapeados
    % El valor en la variable global seleccionMapeo define el tipo de
    % mapeo a utilizar BPSK (1), QPSK (2), 16-QAM(3) o 64-QAM(4)
   if(seleccionMapeo==1)
    % Mapeo BPSK
    datosColumna=datosBinario.'; % Se transpone la matriz que contiene
    % los datos de entrada para obtener un vector columna
    % Se recorre los cada bit de entrada y se asigna un símbolo
    for i=1:length(datosColumna)
        % Con if se discrimina si es igual a 1 y se asigna el
        % valor 0.707+0.707i, caso contrario es 0 y se asigna el
        % valor de -0.707-0.707i
        if(datosColumna(i)==1)
            datosMapeados(i,1)=complex(0.707,0.707);
        else
            datosMapeados(i,1)=complex(-0.707,-0.707);
        end
    end
    % Factor de normalización = 1
    disp('Técnica de corrección de errores: Reed Solomon')
    disp('Tipo de Mapeo: BPSK')
elseif(seleccionMapeo==2)
    % Mapeo QPSK
    M = 4; % Orden de modulación
    Nb=log2(M); % Bits por símbolo, se utiliza 2 bits por símbolo
    % Hay que asegurarse que la longitud de los datos de entrada sea
    % múltiplo de Nb, por lo que se rellena de ceros cuando no es
    % múltiplo de Nb.
    % Se calcula la longitud de relleno y se almacena en long_padd0
    long_padd0=ceil(length(datosBinario)/Nb)*Nb-length(datosBinario);
    %ceil=redondea la longitud al entero inmediato superior
    paddQ=zeros(1,long_padd0); % Vector de relleno con ceros
    datosbin=[datosBinario paddQ]; % Vector con relleno
    %Pasar de binario a símbolos decimales:
    dataMatriz=reshape(datosbin,Nb,[]).'; % Matriz de Nb columnas y
    % cada fila corresponde a un símbolo
    dataSimb=bi2de(dataMatriz,'left-msb'); % Símbolos a modular
    %bi2de cambio de binario a decimal dataMatriz
    %left-msb Indica la columna izquierda (o primera) de la salida binaria,
    %como el bit más significativo(o dígito de mayor orden)
    % Factor de normalización = 1/sqrt(2)
    datosMapeados = (1/sqrt(2))*pskmod(dataSimb,M,pi/4,'gray'); % Datos
    % mapeados con QPSK
    %pskmod modula por desplazamiento de fase dataSimb con un orden de
    %modulación M y una fase inicial igual a pi/4
    %Gray significa el orden de símbolos, que esta codificada en orden gray
```

```
    disp('Técnica de corrección de errores: Reed Solomon')
    disp('Tipo de Mapeo: QPSK')
elseif (seleccionMapeo==3)
    % Mapeo 16-QAM
    M = 16; % orden de modulación
    Nb=log2(M); % Bits por símbolo, se usan 4 bits por símbolo
    % Hay que asegurarse que la longitud de los datos de entrada sea
    % múltiplo de Nb, por lo que se rellena de ceros cuando no es
    % múltiplo de Nb.
    % Se calcula la longitud de relleno y se almacena en long_padd0
    long_padd0=ceil(length(datosBinario)/Nb)*Nb-length(datosBinario);
    %ceil=redondea la longitud al entero inmediato superior
    paddQ=zeros(1,long_padd0); % Vector de relleno con ceros
    datosbin=[datosBinario paddQ]; % Vector con relleno
    %Pasar de binario a símbolos decimales:
    dataMatriz=reshape(datosbin,Nb,[]).'; % Matriz de Nb columnas y
    % cada fila corresponde a un símbolo
    dataSimb=bi2de(dataMatriz,'left-msb');
    %bi2de cambio de binario a decimal
    %left-msb Indica la columna izquierda (o primera) de la salida binaria,
    %como el bit más significativo(o dígito de mayor orden)
    % Factor de normalización = 1/sqrt(10)
    datosMapeados = (1/sqrt(10))*qammod(dataSimb,M); % Modulación 16-QAM
    %qammod Modula la señal dataSimb en amplitud de cuadraturacon la
    %orden de modulación M, la salida es la señal modulada
    disp('Técnica de corrección de errores: Reed Solomon')
    disp('Tipo de Mapeo: 16-QAM')
elseif (seleccionMapeo==4)
    % Mapeo 64-QAM
    M = 64; % Orden de modulación
    Nb=log2(M); % Bits por símbolo, se usan 6 bits por símbolo
    % Hay que asegurarse que la longitud de los datos de entrada sea
    % múltiplo de Nb, por lo que se rellena de ceros cuando no es
    % múltiplo de Nb.
    % Se calcula la longitud de relleno y se almacena en long_padd0
    long_padd0=ceil(length(datosBinario)/Nb)*Nb-length(datosBinario);
    %ceil=redondea la longitud al entero inmediato superior
    paddQ=zeros(1,long_padd0); % Vector de relleno con ceros
    datosbin=[datosBinario paddQ]; % Vector con relleno
    %Pasar de binario a símbolos decimales:
    dataMatriz=reshape(datosbin,Nb,[]).'; % Matriz de Nb columnas y
    % cada fila corresponde a un símbolo
    dataSimb=bi2de(dataMatriz,'left-msb'); % Símbolos a modular
    % bi2de, cambio de binario a decimal
    %left-msb Indica la columna izquierda (o primera) de la salida binaria,
    %como el bit más significativo(o dígito de mayor orden)
    % Factor de normalización = 1/sqrt(42)
    datosMapeados = (1/sqrt(42))*(qammod(dataSimb,M)); % Datos
    % mapeados con 64-QAM
    %qammod Modula la señal dataSimb en amplitud de cuadraturacon la
    %orden de modulación M, la salida es la señal modulada
    disp('Técnica de corrección de errores: Reed Solomon')
    disp('Tipo de Mapeo: 64-QAM')
end
    %% %%%%%%%%%%%%%%%%%%%%%% CREAR SIMBOLO OFDM
%%%%%%%%%%%%%%%%%%%%%%%%%%
    % El vector de entrada de esta etapa es datosMapeados, el cual es un
    % vector tipo columna.
    % El vector de salida es simbolosOFDMCreados
```

```
% La estructura del símbolo OFDM consta de una subportadora DC,
% 4 pilotos, 19 de guarda y 40 de datos.
nPdatos=40; % Número de portadoras de datos
% El vector de entrada debe ser múltiplo de nPdatos, cuando no se
% se cumple dicha condición se realiza un relleno con los datos del
% inicio al final de vector. Esto con el objetivo de hacer que el
% vector sea múltiplo de nPdatos:
longDatos=length(datosMapeados); % Longitud del vector de entrada
nSimbOFDM=ceil(longDatos/nPdatos); % Calcular el número de símbolos
longPadd=nSimbOFDM*nPdatos-longDatos; % Calcular la longitud del
% relleno
datosEntradaPadd=datosMapeados; % Copio los datos de entrada en una
% nueva variable auxiliar
% Si la longitud de relleno es cero no se realiza ninguna operación
% adicional, si es mayor que cero se realiza el relleno
if(longPadd>0)
   % Se hace el relleno con los datos del inicio del vector de entrada
   datosEntradaPadd(end+1:end+longPadd)=datosMapeados(1:longPadd);
end
% Se procede a crear los simbolos OFDM recorriendo en múltiplos de
% nPdatos, mediante un lazo for:
numeroSimbolos=1; % Inicializo la variable que sirve para almacenar los
% símbolos OFDM creados, en una matriz.
longDatosPadd=length(datosEntradaPadd); % Longitud de los datos de
% entrada con relleno
for i=1:nPdatos:longDatosPadd
   datosAux=datosEntradaPadd(i:i+nPdatos-1); %Copio los datos nPdatos,
   % es decir los 40 datos a introducirse dentro del símbolo OFDM
   DC=0;% Valor para la subportadora DC
   % Las variables p1, p2, p3, p4 son los valores que toman las
   % subportadoras piloto, usadas para poder hacer una estimación del
   % canal en recepción
   p1=1;
   p2=1;
   p3=1;
   p4=1;
   simbOFDM=zeros(64,1); % Inicializo símbolo OFDM, con todos los
   % valores en cero
   simbOFDM(1)=DC; % Portadora DC
   % Asignación del valor a cada subportadora de datos, y a las
   % subportadoras piloto:
   simbOFDM(2:7)=datosAux(21:26);
   simbOFDM(8)=p1; % Piloto
   simbOFDM(9:21)=datosAux(27:39);
   simbOFDM(22)=p2; % Piloto
   simbOFDM(23)=datosAux(40);
   % simbOFDM de 24 a 42 son de guarda
   simbOFDM(43)=datosAux(1);
   simbOFDM(44)=p3;% Piloto
   simbOFDM(45:57)=datosAux(2:14);
   simbOFDM(58)=p4; % Piloto
   simbOFDM(59:64)=datosAux(15:20);
   simbolosOFDMCreados(1:64,numeroSimbolos)=simbOFDM; % Matriz de
   % salida que almacena el símbolo OFDM completo, según la columna
   % que dicte la variable numeroSimbolo
   numeroSimbolos=numeroSimbolos+1; % Aumento en uno la variable
end
numeroSimbolos=numeroSimbolos-1; % Disminuyo en uno el valor de la
% variable para que concuerde con la cantidad total de símbolos OFDM
```

```
    disp('Símbolos OFDM creados');
    %% %%%%%%%%%%%%%%%%%%%%%%%%%%%%% IFFT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % En esta sección se realiza la modulación de los símbolos OFDM
    % mediante la ifft. La matriz de entrada es simbolosOFDMCreados y la de
    % salida es simbolosOFDM.
    % La función ifft admite como entrada un vector columna, pero tambien
    % una matriz. En la matriz aplica la ifft a cada columna. Como la
    % matriz de entrada contiene un spimbolo en cada columna se puede
    % aplicar directamente la función ifft
    simbolosOFDM=ifft(simbolosOFDMCreados); % matriz de salida
    disp('IFFT implementada');
    %% %%%%%%%%%%%%%%%%%%%%%%%%%%%%% AÑADIR PC
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % El prefijo cíclico (PC) se añade copiando la parte final de cada
    % símbolo, al inicio del mismo.
    % La matriz de entrada es simbolosOFDM y a la salida se tiene una
    % matriz llamada simbolosOFDMpc con los símbolos en cada columna y
    % un vector contodos los símbolos serializados llamado datosEnviar
    % Con un lazo for se recorre cada símbolo modulado, se añade el PC
    % y se serializa los datos
    datosEnviar=[]; % Variable que almacenara los datos serializados
    for auxPC=1:numeroSimbolos
        % Añadir el prefijo cíclico, de tamaño igual a la cuarta parte del
        % símbolo:
        simbolosOFDMpc(1:80,auxPC)=[simbolosOFDM(49:64,auxPC);simbolosOFDM(:,auxPC)];
        datosEnviar=[datosEnviar;simbolosOFDMpc(:,auxPC)]; % Datos
        % serializados
    end
    disp('Prefijo cíclico (PC) añadido');
    %% %%%%%%%%%%%%%%%%%%%%%%%%%% AÑADIR PREAMBULO
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Añadimos el preambulo dado por el estandar 802.11a antes de los
    % símbolos OFDM serializados que contiene datosEnviar
    % Se inicia por definir parte de la sección corta y larga en el dominio
    % de la frecuencia para luego obtener el preambulo en el dominio del
    % tiempo:
    % 4 secuencias cortas (parte de LSTF) en el dominio de la frecuencia
    parteCorta64=[0;0;0;0;-1.4720-1.4720i;0;0;0
        -1.4720-1.4720i;0;0;0;1.4720+1.4720i;0;0;0
        1.4720+1.4720i;0;0;0;1.4720+1.4720i;0;0;0
        1.4720+1.4720i;0;0;0;0;0;0;0;0;0;0;0;0;0;0
        1.4720+1.4720i;0;0;0;-1.4720-1.4720i;0;0;0
        1.4720+1.4720i;0;0;0;-1.4720-1.4720i;0;0;0
        -1.4720-1.4720i;0;0;0;1.4720+1.4720i;0;0;0];
    % Secuencia larga (parte de LLTF) en el dominio de la frecuencia
    parteLarga=[...
        0;1;-1;-1;1;1;-1;1;-1;1;-1;-1;-1;-1;1
        1;-1;-1;1;-1;1;-1;1;1;1;1;0;0;0;0;0
        0;0;0;0;0;0;1;1;-1;-1;1;1;-1;1;-1;1
        1;1;1;1;1;-1;-1;1;1;-1;1;-1;-1;1;1;1;1];
    % Conversión del dominio de la frecuencia al tiempo, y creación de la
    % sección corta completa (LSTF):
    parteCorta64Time=ifft(parteCorta64); % pasar al dominio del tiempo
    parteCorta80Time=parteCorta64Time(49:64);% copiar las ultimas
    % 16 muestras al inicio de la variable
    parteCorta80Time(17:80)=parteCorta64Time; % 5 secuencias cortas
    preambulo=parteCorta80Time; % copiar las 5 secuencias cortas
    preambulo(81:160)=parteCorta80Time; % añadir las otras 5 secuencias
```

```
    % cortas para obtener la sección corta (LSTF)
    % Conversión del dominio de la frecuencia al tiempo, y adición de PC
    % de la sección larga:
    parteLargaTime=ifft(parteLarga); % conversión al dominio del tiempo la
    % secuencia larga
    preambulo(161:192)=parteLargaTime(33:64); % PC en el preámbulo
    preambulo(193:256)=parteLargaTime; % primera secuencia larga
    preambulo(257:320)=parteLargaTime; % segunda secuecia larga y se obtiene
    % el preámbulo completo.
    % Unir el preambulo a los símbolos OFDM serializados:
    datosConPre=[preambulo;datosEnviar]; % Preambulo con símbolos OFDM
    % creados previamente
    disp('Preámbulo 802.11a añadido');
    % Inicializar con ceros una variable auxiliar, que tenga el tamaño del
    % bloque a enviar:
    auxDat=zeros(tamanioBloque,1); % inicializar variable
    [longDat,~]=size(datosConPre); % longitud de los datos a enviar
    auxDat(1:longDat,1)=datosConPre; % copiar el preambulo con los símbolos
    % OFDM y los demás datos del bloque se mantienen en cero
    datosEnviarBloque=auxDat;% Datos de salida a enviar
    disp('Datos Procesados')
    %% %%%%%%%%%%%%%%%% NORMALIZAR LOS DATOS A TRANSMITIR %%%%%%%%%%%%%%%%
    % Normalizar los datos a transmitir:
    % Se evalua la parte real y la parte imaginaria de los vectores, para
    % encontrar el módulo máximo de la parte real e imaginaria.
    v_maxR=max(abs(real(datosEnviarBloque))); % Valor máximo, parte real
    v_maxI=max(abs(imag(datosEnviarBloque))); % Valor máximo, parte imag
    % Con un condicional se evalúa si la parte imaginaria o la parte real
    % tiene el valor máximo para normalizar los datos
    if(v_maxR<v_maxI)
        valorMax=v_maxI; % La parte imaginaria tiene el máximo valor abs
    else
        valorMax=v_maxR; % La parte real tiene el máximo valor absoluto
    end
    % Normalizo y almaceno los datos a transmitir en una variable auxiliar
    aux_TxNorm=datosEnviarBloque/valorMax;
    % Convertimos los datos a un tipo single que emplea 32 bits, mientras
    % que double emplea 64 bits para la representacion de cada valor:
    auxTx=single(aux_TxNorm);
    %% %%%%%%%%%%%%%%%%%%%%%% TRANSMITIR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Se ingresa al lazo para transmitir cada intervalo intTime:
    while (txRealTime==20)
        % se procede a transmitir el bloque de datos OFDM utilizando el
        % objeto de transmisión txAdalm()
        txAdalm(auxTx(:,1));
        % mostramos un mensaje de transmision terminada
        disp('Datos transmitidos...');
        pause(intTime)
    end
end
```

ANEXO VIII

receptor_ReedSolomon.m

```
%% RECEPTOR REED - SOLOMON
%% Parámetros para seleccionar algoritmos y controlar la recepción
seleccionMapeo=4; % Variable para seleccionar el tipo de mapeo a utilizar,
```

```
% BPSK (1), QPSK (2), 16-QAM(3), 64-QAM(4)
correccionErrores=20; % Variable para activar (20) o desactivar (10) la
% corrección de errores
umbralRx=0.0040; % Umbral para la recepción, tiene que ser mayor a 0.0026
intTimeR=10; %Intervalo de tiempo en segundos, para esperar cuando se haya
% recuperado el texto
rxRealT=20; % Se setea la variable para ingresar al lazo while de recepción
% en tiempo real
%% %%%%%%%%%%%%% SELECCIÓN DE ARCHIVO PARA MEDIR BER
%%%%%%%%%%%%%%%%%%
nombreArchivo=uigetfile('*.txt;','Seleccionar archivo a transmitir');
% La variable nombreArchivo toma el valor de 0 cuando no se ha
% seleccionado ningun archivo y toma el nombre del archivo cuando
% si se ha seleccionado un archivo
if nombreArchivo==0
   % No se ha seleccionado ningun archivo
   disp('Archivo no seleccionado'); % Se utiliza para informar
   actBER=10; % Variable para activar o desactivar(10) la medición del BER
else
   % Se ha seleccionado un archivo y se procede a abrir el mismo
   fid =fopen(nombreArchivo);
   x=fread(fid,'*char'); % Datos Originales
   fclose(fid);
   textoBinario = dec2bin(x,8);% representar cada caracter con 8 bits
   % Serializar:
   tamanioTexto=length(textoBinario);
   aux=1;
   for i = 1:tamanioTexto
      for j=1:8
         textoBitsSerializado(aux)=textoBinario(i,j);
         aux=aux+1;
      end
   end
   datosSerializadosTxOrig=textoBitsSerializado;
   % De char a double:
   for aux=1:length(datosSerializadosTxOrig)
      % Datos originales en bits:
      datosMedirBER(aux)=str2double(datosSerializadosTxOrig(aux));
   end
   longDatosBER=length(datosMedirBER); % longitud de los datos abiertos
   % para obtener el BER
   actBER=20; % Variable para activar(20) o desactivar la medición del BER
end
%% Recepción en tiempo real:
% En el lazo while se implementa todos los procesos de recepción en tiempo
% real
while (rxRealT==20)
   clearvars datosRecibidos; % Limpiar la variable
   %% %%%%%%%%%%%%%%%%%%%%%%%%    RECIBIR DATOS
%%%%%%%%%%%%%%%%%%%%%%%%%%
   for i=0:6
      vaciar=rxAdalm(); % Este vector no se utiliza, solo muestrea
      % el canal varias veces para que en caso de existir algún
      %buffer este se elimine
   end
   condAux=0; % Condición auxiliar, para estar dentro del while
   %Recepcion de los datos:
   while(condAux==0)
      datosRxComplejos= rxAdalm(); % Guarda la señal recuperada del
```

```
        % canal inalámbrico
        datosRecibidos=double(datosRxComplejos); % de single a double
        v_max=max(abs(datosRecibidos)); % Máximo valor de la magnitud
        % de cada dato recibido
        % Imprimir la magnitu máxima:
        fprintf('Valor máximo de las muestras: %8.5f\n',v_max)
        % Verificar si se supera el umbral establecido:
        if(v_max>umbralRx)
            datosRxAux=rxAdalm(); % Guarda la señal recuperada
            datosRec2=double(datosRxAux); % De single a double
            % Verificar si se supera el umbral:
            if (max(abs(datosRec2))>umbralRx)
                % Guardar en el mismo vector
                datosRecibidos=[datosRecibidos;datosRec2];
            end
            condAux=1; % Cambiar el valor para salir del lazo while
        end
    end % Final del bucle while
    disp('Datos recibidos del canal inalámbrico')
 %%   %%%%%%%%%%%%%%%%%%%%%%%%    RECORTAR DATOS
%%%%%%%%%%%%%%%%%%%%%%%%%%
    tic % Para medir el tiempo de procesamiento
    % Se recorre el vector de entrada y  se obtiene la posición del
    % primer valor por arriba del umbral:
    pos_inic=0; % Inicializar variable
    for i=1:length(datosRecibidos)
        if(max(abs(datosRecibidos(i)))>umbralRx)
            pos_inic=i; % Primera posición arriba del umbral
            break;
        end
    end
    % Guardar desde 400 datos antes de superar el umbral, hasta el final:
    if(pos_inic>400)
        datosRec=datosRecibidos(pos_inic-400:end); % Datos recortados
    else
        % El umbral se superó antes de las primeras 400 posiciones, solo
        % se copia los datos
        datosRec=datosRecibidos;
    end
    datosInv=flip(datosRec); % Invierte el vector, tal que los datos
    % finales queden al inicio, con esto se puede repetir el proceso
    % anterior.
    % Se recorre el vector de invertido y  se obtiene la posición del
    % primer valor por arriba del umbral:
    pos_fin=0; % Inicializar variable
    for i=1:length(datosRec)
        if (max(abs(datosInv(i)))>umbralRx)
            pos_fin=i; % Primera posición arriba del umbral
            break
        end
    end
    % Guardar desde 200 datos antes de superar el umbral, hasta el final:
    if(pos_fin>200)
        datosInvRec=datosInv(pos_fin-200:end); % Recortar datos
    else
        % El umbral se superó antes de las primeras 200 posiciones, solo
        % se copia los datos
        datosInvRec=datosInv;
    end
```

```
datosRecortados=flip(datosInvRec); % Invertir nuevamente el vector
disp('Datos recortados')
%% %%%%%%%%%%%%%%%%%%%%    DETECCIÓN TRAMA OFDM     %%%%%%%%%%%%%%%%
datosRX=datosRecortados; % Copiar los datos en una nueva variable
if length(datosRX)>(SamplesPerFrame+600)
    disp('***** Muestras recibidas inválidas *****');
    continue
end
% El preambulo se encuentra al inicio de cada paquete y por tanto
% se trabaja con las primeras muestras, con esto se evita
% procesamiento innecesario
Mmues=1400; % Muestras para calcular la métrica
if(length(datosRX)>Mmues)
    r=datosRX(1:Mmues);
else
    % Si es menor a Mmues se trabaja con todas las muestras
    r=datosRX;
end
L = 48; % Múltiplo dela longitud de una secuencia corta (16*x)
inMax = length(r)-2*L+1; % k+m puede tomar valores de hasta 2L-1
% el indice maximo a evaluar es la longitud de r menos 2L.
m=L-1; % Máximo valor de m
M = zeros(Mmues,1); % Inicializar la variable para mayor velocidad
% Determinar la métrica M(k) para comprobar si es un paquete OFDM:
for k=1:inMax
    Numerador = (r(k:k+m)'*r(k+L:k+m+L)); % Sumatorio del numerador
    Denom = sum((abs(r(k+L:k+m+L))).^2); % Sumatorio del denominador
    M(k)=Numerador/Denom; % Métrica
end
M2=(abs(M)).^2; % Cuadrado de la magnitud de cada valor
umbral=0.60; % Umbral para estimar si existio un paquete OFDM, debe
% ser mayor a 0.40
tramaEncontrada=0; % 0 trama no encontrada
auxET=0; % Inicializar variable que almacena la posición en donde
% se superera el umbral de la métrica
for i=1:length(M2)
    %Se obtiene la posición del primer valor por arriba del umbral:
    if(M2(i)> umbral)
        auxET=i;
        tramaEncontrada=1; % Trama encontrada
        break;
    end
end
datosOFDM=[]; % Iniciar variable que contiene la trama OFDM
if(tramaEncontrada==1)
    % Para el caso de que se haya encontrado una paquete OFDM
    datosOFDM(:,1)=datosRX(auxET:end);
else
    % Si no existe un paquete OFDM se empieza de nuevo el bucle while
    disp('***** Trama OFDM NO encontrada *****');
    pause(0.1)
    continue
end
% datosOFDM son los datos de salida
disp('Trama OFDM detectada');
%% %%%%%%%%%%%%%%%%%%%%    CORRECCIÓN CFO
%%%%%%%%%%%%%%%%%%%%%%%%
r = datosOFDM; % Copiar los datos en r para coincidir con la ecuación
Fs=frecuenciaMuestreo; % Frecuencia de muestreo de los datos recibidos
```

```
N = 128; % máximo valor del indice k
L=16; % longitud d euna secuencia corta
m=L-1; % Máxima longitud que puede tomar m
fEst = zeros(N,1); % Iniciar vector que almacena la estimación de freq
% Lazo que recorre de k igual a 1 hasta N
for k=1:N
    P = r(k:k+m)'*r(k+L:k+m+L); % Sumarotio p(k)
    fEst(k) = (Fs/(2*pi*L))*(angle(P)); % Calcular la estimación
    % de frecuencia en la posición k
end
estFreq=-mean(fEst); % Negativo del promedio del CFO estimado
%Corrección de frecuencia con la estimación realizada:
despFreqObj = comm.PhaseFrequencyOffset('FrequencyOffset',...
    estFreq,'SampleRate',Fs);
% Aplicar la estimacion de CFO a las muestras recibidas para
% corregir el CFO:
datosCorrCFO=despFreqObj(datosOFDM); % Datos corregidos el CFO
%datosCorrCFO=datosOFDM; % Esta linea se usa para evitar corregir en
%frecuencia
disp('Correccion CFO terminada');
%% %%%%%%%%%%%%%%%%%%%% SINCRONIZACIÓN DE SÍMBOLO
%%%%%%%%%%%%%%%%%%
if length(datosCorrCFO)>5000
    datSinCFO=datosCorrCFO(1:5000); % Se toma dolos primeros 5000 datos
    % para reducir el tiempo de procesamiento
else
    datSinCFO=datosCorrCFO;
end
% Crear la parte larga del preámbulo 802.11a:
% Secuencia larga (parte de LLTF) en el dominio de la frecuencia
parteLarga=[...
    0;1;-1;-1;1;1;-1;1;-1;1;-1;1;-1;-1;-1;-1;1
    1;-1;-1;1;-1;1;-1;1;1;1;1;0;0;0;0;0
    0;0;0;0;0;0;1;1;-1;-1;1;1;-1;1;-1;1
    1;1;1;1;1;-1;-1;1;1;-1;1;-1;-1;1;1;1;1];
% Conversión del dominio de la frecuencia al tiempo, y adición de PC
% de la sección larga:
parteLargaTime=ifft(parteLarga); % conversión al dominio del tiempo
LLTF=[parteLargaTime(33:64);parteLargaTime;parteLargaTime]; % LLTF
secLLTF80=LLTF(1:80); % Primeras 80 muestras de la LLTF
% Cuadrado del valor absoluto de la orrelacíón cruzada entre
% la señal recibida y secLLTF80:
datosCorrRell=(abs(xcorr(datSinCFO,secLLTF80))).^2;
% Se corrige en tiempo los datos correlacionados, para que
% coincidan con las posiciones de los datos de entrada:
datosCorr=datosCorrRell(length(datSinCFO):end);
% Almacenar la posición de los dos picos:
[~,posPico1]=max(datosCorr); % Posición del primer pico
datosCorr(posPico1)=0; % Eliminar el primer pico para poder
% encontrar el segundo
[~,posPico2]=max(datosCorr); % Posición del segundo pico
% Se comprueba que la distancia entre los dos picos sea 64
if abs(posPico2-posPico1)==64
    % Se encuentra el inicio real del LTTF y de LSTF
    inicioLLTF=min([posPico1 posPico2]); % Se toma la posición
    % menor y esa es la ubicación del primer dato del LLTF
    inicioLSTF=inicioLLTF-160; % Determinar el inicio del preambulo
    % Comprobar que la posición inicial sea mayor a 0
    if (inicioLSTF>0)
```

```
        % Recortar los datos
        datosSincronizados=datosCorrCFO(inicioLSTF:end); % Datos
        % de salida
    else
        disp('Falla en la sincronizacion de simbolo (2)')
        continue
    end
else
    disp('Falla de sincronización de símbolo')
    continue
end
% datosSincronizados es los datos de salida
disp('Sincronización de símbolo terminado');
%% %%%%%%%%%%%%%%%%%%%%    CORRECCIÓN DE FASE
%%%%%%%%%%%%%%%%%%%%%%%%
% Pasar las dos secuencias largas del LLTF del dominio del tiempo al
% dominio de la frecuencia:
a=fft(datosSincronizados(257:320)); % segunda sección larga de LLTF
b=fft(datosSincronizados(193:256)); % Primera sección larga de LLTF
%  Valores de las secuencias largas a calcular el ángulo:
angulosAux=[a(2) a(5) a(6) a(8) a(10) a(16) a(17) a(20) ...
    a(53) a(56) a(57) a(59) a(61) a(62) a(63) a(64) ...
    b(2) b(5) b(6) b(8) b(10) b(16) b(17) b(20) ...
    b(53) b(56) b(57) b(59) b(61) b(62) b(63) b(64)];
angulos=(angle(angulosAux)*180)/pi; % Calcular el ángulo y pasar de
% radianes a grados
anguloProm=-mean(angulos); % Negativo del promedio de los ángulos
% Corregir el desfase
desFaseObj = comm.PhaseFrequencyOffset('PhaseOffset',anguloProm,...
    'SampleRate',Fs);
datFase=desFaseObj(datosSincronizados);
% Se verifica que los datos corregidos sean multiplo de 80,
% caso contrario se realiza un padding de las ultimas muestras
longDatF=length(datFase);
padd80=ceil(longDatF/80)*80-longDatF;
if(padd80>=1)
    datosCorrFase=[datFase;datFase(end-padd80+1:end)];
else
    datosCorrFase=datFase;
end
% El vector de salida es datosCorrFase
disp('Corrección de fase terminado');
%% %%%%%%%%%%%%%%%%%%%%%%%    ELIMINAR PC
%%%%%%%%%%%%%%%%%%%%%%%%
clearvars simbolosSinPC;
%Eliminacion del Prefijo ciclico
nSimOFDM=0;
tramaDatos=datosCorrFase(321:end,1); %Quitar el preambulo
for j=1:80:length(tramaDatos)
    nSimOFDM=nSimOFDM+1;
    simbolosSinPC(:,nSimOFDM)=tramaDatos(j+16:j+79);
end
disp('Proceso Eliminación PC terminado');
%% %%%%%%%%%%%%%%%%%%%%%%%%    FFT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Demodulación OFDM (FFT)
% La función fft aplica la DFT a cada columna de la matriz
% simbolosSinPC
simbolosfft=fft(simbolosSinPC); % Demodular los símbolos sin PC
```

```
disp('Proceso FFT terminado');
%% %%%%%%%%%%    ESTIMACIÓN DE CANAL Y ECUALIZADOR  %%%%%%%%%%%%%%
umbralF=2; % Definir umbral
simbolosEcualizados=[];
for auxnSimb=1:nSimOFDM
   % Almacenamos cada simbolo OFDM de 64 muestras en una variable
   simbF=simbolosfft(:,auxnSimb); % Símbolo en frecuencia
   % Determinamos el comportamiento del canal sobre las portadoras
   % piloto
   v1=[simbF(8) simbF(22)]; % Valores de pilotos rx
   v2=[simbF(44) simbF(58)]; % Valores de pilotos rx (2)
   % Realizamos el proceso de interpolacion lineal a los datos
   % anteriores obtenidos
   x1=[8 22]; % Puntos de la muestra
   xq1=(2:23);      % Puntos a estimar
   x2=[44 58];% Puntos de la muestra
   xq2=(43:64);     % Puntos a estimar
   val_interp1 = (interp1(x1,v1,xq1,'linear','extrap')).';
   val_interp2 = (interp1(x2,v2,xq2,'linear','extrap')).';
   % Se aplica la interpolación a los datos
   datEc1=simbF(2:23)./val_interp1;
   datEc2=simbF(43:64)./val_interp2;
   % Almacenamos los datos ecualizados en una nueva variable
   DatosEc=zeros(64,1);
   DatosEc(2:23)=datEc1;
   DatosEc(43:64)=datEc2;
   simbolosEcualizados(:,auxnSimb)=DatosEc;
end
nSimOFDM=auxnSimb; % número de símbolos OFDM
disp('Ecualizacion terminada');
%% %%%%%%%%%%%% RECUPERAR DATOS DEL SÍMBOLO OFDM    %%%%%%%%%%%%%%
nPdatos=40; % Número de portadoras de datos
datosMapeo=zeros(nPdatos*nSimOFDM,1); % Variable que contiene los
% datos serializados
for aux=1:nSimOFDM
   % obtener los datos OFDM
   simAux=simbolosEcualizados(:,aux); % copiar en una var. auxiliar
   % Desamblaje del símbolo OFDM:
   datosAux=zeros(40,1); % Inicializar en 0 un vector auxiliar
   %            simAux(1) es Portadora DC
   datosAux(21:26)=simAux(2:7);
   %            simAux(8) es la primera piloto
   datosAux(27:39)=simAux(9:21);
   %            simAux(22) es la segunda piloto
   datosAux(40)=simAux(23);
   %          simAux(24:42) son las portadoras de guarda
   datosAux(1)=simAux(43);
   %         simAux(44) es la tercera piloto
   datosAux(2:14)=simAux(45:57);
   %            simAux(58) es la cuarta piloto
   datosAux(15:20)=simAux(59:64);
   % Serializar los datos:
   datosMapeo((aux-1)*nPdatos+1:aux*nPdatos)=datosAux;
end
disp('Proceso desamblaje OFDM terminado')
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%    DEMAPEO
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if(seleccionMapeo==1)
% Demapeo BPSK
```

```
    % Los datos filtrados no se normalizan
    longitud=length(datosMapeo); % Longitud datos de entrada
    datosDemod=zeros(longitud,1); % Inicializar vector
    % Implementar el Demapeo BPSK con el lazo for
    for i=1:longitud
        if(real(datosMapeo(i))>=0 && imag(datosMapeo(i))>=0)
            datosDemod(i)=1;
        elseif(real(datosMapeo(i))<0 && imag(datosMapeo(i))<0)
            datosDemod(i)=0;
        end
    end
elseif(seleccionMapeo==2)
    % Demapeo QPSK
    M = 4; % orden de modulación
    Nb=log2(M); % Bits por símbolo
    % Relleno de ceros:
    long_pM=ceil(length(datosMapeo)/Nb)*Nb-length(datosMapeo);
    %ceil=redondea la longitud al entero inmediato superior
    paddM=zeros(long_pM,1); % relleno con ceros
    % Los datos filtrados solo se multiplican por sqrt(2)
    ModDat=(sqrt(2))*[datosMapeo;paddM]; %Datos con relleno
    % Demodulación QPSK:
    datosDemodS=pskdemod(ModDat,M,pi/4,'gray');
    %pskmod modula por desplazamiento de fase dataSimb con un orden de
    %modulación M y una fase inicial igual a pi/4
    %Gray significa el orden de símbolos, que esta codificada en orden gray
    demodMatriz=de2bi(datosDemodS,Nb,'left-msb');
    %de2bi pasar de decimal a binario
    %left-msb Indica la columna izquierda (o primera) de la salida binaria,
    %como el bit más significativo(o dígito de mayor orden)
    datosDemod=reshape(demodMatriz.',1,[]).'; % Datos
    % recuperados (bits Serializados)
elseif(seleccionMapeo==3)
    % Demapeo 16-QAM
    M = 16; % Orden de modulación
    Nb = log2(M); % Bits por símbolo
    % Relleno de ceros:
    long_pM=ceil(length(datosMapeo)/Nb)*Nb-length(datosMapeo);
    %ceil=redondea la longitud al entero inmediato superior
    paddM=zeros(long_pM,1); % relleno con ceros
    % Los datos filtrados solo se multiplican por sqrt(10)
    ModDat=(sqrt(10))*[datosMapeo;paddM]; % Datos con relleno
    % Demodulación 16-QAM:
    datosDemodS=qamdemod(ModDat,M);
    %qamdemod, demodulación Qam de ModDat con un orden de modulación M
    % Pasar de símbolos a bits:
    demodMatriz=de2bi(datosDemodS,Nb,'left-msb');
    %left-msb Indica la columna izquierda (o primera) de la salida binaria,
    %como el bit más significativo(o dígito de mayor orden)
    datosDemod=reshape(demodMatriz.',1,[]).'; %Datos
    % recuperados (bits Serializados)
elseif(seleccionMapeo==4)
    % Demapeo 64-QAM
    M = 64; % Puntos en la constelación (símbolos)
    Nb = log2(M); % Bits por símbolo
    % Relleno de ceros:
    long_pM=ceil(length(datosMapeo)/Nb)*Nb-length(datosMapeo);
    %ceil=redondea la longitud de(datosMapeo)/Nb)*Nb-length(datosMapeo)
    paddM=zeros(long_pM,1); % relleno con ceros
```

```
    % Los datos filtrados solo se multiplican por sqrt(42)
    ModDat=(sqrt(42))*[datosMapeo;paddM]; % Datos con relleno
    % Demodulación 64-QAM:
    datosDemodS=qamdemod(ModDat,M);
    %qamdemod, demodulación Qam de ModDat con un orden de modulación M
    % Pasar de símbolos a bits:
    demodMatriz=de2bi(datosDemodS,Nb,'left-msb');
    %de2bi, pasar de decimal a binario
    %left-msb Indica la columna izquierda (o primera) de la salida binaria,
    %como el bit más significativo(o dígito de mayor orden)
    datosDemod=reshape(demodMatriz.',1,[]).'; %Datos
    % recuperados (bits Serializados)
    else
        % La variable selección mapeo tiene otro valor por
        % lo que se regresa al inicio del lazo while
        disp('Mapeo no realizado')
        continue
    end
    % La variable datosDemod es el vector de salida
    disp('Datos Demapeados')
    %% %%%%%%%%%%%%%%%%%% DECODIFICADOR REED SOLOMON
%%%%%%%%%%%%%%%%%%%%
    if correccionErrores==20
        % Calcular el número de bits codificados por símbolo:
        % Esquema RS (5,3,1) (n,k,t)
        NPdatos=40; % número de portadora de datos por símbolo OFDM
        NRS=5;
        KRS=3;
        tasaRS=KRS/NRS;
        Nbits=tasaRS*NPdatos; % bits de datos de entrada por cada símbolo OFDM
        % Rellenar los datos:
        m=8; % tamaño de símbolo (bits). Columnas de la matriz
        % Relleno de ceros para hacer múltiplo de 8 y 5 (m y NRS)
        k=ceil(length(datosDemod)/(m*NRS)); % Calcular el número de grupos
        % de bits a decodificar
        %ceil=redondea la longitud al inmediato superior entero
        longiPadd=m*NRS*k-length(datosDemod); % Longitud de relleno
        v_relleno=zeros(longiPadd,1); % relleno de ceros
        datosRell=[datosDemod;v_relleno]; % Datos serializados con
        % relleno de ceros
        datINbin=reshape(datosRell,m,[]).';
        datDec=bi2de(datINbin,'left-msb').'; % Datos decimales codificados
        %bi2de, paso de binario a decimal
        LdatCod=length(datDec);%obtengo longitud de datDec
        datDecodReed=[];
        for i=1:NRS:LdatCod
            auxD=datDec(i:i+NRS-1); % Tomar de NSR en NRS símbolos decim
            datGalois=gf(auxD,m);
            % Campo de Galois de la matriz auxDecimal con el tamaño de símbolo
            %para relleno de datos, m tiene que ser entero de 1 a 16
            decRS=rsdec(datGalois,NRS,KRS); % Decodificar
            %rsdec, decodificador Reed Solomon, lo hace con datGalois, con su
            %respectiva longitud del codificador y símbolos originales
            decRSd=double(decRS.x); % Extraer los datos decodificados del campo gf
            % y pasar a decimal
            %Símbolos a bits
            bitdRS=de2bi(decRSd.',m,'left-msb');
            %de2bi, paso de decimal a binario
            bitdRSs=reshape(bitdRS.',1,[]); % vector en bits serializado
```

```
        datDecodReed=[datDecodReed bitdRSs]; % bits serializados acumulados
        %Agrupar y poner en byte, reshape y obtener el texto
      end

        % El vector de salida es datosDecodConv
        disp('Corrección de errores implementado')
    else
      datDecodReed=datosDemod; % Copiar los datos
    end
    %% %%%%%%%%%%%%%%%%%%%%%%%    MEDIR BER
%%%%%%%%%%%%%%%%%%%%%%%%%
    if(actBER==20)
      % Medición del BER activado
      LongDatDecod=length(datDecodReed); % longitud del archivo
      if (longDatosBER<=LongDatDecod)
        % Longitud correcta y se puede recortar los datos y medir el
        % BER
        datosSerializadosRx=datDecodReed(1:longDatosBER);
        bitsErrado=0;
        % la variable longDatosBER es la longitud del archivo y se
        % define en el botón Abrir Archivo
        for aux=1:longDatosBER
          if(datosSerializadosRx(aux)~=datosMedirBER(aux))
            bitsErrado=bitsErrado+1;
          end
        end
        disp('Medición del BER:');
        % Imprimir la cantidad de bits errados sobre la longitud total:
        v_BER=bitsErrado/longDatosBER; % Valor de BER
        fprintf('Bits errados/Bits totales = %g/%g \n',...
          bitsErrado,longDatosBER);
        fprintf('BER = %g \n',v_BER);

      else
        disp('Archivo Recuperado, BER no medido')
        disp('Longitud del archivo recuperdado menor al original');
      end
    else
      % Copiar los datos de entrada en el vector de salida
      datosSerializadosRx=datDecodReed;
    end
    % El vector de salida es datosSerializadosRx
    %% %%%%%%%%%%%%%%%%%%%%%%%   RECUPERAR TEXTO
%%%%%%%%%%%%%%%%%%%%%%%
    % Relleno de ceros:
    bPc=8; % Bits por cada caracter
    numC=ceil(length(datosSerializadosRx)/8); %Número caracteres
    LongRellnumC=bPc*numC-length(datosSerializadosRx); % Longitud
    % de relleno para hacer múltiplo de 8
    rellenoC=zeros(LongRellnumC,1); % Vector de relleno ceros
    datosSerializados=[datosSerializadosRx ; rellenoC]; %Rellenar
    % en caso de ser necesario
    % Pasar de los bits serializados a una matriz, en donde las
    % filas contiene a los caracteres en binario:
    datosMtz=reshape(datosSerializados,[8 numC]).'; % numC filas
    % x 8 columnas
    % Pasar de binario a decimal, los datos de cada fila:
    datosTexto=[]; % Inicializar vector
    for i=1:numC
```

```
        % Se pasa cada fila a decimal y se almacena en datosTexto
        datosTexto(1,i)=double(bi2de(datosMtz(i,:),'left-msb'));
    end
     %Texto Original
    Texto_Original=transpose(x)
    % Pasar de decimal a caracteres:
    disp('Texto recuperado: ');
    Texto_Recuperado=char(datosTexto) % Texto recuperado
    disp('Proceso recuperacion de texto terminado');
    toc % Fin de medir el tiempo de procesamiento
    pause(intTimeR) % Pause para esperar un momento antes de volver a
    % recibir
end
```

ANEXO IX

transmisor_Turbocodigo.m

```
%% TRANSMISOR TURBO CÓDIGO
%% Parámetros para seleccionar algoritmos y controlar la transmisión
seleccionMapeo=4; % Variable para seleccionar el tipo de mapeo a utilizar,
% BPSK (1), QPSK (2), 16-QAM(3), 64-QAM(4)
correccionErrores=20; % Variable para activar (20) o desactivar (10) la
% corrección de errores
intTime=3; %Intervalo de tiempo en segundos, para esperar entre
% transmisiones
txRealTime=20; % Se setea la variable para ingresar al lazo while de
% transmisión repetitiva cada intTime segundos
%% Seleccionar archivo de texto y procesarlo para transmitir
nombreArchivo=uigetfile('*.txt;','Seleccionar archivo a transmitir');
% La variable nombreArchivo toma el valor de 0 cuando no se ha
% seleccionado ningun archivo y toma el nombre del archivo cuando
% si se ha seleccionado un archivo
if nombreArchivo==0
    % No se ha seleccionado ningun archivo
    disp('Archivo no seleccionado'); % Se utiliza para informar
else
    %% %%%%%%%%%%%%%%%%%%% ABRIR ARCHIVO DE TEXTO
%%%%%%%%%%%%%%%%%%%%
    % Se ha seleccionado un archivo y se procede a abrir el mismo
    fID = fopen(nombreArchivo); % Abrir el archivo
    datosAbiertos=fread(fID,'*char'); % Leer los caracteres
    msg=reshape(dec2bin(datosAbiertos,8).'-'0',1,[]);%Transformo a binario
    fclose(fID); % Cerrar el archivo
    %% %%%%%%%%%%%%%%%%%%%% SERIALIZAR DATOS
%%%%%%%%%%%%%%%%%%%%%%%%%%
    textoBinario = dec2bin(datosAbiertos,8); % representacion de cada
    % caracter con 8 bits
    % Para serializar los datos se utiliza dos lazos for(). El primer
    % lazo for() recorre cada fila, length(textoBinario) representa el
    % número de filas. El segundo lazo for() en cada fila recorre todos
    % los elementos (8 columnas). Y con una variable auxiliar (aux) se
    % almacena los datos en la posición que dicte la variable aux.
    aux=1; % Inicializar variable auxiliar
    for i = 1:length(textoBinario)
        for j=1:8
            textoBitsSerial(aux)=textoBinario(i,j); % tipo char
            aux=aux+1; % Sumar 1 en la variable auxiliar
        end
```

```
end
% Para convertir los datos del tipo char al tipo str se utiliza un
% lazo for para recorrer todos los elementos del vector serializado
for aux=1:length(textoBitsSerial)
    % Se utiliza una variable auxiliar para almacenar los datos
    datosSerializados(aux)=str2double(textoBitsSerial(aux));
end
disp('Datos Serializados') % Mostrar un msj en el Command Window
%% %%%%%%%% CODIFICADOR CONVOLUCIONAL (Corrección de errores)%%%%%%%%%
datosBina=datosSerializados; % Copiar los datos serializados en una
% nueva variable
% Se utiliza un lazo condicional if() para determinar si se utiliza la
% correción de errores a nivel de bit
if correccionErrores==10
    % No se ha seleccionado la corrección de errores
    datosBinario=datosBina; % Se copia los datos serializados en la
    % variable de salida de esta etapa y por ende la entrada de la
    % siguiente etapa
elseif correccionErrores==20
    % Se ha seleccionado la corrección de errores.
    % Se define los valores a considerar de las etapas posteriores:
    NPdatos=40; % número de portadora de datos por símbolo OFDM
    % Dependiendo de la opción de mapeo se asigna el valor de puntos en
    % la constelación (M)
    if (seleccionMapeo==1)
        % BPSK
        M = 2; % 2 para BPSK
    elseif (seleccionMapeo==2)
        % QPSK
        M = 4; % 4 para QPSK
    elseif (seleccionMapeo==3)
        % 16-QAM
        M = 16; % 16 para 16-QAM
    elseif (seleccionMapeo==4)
        % 64-QAM
        M = 64; % 64 para 64-QAM
    end
    Nbps_M=log2(M); % Se calcula los bits por símbolo en según la
    % opción de mapeo que se ha escogido
    %Valores para realizar la codificación
    mc=4; % m es el tamaño de registro de la memoria
    g_p=[13 15 17]; % Polinomios generadores
    Nbits=40; % número de bits serializados a codificar
    % Creación de la estructura de Trellis:
    P_Trellis = poly2trellis(mc, g_p);
    % Creación del codificador para turbo códigos
    intrlvrIndices = Nbits:-1:1; %Permutación de números enteros.
    %Sentencia codificador Turbo código
    turboenc = comm.TurboEncoder(P_Trellis,intrlvrIndices);
    %comm.TurboEncoder= codifica la señal utilizando un esquema de codificación
    %concatenado paralelo, se maneja con la estructura de Trellis e
    %intrlvrIndices que es un índice de entrelazado especificado como un
    %vector de columna de enteros
    % Cálculo para rellenar el vector de entrada, y así obtener un vector
    % que sea múltiplo de Nbits:
    NumSimb=ceil(length(msg)/Nbits); % Número de simbolos OFDM
    longitudPadd=Nbits*NumSimb-length(msg); % Longitud de relleno
    vector_padd=zeros(1,longitudPadd); % vector de relleno de ceros
    datosConPadd=[msg vector_padd]; % Datos serializados con relleno
```

```
        % Codificador:
        datosCodConv=[]; % se inicializa la variable que va a almacenar los
        % datos codificados y serializados
        % Se recorre todos los símbolos OFDM con un lazo for
        for i=1:NumSimb
            auxDatosCc = datosConPadd(1,(Nbits*(i-1)+1):Nbits*i); % variable
            % auxiliar para almacenar los datos a codificar
            %Codificador Turbo código;
            codConv = turboenc(transpose(auxDatosCc));
            datosCodConv=[datosCodConv;CodConv]; % Datos de salida
        end
        datosBinario=datosCodConv; % Copiar los datos de salida en una variable
        disp('Codificación Convolucional Agregada') % Mensaje informativo
    end
    %% %%%%%%%%%%%%%%%%%%%%%%%%%%%% MAPEO
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % El vector de entrada de esta etapa es datosBinario y el de salida es
    % datosMapeados
    % El valor en la variable global seleccionMapeo define el tipo de
    % mapeo a utilizar BPSK (10), QPSK (20), 16-QAM(30) o 64-QAM(40)
    if(seleccionMapeo==1)
    % Mapeo BPSK
    % los datos de entrada es un vector columna
    modBPSK=comm.BPSKModulator(pi/4);
    %comm.BPSKModulator es una modulación usando método BPSK donde usamos
    %un valor de pi/4 para la fase del punto cero de constelación
    datosMapeados=modBPSK(datosBinario);
    disp('Técnica de corrección de errores: Turbo código')
    disp('Tipo de Mapeo: BPSK')
elseif(seleccionMapeo==2)
    % Mapeo QPSK
    M = 4; % orden de modulación
    Nb=log2(M); % Bits por símbolo, se utiliza 2 bits por símbolo
    % Hay que asegurarse que la longitud de los datos de entrada sea
    % múltiplo de Nb, por lo que se rellena de ceros cuando no es
    % múltiplo de Nb.
    % Se calcula la longitud de relleno y se almacena en long_padd0
    long_padd0=ceil(length(datosBinario)/Nb)*Nb-length(datosBinario);
    %ceil=redondea la longitud al entero inmediato superior
    paddQ=zeros(1,long_padd0); % Vector de relleno con ceros
    datosbin=[datosBinario paddQ]; % Vector con relleno
    %Pasar de binario a símbolos decimales:
    dataMatriz=reshape(datosbin,Nb,[]).'; % Matriz de Nb columnas y
    % cada fila corresponde a un símbolo
    dataSimb=bi2de(dataMatriz,'left-msb'); % Símbolos a modular
    % bi2de, transforma binario a decimal
    % Factor de normalización = 1/sqrt(2)
    datosMapeados = (1/sqrt(2))*pskmod(dataSimb,M,pi/4,'gray'); % Datos
    % mapeados con QPSK
    %pskmod=modulación por desplazamiento de fase de la entrada dataSimb
    %con una orden de modulación M, y una fase inial igual a pi/4
    %Gray significa el orden de símbolos, que esta codificada en orden gray
    disp('Técnica de corrección de errores: Turbo código')
    disp('Tipo de Mapeo: QPSK')
elseif (seleccionMapeo==3)
    % Mapeo 16-QAM
    M = 16; % Puntos en la constelación (símbolos)
    Nb=log2(M); % Bits por símbolo, se usan 4 bits por símbolo
    % Hay que asegurarse que la longitud de los datos de entrada sea
```

```
    % múltiplo de Nb, por lo que se rellena de ceros cuando no es
    % múltiplo de Nb.
    % Se calcula la longitud de relleno y se almacena en long_padd0
    long_padd0=ceil(length(datosBinario)/Nb)*Nb-length(datosBinario);
    %ceil=redondea la longitud al entero inmediato superior
    paddQ=zeros(long_padd0,1); % Vector de relleno con ceros
    datosbin=[datosBinario;paddQ]; % Vector con relleno
    %Pasar de binario a símbolos decimales:
    dataMatriz=reshape(datosbin,Nb,[]).'; % Matriz de Nb columnas y
    % cada fila corresponde a un símbolo
    dataSimb=bi2de(dataMatriz,'left-msb'); % Símbolos a modular
    % Modulador:
    % Factor de normalización = 1/sqrt(10)
    datosMapeados = (1/sqrt(10))*qammod(dataSimb,M); % Modulación 16-QAM
    %qammod=modula la señal dataSimb con QAM con el orden de modulación M
    disp('Técnica de corrección de errores: Turbo código')
    disp('Tipo de Mapeo: 16-QAM')
elseif (seleccionMapeo==4)
    % Mapeo 64-QAM
    M = 64; % orden de modulación
    Nb=log2(M); % Bits por símbolo, se usan 6 bits por símbolo
    % Hay que asegurarse que la longitud de los datos de entrada sea
    % múltiplo de Nb, por lo que se rellena de ceros cuando no es
    % múltiplo de Nb.
    % Se calcula la longitud de relleno y se almacena en long_padd0
    long_padd0=ceil(length(datosBinario)/Nb)*Nb-length(datosBinario);
    %ceil=redondea la longitud al entero inmediato superior
    paddQ=zeros(long_padd0,1); % Vector de relleno con ceros
    datosbin=[datosBinario;paddQ]; % Vector con relleno
    %Pasar de binario a símbolos decimales:
    dataMatriz=reshape(datosbin,Nb,[]).'; % Matriz de Nb columnas y
    % cada fila corresponde a un símbolo
    dataSimb=bi2de(dataMatriz,'left-msb'); % Símbolos a modular
    % bi2de, binario a decimal
    % Factor de normalización = 1/sqrt(42)
    datosMapeados = (1/sqrt(42))*(qammod(dataSimb,M)); % Datos
    % mapeados con 64-QAM
    %qammod=modula la señal dataSimb con QAM con el orden de modulación M
    disp('Técnica de corrección de errores: Turbo código')
    disp('Tipo de Mapeo: 64-QAM')
end
    %% %%%%%%%%%%%%%%%%%%%%%% CREAR SIMBOLO OFDM
%%%%%%%%%%%%%%%%%%%%%%%%%%
    % El vector de entrada de esta etapa es datosMapeados, el cual es un
    % vector tipo columna.
    % El vector de salida es simbolosOFDMCreados
    % La estructura del símbolo OFDM consta de una subportadora DC,
    % 4 pilotos, 19 de guarda y 40 de datos.
    nPdatos=40; % Número de portadoras de datos
    % El vector de entrada debe ser múltiplo de nPdatos, cuando no se
    % se cumple dicha condición se realiza un relleno con los datos del
    % inicio al final de vector. Esto con el objetivo de hacer que el
    % vector sea múltiplo de nPdatos:
    longDatos=length(datosMapeados); % Longitud del vector de entrada
    nSimbOFDM=ceil(longDatos/nPdatos); % Calcular el número de símbolos
    longPadd=nSimbOFDM*nPdatos-longDatos; % Calcular la longitud del
    % relleno
    datosEntradaPadd=datosMapeados; % Copio los datos de entrada en una
    % nueva variable auxiliar
```

```
    % Si la longitud de relleno es cero no se realiza ninguna operación
    % adicional, si es mayor que cero se realiza el relleno
    if(longPadd>0)
        % Se hace el relleno con los datos del inicio del vector de entrada
        datosEntradaPadd(end+1:end+longPadd)=datosMapeados(1:longPadd);
    end
    % Se procede a crear los simbolos OFDM recorriendo en múltiplos de
    % nPdatos, mediante un lazo for:
    numeroSimbolos=1; % Inicializo la variable que sirve para almacenar los
    % símbolos OFDM creados, en una matriz.
    longDatosPadd=length(datosEntradaPadd); % Longitud de los datos de
    % entrada con relleno
    for i=1:nPdatos:longDatosPadd
        datosAux=datosEntradaPadd(i:i+nPdatos-1); %Copio los datos nPdatos,
        % es decir los 40 datos a introducirse dentro del símbolo OFDM
        DC=0;% Valor para la subportadora DC
        % Las variables p1, p2, p3, p4 son los valores que toman las
        % subportadoras piloto, usadas para poder hacer una estimación del
        % canal en recepción
        p1=1;
        p2=1;
        p3=1;
        p4=1;
        simbOFDM=zeros(64,1); % Inicializo símbolo OFDM, con todos los
        % valores en cero
        simbOFDM(1)=DC; % Portadora DC
        % Asignación del valor a cada subportadora de datos, y a las
        % subportadoras piloto:
        simbOFDM(2:7)=datosAux(21:26);
        simbOFDM(8)=p1; % Piloto
        simbOFDM(9:21)=datosAux(27:39);
        simbOFDM(22)=p2; % Piloto
        simbOFDM(23)=datosAux(40);
        % simbOFDM de 24 a 42 son de guarda
        simbOFDM(43)=datosAux(1);
        simbOFDM(44)=p3;% Piloto
        simbOFDM(45:57)=datosAux(2:14);
        simbOFDM(58)=p4; % Piloto
        simbOFDM(59:64)=datosAux(15:20);
        simbolosOFDMCreados(1:64,numeroSimbolos)=simbOFDM; % Matriz de
        % salida que almacena el símbolo OFDM completo, según la columna
        % que dicte la variable numeroSimbolo
        numeroSimbolos=numeroSimbolos+1; % Aumento en uno la variable
    end
    numeroSimbolos=numeroSimbolos-1; % Disminuyo en uno el valor de la
    % variable para que concuerde con la cantidad total de símbolos OFDM
    disp('Símbolos OFDM creados');
    %% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%% IFFT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % En esta sección se realiza la modulación de los símbolos OFDM
    % mediante la ifft. La matriz de entrada es simbolosOFDMCreados y la de
    % salida es simbolosOFDM.
    % La función ifft admite como entrada un vector columna, pero tambien
    % una matriz. En la matriz aplica la ifft a cada columna. Como la
    % matriz de entrada contiene un spimbolo en cada columna se puede
    % aplicar directamente la función ifft
    simbolosOFDM=ifft(simbolosOFDMCreados); % matriz de salida
    disp('IFFT implementada');
```

```
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%% AÑADIR PC
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% El prefijo cíclico (PC) se añade copiando la parte final de cada
% símbolo, al inicio del mismo.
% La matriz de entrada es simbolosOFDM y a la salida se tiene una
% matriz llamada simbolosOFDMpc con los símbolos en cada columna y
% un vector contodos los símbolos serializados llamado datosEnviar
% Con un lazo for se recorre cada símbolo modulado, se añade el PC
% y se serializa los datos
datosEnviar=[]; % Variable que almacenara los datos serializados
for auxPC=1:numeroSimbolos
    % Añadir el prefijo cíclico, de tamaño igual a la cuarta parte del
    % símbolo:
    simbolosOFDMpc(1:80,auxPC)=[simbolosOFDM(49:64,auxPC);simbolosOFDM(:,auxPC)];
    datosEnviar=[datosEnviar;simbolosOFDMpc(:,auxPC)]; % Datos
    % serializados
end
disp('Prefijo cíclico (PC) añadido');
%% %%%%%%%%%%%%%%%%%%%%%%%%%% AÑADIR PREAMBULO
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Añadimos el preambulo dado por el estandar 802.11a antes de los
% símbolos OFDM serializados que contiene datosEnviar
% Se inicia por definir parte de la sección corta y larga en el dominio
% de la frecuencia para luego obtener el preambulo en el dominio del
% tiempo:
% 4 secuencias cortas (parte de LSTF) en el dominio de la frecuencia
parteCorta64=[0;0;0;0;-1.4720-1.4720i;0;0;0
    -1.4720-1.4720i;0;0;0;1.4720+1.4720i;0;0;0
    1.4720+1.4720i;0;0;0;1.4720+1.4720i;0;0;0
    1.4720+1.4720i;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
    1.4720+1.4720i;0;0;0;-1.4720-1.4720i;0;0;0
    1.4720+1.4720i;0;0;0;-1.4720-1.4720i;0;0;0
    -1.4720-1.4720i;0;0;0;1.4720+1.4720i;0;0;0];
% Secuencia larga (parte de LLTF) en el dominio de la frecuencia
parteLarga=[...
    0;1;-1;-1;1;1;-1;1;-1;1;-1;-1;-1;-1;-1;1
    1;-1;-1;1;-1;1;-1;1;1;1;1;0;0;0;0;0
    0;0;0;0;0;0;1;1;-1;-1;1;1;-1;1;-1;1
    1;1;1;1;1;-1;-1;1;1;-1;1;-1;1;1;1;1];
% Conversión del dominio de la frecuencia al tiempo, y creación de la
% sección corta completa (LSTF):
parteCorta64Time=ifft(parteCorta64); % pasar al dominio del tiempo
parteCorta80Time=parteCorta64Time(49:64);% copiar las ultimas
% 16 muestras al inicio de la variable
parteCorta80Time(17:80)=parteCorta64Time; % 5 secuencias cortas
preambulo=parteCorta80Time; % copiar las 5 secuencias cortas
preambulo(81:160)=parteCorta80Time; % añadir las otras 5 secuencias
% cortas para obtener la sección corta (LSTF)
% Conversión del dominio de la frecuencia al tiempo, y adición de PC
% de la sección larga:
parteLargaTime=ifft(parteLarga); % conversión al dominio del tiempo la
% secuencia larga
preambulo(161:192)=parteLargaTime(33:64); % PC en el preámbulo
preambulo(193:256)=parteLargaTime; % primera secuencia larga
preambulo(257:320)=parteLargaTime; % segunda secuecia larga y se obtiene
% el preámbulo completo.
% Unir el preambulo a los símbolos OFDM serializados:
datosConPre=[preambulo;datosEnviar]; % Preambulo con símbolos OFDM
% creados previamente
```

```
    disp('Preámbulo 802.11a añadido');
    % Inicializar con ceros una variable auxiliar, que tenga el tamaño del
    % bloque a enviar:
    auxDat=zeros(tamanioBloque,1); % inicializar variable
    [longDat,~]=size(datosConPre); % longitud de los datos a enviar
    auxDat(1:longDat,1)=datosConPre; % copiar el preambulo con los símbolos
    % OFDM y los demás datos del bloque se mantienen en cero
    datosEnviarBloque=auxDat;% Datos de salida a enviar
    disp('Datos Procesados')
    %% %%%%%%%%%%%%%%%% NORMALIZAR LOS DATOS A TRANSMITIR %%%%%%%%%%%%%%%%
    % Normalizar los datos a transmitir:
    % Se evalua la parte real y la parte imaginaria de los vectores, para
    % encontrar el módulo máximo de la parte real e imaginaria.
    v_maxR=max(abs(real(datosEnviarBloque))); % Valor máximo, parte real
    v_maxI=max(abs(imag(datosEnviarBloque))); % Valor máximo, parte imag
    % Con un condicional se evalúa si la parte imaginaria o la parte real
    % tiene el valor máximo para normalizar los datos
    if(v_maxR<v_maxI)
        valorMax=v_maxI; % La parte imaginaria tiene el máximo valor abs
    else
        valorMax=v_maxR; % La parte real tiene el máximo valor absoluto
    end
    % Normalizo y almaceno los datos a transmitir en una variable auxiliar
    aux_TxNorm=datosEnviarBloque/valorMax;
    % Convertimos los datos a un tipo single que emplea 32 bits, mientras
    % que double emplea 64 bits para la representacion de cada valor:
    auxTx=single(aux_TxNorm);
    %% %%%%%%%%%%%%%%%%%%%% TRANSMITIR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Se ingresa al lazo para transmitir cada intervalo intTime:
    while (txRealTime==20)
        % se procede a transmitir el bloque de datos OFDM utilizando el
        % objeto de transmisión txAdalm()
        txAdalm(auxTx(:,1));
        % mostramos un mensaje de transmision terminada
        disp('Datos transmitidos...');
        pause(intTime)
    end
end
```

ANEXO X

receptor_Turbocodigo.m

```
%% RECEPTOR TURBO CÓDIGOS
%% Parámetros para seleccionar algoritmos y controlar la recepción
seleccionMapeo=4; % Variable para seleccionar el tipo de mapeo a utilizar,
% BPSK (1), QPSK (2), 16-QAM(3), 64-QAM(4)
correccionErrores=20; % Variable para activar (20) o desactivar (10) la
% corrección de errores
umbralRx=0.0040; % Umbral para la recepción, tiene que ser mayor a 0.0026
intTimeR=10; %Intervalo de tiempo en segundos, para esperar cuando se haya
% recuperado el texto
rxRealT=20; % Se setea la variable para ingresar al lazo while de recepción
% en tiempo real
%% %%%%%%%%%%%%%% SELECCIÓN DE ARCHIVO PARA MEDIR BER
%%%%%%%%%%%%%%%%%%%%
nombreArchivo=uigetfile('*.txt;','Seleccionar archivo a transmitir');
% La variable nombreArchivo toma el valor de 0 cuando no se ha
```

```
% seleccionado ningun archivo y toma el nombre del archivo cuando
% si se ha seleccionado un archivo
if nombreArchivo==0
   % No se ha seleccionado ningun archivo
   disp('Archivo no seleccionado'); % Se utiliza para informar
   actBER=10; % Variable para activar o desactivar(10) la medición del BER
else
   % Se ha seleccionado un archivo y se procede a abrir el mismo
   fid =fopen(nombreArchivo);
   x=fread(fid,'*char'); % Datos Originales
   fclose(fid);
   textoBinario = dec2bin(x,8);% representar cada caracter con 8 bits
   % Serializar:
   tamanioTexto=length(textoBinario);
   aux=1;
   for i = 1:tamanioTexto
      for j=1:8
         textoBitsSerializado(aux)=textoBinario(i,j);
         aux=aux+1;
      end
   end
   datosSerializadosTxOrig=textoBitsSerializado;
   % De char a double:
   for aux=1:length(datosSerializadosTxOrig)
      % Datos originales en bits:
      datosMedirBER(aux)=str2double(datosSerializadosTxOrig(aux));
   end
   longDatosBER=length(datosMedirBER); % longitud de los datos abiertos
   % para obtener el BER
   actBER=20; % Variable para activar(20) o desactivar la medición del BER
end
%% Recepción en tiempo real:
% En el lazo while se implementa todos los procesos de recepción en tiempo
% real
while (rxRealT==20)
   clearvars datosRecibidos; % Limpiar la variable
   %% %%%%%%%%%%%%%%%%%%%%%%    RECIBIR DATOS
%%%%%%%%%%%%%%%%%%%%%%%%
   for i=0:6
      vaciar=rxAdalm(); % Este vector no se utiliza, solo muestrea
      % el canal varias veces para que en caso de existir algún
      %buffer este se elimine
   end
   condAux=0; % Condición auxiliar, para estar dentro del while
   %Recepcion de los datos:
   while(condAux==0)
      datosRxComplejos= rxAdalm(); % Guarda la señal recuperada del
      % canal inalámbrico
      datosRecibidos=double(datosRxComplejos); % de single a double
      v_max=max(abs(datosRecibidos)); % Máximo valor de la magnitud
      % de cada dato recibido
      % Imprimir la magnitu máxima:
      fprintf('Valor máximo de las muestras: %8.5f\n',v_max)
      % Verificar si se supera el umbral establecido:
      if(v_max>umbralRx)
         datosRxAux=rxAdalm(); % Guarda la señal recuperada
         datosRec2=double(datosRxAux); % De single a double
         % Verificar si se supera el umbral:
         if (max(abs(datosRec2))>umbralRx)
```

```
            % Guardar en el mismo vector
            datosRecibidos=[datosRecibidos;datosRec2];
        end
        condAux=1; % Cambiar el valor para salir del lazo while
    end
end % Final del bucle while
disp('Datos recibidos del canal inalámbrico')
%%  %%%%%%%%%%%%%%%%%%%%%%%   RECORTAR DATOS
%%%%%%%%%%%%%%%%%%%%%%%
tic % Para medir el tiempo de procesamiento
% Se recorre el vector de entrada y  se obtiene la posición del
% primer valor por arriba del umbral:
pos_inic=0; % Inicializar variable
for i=1:length(datosRecibidos)
    if(max(abs(datosRecibidos(i)))>umbralRx)
        pos_inic=i; % Primera posición arriba del umbral
        break;
    end
end
% Guardar desde 400 datos antes de superar el umbral, hasta el final:
if(pos_inic>400)
    datosRec=datosRecibidos(pos_inic-400:end); % Datos recortados
else
    % El umbral se superó antes de las primeras 400 posiciones, solo
    % se copia los datos
    datosRec=datosRecibidos;
end
datosInv=flip(datosRec); % Invierte el vector, tal que los datos
% finales queden al inicio, con esto se puede repetir el proceso
% anterior.
% Se recorre el vector de invertido y  se obtiene la posición del
% primer valor por arriba del umbral:
pos_fin=0; % Inicializar variable
for i=1:length(datosRec)
    if (max(abs(datosInv(i)))>umbralRx)
        pos_fin=i; % Primera posición arriba del umbral
        break
    end
end
% Guardar desde 200 datos antes de superar el umbral, hasta el final:
if(pos_fin>200)
    datosInvRec=datosInv(pos_fin-200:end); % Recortar datos
else
    % El umbral se superó antes de las primeras 200 posiciones, solo
    % se copia los datos
    datosInvRec=datosInv;
end
datosRecortados=flip(datosInvRec); % Invertir nuevamente el vector
disp('Datos recortados')
%% %%%%%%%%%%%%%%%%%%%%   DETECCIÓN TRAMA OFDM    %%%%%%%%%%%%%%%
datosRX=datosRecortados; % Copiar los datos en una nueva variable
if length(datosRX)>(SamplesPerFrame+600)
    disp('***** Muestras recibidas inválidas *****');
    continue
end
% El preambulo se encuentra al inicio de cada paquete y por tanto
% se trabaja con las primeras muestras, con esto se evita
% procesamiento innecesario
Mmues=1400; % Muestras para calcular la métrica
```

```
    if(length(datosRX)>Mmues)
        r=datosRX(1:Mmues);
    else
        % Si es menor a Mmues se trabaja con todas las muestras
        r=datosRX;
    end
    L = 48; % Múltiplo dela longitud de una secuencia corta (16*x)
    inMax = length(r)-2*L+1; % k+m puede tomar valores de hasta 2L-1
    % el indice maximo a evaluar es la longitud de r menos 2L.
    m=L-1; % Máximo valor de m
    M = zeros(Mmues,1); % Inicializar la variable para mayor velocidad
    % Determinar la métrica M(k) para comprobar si es un paquete OFDM:
    for k=1:inMax
        Numerador = (r(k:k+m)'*r(k+L:k+m+L)); % Sumatorio del numerador
        Denom = sum((abs(r(k+L:k+m+L))).^2); % Sumatorio del denominador
        M(k)=Numerador/Denom; % Métrica
    end
    M2=(abs(M)).^2; % Cuadrado de la magnitud de cada valor
    umbral=0.60; % Umbral para estimar si existio un paquete OFDM, debe
    % ser mayor a 0.40
    tramaEncontrada=0; % 0 trama no encontrada
    auxET=0; % Inicializar variable que almacena la posición en donde
    % se superera el umbral de la métrica
    for i=1:length(M2)
        %Se obtiene la posición del primer valor por arriba del umbral:
        if(M2(i)> umbral)
            auxET=i;
            tramaEncontrada=1; % Trama encontrada
            break;
        end
    end
    datosOFDM=[]; % Iniciar variable que contiene la trama OFDM
    if(tramaEncontrada==1)
        % Para el caso de que se haya encontrado una paquete OFDM
        datosOFDM(:,1)=datosRX(auxET:end);
    else
        % Si no existe un paquete OFDM se empieza de nuevo el bucle while
        disp('***** Trama OFDM NO encontrada *****');
        pause(0.1)
        continue
    end
    % datosOFDM son los datos de salida
    disp('Trama OFDM detectada');
    %% %%%%%%%%%%%%%%%%%%%%%%%%    CORRECCIÓN CFO
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    r = datosOFDM; % Copiar los datos en r para coincidir con la ecuación
    Fs=frecuenciaMuestreo; % Frecuencia de muestreo de los datos recibidos
    N = 128; % máximo valor del indice k
    L=16; % longitud d euna secuencia corta
    m=L-1; % Máxima longitud que puede tomar m
    fEst = zeros(N,1); % Iniciar vector que almacena la estimación de freq
    % Lazo que recorre de k igual a 1 hasta N
    for k=1:N
        P = r(k:k+m)'*r(k+L:k+m+L); % Sumarotio p(k)
        fEst(k) = (Fs/(2*pi*L))*(angle(P)); % Calcular la estimación
        % de frecuencia en la posición k
    end
    estFreq=-mean(fEst); % Negativo del promedio del CFO estimado
    %Corrección de frecuencia con la estimación realizada:
```

```
despFreqObj = comm.PhaseFrequencyOffset('FrequencyOffset',...
    estFreq,'SampleRate',Fs);
% Aplicar la estimacion de CFO a las muestras recibidas para
% corregir el CFO:
datosCorrCFO=despFreqObj(datosOFDM); % Datos corregidos el CFO
%datosCorrCFO=datosOFDM; % Esta linea se usa para evitar corregir en
%frecuencia
disp('Correccion CFO terminada');
%% %%%%%%%%%%%%%%%%%%%% SINCRONIZACIÓN DE SÍMBOLO
%%%%%%%%%%%%%%%%%%
if length(datosCorrCFO)>5000
    datSinCFO=datosCorrCFO(1:5000); % Se toma dolos primeros 5000 datos
% para reducir el tiempo de procesamiento
else
    datSinCFO=datosCorrCFO;
end
% Crear la parte larga del preámbulo 802.11a:
% Secuencia larga (parte de LLTF) en el dominio de la frecuencia
parteLarga=[...
    0;1;-1;-1;1;1;-1;1;-1;1;-1;1;-1;-1;-1;-1;1
    1;-1;-1;1;-1;1;-1;1;1;1;1;0;0;0;0;0
    0;0;0;0;0;0;1;1;-1;-1;1;1;-1;1;-1;1
    1;1;1;1;1;-1;-1;1;1;-1;1;-1;1;1;1;1];
% Conversión del dominio de la frecuencia al tiempo, y adición de PC
% de la sección larga:
parteLargaTime=ifft(parteLarga); % conversión al dominio del tiempo
LLTF=[parteLargaTime(33:64);parteLargaTime;parteLargaTime]; % LLTF
secLLTF80=LLTF(1:80); % Primeras 80 muestras de la LLTF
% Cuadrado del valor absoluto de la orrelacíón cruzada entre
% la señal recibida y secLLTF80:
datosCorrRell=(abs(xcorr(datSinCFO,secLLTF80))).^2;
% Se corrige en tiempo los datos correlacionados, para que
% coincidan con las posiciones de los datos de entrada:
datosCorr=datosCorrRell(length(datSinCFO):end);
% Almacenar la posición de los dos picos:
[~,posPico1]=max(datosCorr); % Posición del primer pico
datosCorr(posPico1)=0; % Eliminar el primer pico para poder
% encontrar el segundo
[~,posPico2]=max(datosCorr); % Posición del segundo pico
% Se comprueba que la distancia entre los dos picos sea 64
if abs(posPico2-posPico1)==64
    % Se encuentra el inicio real del LTTF y de LSTF
    inicioLLTF=min([posPico1 posPico2]); % Se toma la posición
    % menor y esa es la ubicación del primer dato del LLTF
    inicioLSTF=inicioLLTF-160; % Determinar el inicio del preambulo
    % Comprobar que la posición inicial sea mayor a 0
    if (inicioLSTF>0)
        % Recortar los datos
        datosSincronizados=datosCorrCFO(inicioLSTF:end); % Datos
        % de salida
    else
        disp('Falla en la sincronizacion de simbolo (2)')
        continue
    end
else
    disp('Falla de sincronización de símbolo')
    continue
end
% datosSincronizados es los datos de salida
```

```
    disp('Sincronización de símbolo terminado');
    %% %%%%%%%%%%%%%%%%%%%%    CORRECCIÓN DE FASE
%%%%%%%%%%%%%%%%%%%%%%%%
    % Pasar las dos secuencias largas del LLTF del dominio del tiempo al
    % dominio de la frecuencia:
    a=fft(datosSincronizados(257:320)); % segunda sección larga de LLTF
    b=fft(datosSincronizados(193:256)); % Primera sección larga de LLTF
    %  Valores de las secuencias largas a calcular el ángulo:
    angulosAux=[a(2) a(5) a(6) a(8) a(10) a(16) a(17) a(20) ...
        a(53) a(56) a(57) a(59) a(61) a(62) a(63) a(64) ...
        b(2) b(5) b(6) b(8) b(10) b(16) b(17) b(20) ...
        b(53) b(56) b(57) b(59) b(61) b(62) b(63) b(64)];
    angulos=(angle(angulosAux)*180)/pi; % Calcular el ángulo y pasar de
    % radianes a grados
    anguloProm=-mean(angulos); % Negativo del promedio de los ángulos
        % Corregir el desfase
    desFaseObj = comm.PhaseFrequencyOffset('PhaseOffset',anguloProm,...
        'SampleRate',Fs);
    datFase=desFaseObj(datosSincronizados);
    % Se verifica que los datos corregidos sean multiplo de 80,
    % caso contrario se realiza un padding de las ultimas muestras
    longDatF=length(datFase);
    padd80=ceil(longDatF/80)*80-longDatF;
    if(padd80>=1)
        datosCorrFase=[datFase;datFase(end-padd80+1:end)];
    else
        datosCorrFase=datFase;
    end
    % El vector de salida es datosCorrFase
    disp('Corrección de fase terminado');
    %% %%%%%%%%%%%%%%%%%%%%    ELIMINAR PC
%%%%%%%%%%%%%%%%%%%%%%%%
    clearvars simbolosSinPC;
    %Eliminacion del Prefijo ciclico
    nSimOFDM=0;
    tramaDatos=datosCorrFase(321:end,1); %Quitar el preambulo
    for j=1:80:length(tramaDatos)
        nSimOFDM=nSimOFDM+1;
        simbolosSinPC(:,nSimOFDM)=tramaDatos(j+16:j+79);
    end
    disp('Proceso Eliminación PC terminado');
    %% %%%%%%%%%%%%%%%%%%%%%%%%    FFT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Demodulación OFDM (FFT)
    % La función fft aplica la DFT a cada columna de la matriz
    % simbolosSinPC
    simbolosfft=fft(simbolosSinPC); % Demodular los símbolos sin PC
    disp('Proceso FFT terminado');
    %% %%%%%%%%%%%    ESTIMACIÓN DE CANAL Y ECUALIZADOR  %%%%%%%%%%%%%%
    umbralF=2; % Definir umbral
    simbolosEcualizados=[];
    for auxnSimb=1:nSimOFDM
        % Almacenamos cada simbolo OFDM de 64 muestras en una variable
        simbF=simbolosfft(:,auxnSimb); % Símbolo en frecuencia
        % Determinamos el comportamiento del canal sobre las portadoras
        % piloto
        v1=[simbF(8) simbF(22)]; % Valores de pilotos rx
        v2=[simbF(44) simbF(58)]; % Valores de pilotos rx (2)
        % Realizamos el proceso de interpolacion lineal a los datos
```

```
        % anteriores obtenidos
        x1=[8 22]; % Puntos de la muestra
        xq1=(2:23);      % Puntos a estimar
        x2=[44 58];% Puntos de la muestra
        xq2=(43:64);     % Puntos a estimar
        val_interp1 = (interp1(x1,v1,xq1,'linear','extrap')).';
        val_interp2 = (interp1(x2,v2,xq2,'linear','extrap')).';
        % Se aplica la interpolación a los datos
        datEc1=simbF(2:23)./val_interp1;
        datEc2=simbF(43:64)./val_interp2;
        % Almacenamos los datos ecualizados en una nueva variable
        DatosEc=zeros(64,1);
        DatosEc(2:23)=datEc1;
        DatosEc(43:64)=datEc2;
        simbolosEcualizados(:,auxnSimb)=DatosEc;
    end
    nSimOFDM=auxnSimb; % número de símbolos OFDM
    disp('Ecualizacion terminada');
    %% %%%%%%%%%%%% RECUPERAR DATOS DEL SÍMBOLO OFDM    %%%%%%%%%%%%%%
    nPdatos=40; % Número de portadoras de datos
    datosMapeo=zeros(nPdatos*nSimOFDM,1); % Variable que contiene los
    % datos serializados
    for aux=1:nSimOFDM
        % obtener los datos OFDM
        simAux=simbolosEcualizados(:,aux); % copiar en una var. auxiliar
        % Desamblaje del símbolo OFDM:
        datosAux=zeros(40,1); % Inicializar en 0 un vector auxiliar
        %             simAux(1) es Portadora DC
        datosAux(21:26)=simAux(2:7);
        %             simAux(8) es la primera piloto
        datosAux(27:39)=simAux(9:21);
        %             simAux(22) es la segunda piloto
        datosAux(40)=simAux(23);
        %          simAux(24:42) son las portadoras de guarda
        datosAux(1)=simAux(43);
        %         simAux(44) es la tercera piloto
        datosAux(2:14)=simAux(45:57);
        %             simAux(58) es la cuarta piloto
        datosAux(15:20)=simAux(59:64);
        % Serializar los datos:
        datosMapeo((aux-1)*nPdatos+1:aux*nPdatos)=datosAux;
    end
    disp('Proceso desamblaje OFDM terminado')
    %% %%%%%%%%%%%%%%%%%%%%%%%%%%    DEMAPEO
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Con un lazo if se discrimina el demapeo a utilizar
if(seleccionMapeo==1)
    % Demapeo BPSK
    demodBPSK=comm.BPSKDemodulator(pi/4,'DecisionMethod','Log-likelihood ratio'); % Demodulador
    %comm.BPSKDemodulator es un demodulador usando método BPSK, donde se
    %maneja una fase del punto cero de constelación con valor de pi/4,
    %donde DecisionMethod para manejar los valores de simple o double y
    %Log-likelihood ratio se utiliza para especificar que es demodulación
    datosDemod=demodBPSK(datosMapeo); % Demodulación QPSK, bits suaves
    % determinados mediante un método LLR.
elseif(seleccionMapeo==2)
    % Demapeo QPSK
    ModDat=(sqrt(2))*datosMapeo; % Desnormalizar datos
    demodQPSK=comm.QPSKDemodulator('BitOutput',true,...
```

```
        'DecisionMethod','Log-likelihood ratio'); % Demodulados
    % Demodulación QPSK:
    %comm.QPSKDemodulator , demodulador con método QPSK, BitOutput
    %significa que los datos de salida salen
    %en forma de bits, al poner true genera un vector de columna de valores
    %de bits con una longitud igual al doble de la cantidad de
    %símbolos demodulados, DecisionMethod por default es hard y
    %Log-likelihood ratio para que demodule nuestro sistema
    datosDemod=demodQPSK(ModDat); % bits suaves determinados mediante
    %un método LLR
elseif(seleccionMapeo==3)
    % Demapeo 16-QAM
    M = 16; % Puntos en la constelación (símbolos)
    % Los datos filtrados solo se multiplican por sqrt(10)
    ModDat=(sqrt(10))*datosMapeo; % Datos con relleno
    % Demodulación 16-QAM:
    datosDemod=qamdemod(ModDat,M,'OutputType','llr');
    % Pasar de símbolos a bits:
    %qamdemod, demodula ModDat a una orden de modulación M, OutputType
    %especifica el tipo de salida en este caso LLR
elseif(seleccionMapeo==4)
    % Demapeo 64-QAM
    M = 64; % Puntos en la constelación (símbolos)
    ModDat=(sqrt(42))*datosMapeo; % Datos con relleno
    % Demodulación 64-QAM:
    datosDemod=qamdemod(ModDat,M,'OutputType','llr');
    %qamdemod, demodula ModDat a una orden de modulación M, OutputType
    %especifica el tipo de salida en este caso LLR
else
        % La variable selección mapeo tiene otro valor por
        % lo que se regresa al inicio del lazo while
        disp('Mapeo no realizado')
        continue
    end
    % La variable datosDemod es el vector de salida
    disp('Datos Demapeados')
    %%  DECODIFICADOR TURBO CÓDIGO
    if correccionErrores==20
        % Calcular el número de bits codificados por símbolo:
        NPdatos=40; % Portadoras de datos por símbolo OFDM
        % Nb es en número de bits por símbolo mapeado (definido
        % en la etapa demapeo)
        %Valores para Trellis
        mc=4; % m es el tamaño de registro de la memoria
        g_p=[13 15 17]; % Polinomios generadores
        P_Trellis = poly2trellis(mc, g_p);
        Nbits=40; % número de bits serializados a codificar
        n = log2(P_Trellis.numOutputSymbols); %Obteniendo el logaritmo entre
        %Trellis y el número de símbolos de salida del decodificador.
        numTails = log2(P_Trellis.numStates)*n; %Obteniedo el logaritmo entre
        %Trellis y el número de estados en el decodificador
        L_sal = Nbits*(2*n - 1) + 2*numTails; %Longitud del paquete de palabra código de salida.
        intrlvrIndices = Nbits:-1:1; %Permutación de números enteros.
        numiter=4; %Asignando un valor de 4 para el decodificador
        turbodec = comm.TurboDecoder(P_Trellis,intrlvrIndices,numiter); % Decodificador
        %comm.TurboDecoder Decodificador Turbo código que utiliza la estructura
        %de Trellis, intrlvrIndices es un índice de entrelazado que por defecto
        %realiza vector columna de enteros, numiter es un valor de número de
        %iteraciones
```

```
        % Relleno de bits para que el vector de entrada sea multiplo de nBc:
        nGrupos=ceil(length(datosDemod)/L_sal); % Calcular el número de grupos
        % de bits a decodificar
        %%ceil=redondea la longitud al entero inmediato superior
        longiPadd=L_sal*nGrupos-length(datosDemod); % Longitud de relleno
        v_relleno=zeros(longiPadd,1); % relleno de ceros
        datosDemodRell=[datosDemod;v_relleno]; % Datos serializados con
        % relleno de ceros
        % Decodificación por turbo códigos
        datosDecodConv=[]; % Inicilizar vector que almacena los datos
        % decodificados
        % Con un lazo for se recorre los grupos de bits a ser decodificados
        for i=1:nGrupos
            auxDecod = datosDemodRell((L_sal*(i-1)+1):L_sal*i,1); % Se toma el
            % grupo correspondiente
            %Turbo código
            DecodVit = turbodec(-auxDecod); %Obtención de la decodificación
            datosDecodConv=[datosDecodConv;DecodVit]; % Almacenar los bits
            % decodificados
        end
            % El vector de salida es datosDecodConv
            disp('Corrección de errores implementado')
    else
            datosDecodConv=datosDemod; % Copiar los datos
    end
    %% %%%%%%%%%%%%%%%%%%%%%%     MEDIR BER
%%%%%%%%%%%%%%%%%%%%%%%%%%
    if(actBER==20)
        % Medición del BER activado
        LongDatDecod=length(datosDecodConv); % longitud del archivo
        if (longDatosBER<=LongDatDecod)
            % Longitud correcta y se puede recortar los datos y medir el
            % BER
            datosSerializadosRx=datosDecodConv(1:longDatosBER);
            bitsErrado=0;
            % la variable longDatosBER es la longitud del archivo y se
            % define en el botón Abrir Archivo
            for aux=1:longDatosBER
                if(datosSerializadosRx(aux)~=datosMedirBER(aux))
                    bitsErrado=bitsErrado+1;
                end
            end
            disp('Medición del BER:');
            % Imprimir la cantidad de bits errados sobre la longitud total:
            v_BER=bitsErrado/longDatosBER; % Valor de BER
            fprintf('Bits errados/Bits totales = %g/%g \n',...
            bitsErrado,longDatosBER);
            fprintf('BER = %g \n',v_BER);

        else
            disp('Archivo Recuperado, BER no medido')
            disp('Longitud del archivo recuperdado menor al original');
        end
    else
        % Copiar los datos de entrada en el vector de salida
        datosSerializadosRx=datosDecodConv;
    end
    % El vector de salida es datosSerializadosRx
```

```
    %% %%%%%%%%%%%%%%%%%%%%%%   RECUPERAR TEXTO
%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Relleno de ceros:
    bPc=8; % Bits por cada caracter
    numC=ceil(length(datosSerializadosRx)/8); %Número caracteres
    LongRellnumC=bPc*numC-length(datosSerializadosRx); % Longitud
    % de relleno para hacer múltiplo de 8
    rellenoC=zeros(LongRellnumC,1); % Vector de relleno ceros
    datosSerializados=[datosSerializadosRx ; rellenoC]; %Rellenar
    % en caso de ser necesario
    % Pasar de los bits serializados a una matriz, en donde las
    % filas contiene a los caracteres en binario:
    datosMtz=reshape(datosSerializados,[8 numC]).'; % numC filas
    % x 8 columnas
    % Pasar de binario a decimal, los datos de cada fila:
    datosTexto=[]; % Inicializar vector
    for i=1:numC
        % Se pasa cada fila a decimal y se almacena en datosTexto
        datosTexto(1,i)=double(bi2de(datosMtz(i,:),'left-msb'));
    end
    %Texto Original
    Texto_Original=transpose(x) % Texto recuperado
    % Pasar de decimal a caracteres:
    disp('Texto recuperado: ');
    Texto_Recuperado=char(datosTexto) % Texto recuperado
    disp('Proceso recuperacion de texto terminado');
    toc % Fin de medir el tiempo de procesamiento
    pause(intTimeR) % Pause para esperar un momento antes de volver a
    % recibir
end
```