[ANEXO N° 1]

Tabla A1.1 Parámetros del entrelazador interno del Turbo Código

i	*K*	f_1	f_2	*i*	*K*	f_1	f_2	*i*	*K*	f_1	f_2	*i*	*K*	f_1	f_2
1	40	3	10	48	416	25	52	95	1120	67	140	142	3200	111	240
2	48	7	12	49	424	51	106	96	1152	35	72	143	3264	443	204
3	56	19	42	50	432	47	72	97	1184	19	74	144	3328	51	104
4	64	7	16	51	440	91	110	98	1216	39	76	145	3392	51	212
5	72	7	18	52	448	29	168	99	1248	19	78	146	3456	451	192
6	80	11	20	53	456	29	114	100	1280	199	240	147	3520	257	220
7	88	5	22	54	464	247	58	101	1312	21	82	148	3584	57	336
8	96	11	24	55	472	29	118	102	1344	211	252	149	3648	313	228
9	104	7	26	56	480	89	180	103	1376	21	86	150	3712	271	232
10	112	41	84	57	488	91	122	104	1408	43	88	151	3776	179	236
11	120	103	90	58	496	157	62	105	1440	149	60	152	3840	331	120
12	128	15	32	59	504	55	84	106	1472	45	92	153	3904	363	244
13	136	9	34	60	512	31	64	107	1504	49	846	154	3968	375	248
14	144	17	108	61	528	17	66	108	1536	71	48	155	4032	127	168
15	152	9	38	62	544	35	68	109	1568	13	28	156	4096	31	64
16	160	21	120	63	560	227	420	110	1600	17	80	157	4160	33	130
17	168	101	84	64	576	65	96	111	1632	25	102	158	4224	43	264
18	176	21	44	65	592	19	74	112	1664	183	104	159	4288	33	134
19	184	57	46	66	608	37	76	113	1696	55	954	160	4352	477	408
20	192	23	48	67	624	41	234	114	1728	127	96	161	4416	35	138
21	200	13	50	68	640	39	80	115	1760	27	110	162	4480	233	280
22	208	27	52	69	656	185	82	116	1792	29	112	163	4544	357	142
23	216	11	36	70	672	43	252	117	1824	29	114	164	4608	337	480
24	224	27	56	71	688	21	86	118	1856	57	116	165	4672	37	146
25	232	85	58	72	704	155	44	119	1888	45	354	166	4736	71	444
26	240	29	60	73	720	79	120	120	1920	31	120	167	4800	71	120
27	248	33	62	74	736	139	92	121	1952	59	610	168	4864	37	152
28	256	15	32	75	752	23	94	122	1984	185	124	169	4928	39	462
29	264	17	198	76	768	217	48	123	2016	113	420	170	4992	127	234
30	272	33	68	77	784	25	98	124	2048	31	64	171	5056	39	158
31	280	103	210	78	800	17	80	125	2112	17	66	172	5120	39	80
32	288	19	36	79	816	127	102	126	2176	171	136	173	5184	31	96
33	296	19	74	80	832	25	52	127	2240	209	420	174	5248	113	902
34	304	37	76	81	848	239	106	128	2304	253	216	175	5312	41	166
35	312	19	78	82	864	17	48	129	2368	367	444	176	5376	251	336
36	320	21	120	83	880	137	110	130	2432	265	456	177	5440	43	170
37	328	21	82	84	896	215	112	131	2496	181	468	178	5504	21	86
38	336	115	84	85	912	29	114	132	2560	39	80	179	5568	43	174
39	344	193	86	86	928	15	58	133	2624	27	164	180	5632	45	176
40	352	21	44	87	944	147	118	134	2688	127	504	181	5696	45	178
41	360	133	90	88	960	29	60	135	2752	143	172	182	5760	161	120
42	368	81	46	89	976	59	122	136	2816	43	88	183	5824	89	182
43	376	45	94	90	992	65	124	137	2880	29	300	184	5888	323	184
44	384	23	48	91	1008	55	84	138	2944	45	92	185	5952	47	186
45	392	243	98	92	1024	31	64	139	3008	157	188	186	6016	23	94
46	400	151	40	93	1056	17	66	140	3072	47	96	187	6080	47	190
47	408	155	102	94	1088	171	204	141	3136	13	28	188	6144	263	480

[ANEXO N° 2]

Mapeador LTE

De acuerdo a la función utilizada en el programa de simulación, cada símbolo es mapeado según la forma *I + jQ* en los siguientes esquemas de modulación:

Tabla A2.1 Mapeador de la Modulación QPSK

Secuencia binaria	**I**	**Q**
00	$1/\sqrt{2}$	$1/\sqrt{2}$
01	$-1/\sqrt{2}$	$1/\sqrt{2}$
10	$1/\sqrt{2}$	$-1/\sqrt{2}$
11	$-1/\sqrt{2}$	$-1/\sqrt{2}$

Tabla A2.2 Mapeador de la Modulación 8QAM

Secuencia	**I**	**Q**	**Secuencia**	**I**	**Q**
000	-3	1	100	3	1
001	-3	-1	101	3	-1
010	-1	1	110	1	1
011	-1	-1	111	1	-1

Tabla A2.3 Mapeador de la Modulación 16QAM

Secuencia	I	Q	Secuencia	I	Q
0000	-3	3	1000	3	3
0001	-3	1	1001	3	1
0010	-3	-3	1010	3	-3
0011	-3	-1	1011	3	-1
0100	-1	3	1100	1	3
0101	-1	1	1101	1	1
0110	-1	-3	1110	1	-3
0111	-1	-1	1111	1	-1

[ANEXO N° 3]

Código de la herramienta de simulación

En este anexo se presenta una breve síntesis de cada sub-rutina perteneciente al modelo de simulación de la herramienta con la siguiente estructura:

[Variables de salida] = Nombre de la función (Variables de entrada)

Explicación

Posteriormente se encuentra el código empleado para su elaboración.

[BER, BER2] = Prefacio Es la rutina que agrupa a todas las subrutinas recursivas del código, por lo que el mismo se corre en esta instancia. En ella se escoge el tipo de modulación a emplear y por ende la longitud del bloque de transporte, representada por la variable **x**. La variable **BER** corresponde a la tasa de bits errados cuando se aplica la codificación de canal y **BER2** cuando no se aplica.
[x_dataCRC] = CRC_Attachment (x) Es la función encargada de determinar los bits de control correspondiente a la comprobación de redundancia cíclica. La variable **x** representa el bloque de transporte a codificar y **x_dataCRC** es el mismo bloque con los 24 bits de control.
[x_posicion] = EntrelazadoTC (x_dataCRC) **EntrelazadoTC** determina la nueva posición de cada bit del bloque del transporte después de cruzar el *interleaver* del Turbo Código. La variable **x_posicion** almacena el número de la nueva posición del bit en cuestión, por lo que su longitud es igual al del bloque de transporte.

[dataSist, dataParity, dataInterlevd, trellis] = TurboCoding2 (x_dataCRC, x_posicion) Determina la estructura de las tres ráfagas de salida del Turbo Código y de la terminación de trellis. La variable **dataSist** corresponde a la ráfaga sistemática de bits, **dataParity** contiene los bits de paridad y **dataInterlevd** a la ráfaga de bits entrelazados acorde a la variable **x_posicion.** La función utiliza un objeto propio de la herramienta de simulación capaz de calcular la ráfaga de bits salida del Turbo Código, todos concatenados, por lo que hace falta desarrollar el proceso para separar a cada una. Por último, la variable **trellis** representa los doce bits la terminación de trellis que emplea el Turbo Código.
[Vk1, Vk2, Vk3, p] = SubBlockInterleaver (dataSist, dataParity, dataInterlevd) La función simula el primer proceso del mecanismo de *Rate Matching*, determinando los vectores **Vk1**, **Vk2** y **Vk3**, que son las ráfagas de bits de la etapa anterior al cruzar por los bloques de *interleaving*, de acuerdo a su nuevo arreglo en matrices y posteriormente permutadas. La variable **p** almacena la dimensión de la matriz.
[Wk, LB] = BitCollector (Vk1, Vk2, Vk3) Concatena las tres ráfagas anteriores en un único vector **Wk** de longitud **LB**, colocando primero todos los elementos de **Vk1** y después intercalando los elementos de **Vk2** y **Vk3.**
[Ek] = BitSelection (Wk, LB) Calcula el vector de bits a transmitirse **Ek** de acuerdo a los recursos asignados.

Seguido a esta instancia, intervienen procesos propios de la capa física, representados por los códigos que se elaboraron previamente. Dichos procesos son la modulación, la selección del tipo de canal y la demodulación. A pesar de que se muestran posteriormente, intervienen muchas variables y funciones secundarias, las cuales no fueron objeto a desarrollar en este trabajo.

[F] = Inverse_BitSelection (EkDemT, WkDemT)

Es el proceso inverso al descrito en la función **BitSelection**, a partir de los bits que se recibieron y ya demodulados. Esta función determina el nuevo *buffer* **F** a partir de los bits que fueron afectados por el modelo de canal seleccionado (**EkDemT)** y del buffer del transmisor ya demodulado (**WkDemT**).

[v1, v2, v3] = Inverse_Buffer (F)

Se describe como el procedimiento inverso al descrito en la función **BitCollector**. A partir del vector **F** que contiene las tres ráfagas concatenadas, este procedimiento separa a las mismas en los vectores **v1, v2** y **v3**, los que equivaldrían a las variables

Vk1, **Vk2** y **Vk3**, explicados anteriormente.

[Dp, Ds, Di] = SubBlockDeinterleaver (v1, v2, v3, p)

Describe el proceso inverso al de la función **SubBlockInterleaver**. Tomando los vectores **v1**, **v2** y **v3**, realiza una reorganización en matrices contraria a la descrita en el transmisor, resultando en las tres ráfagas **Dp, Ds** y **Di**, las cuales serán decodificadas.

[H] = TurboDecoding (x_posicion, Dp, Ds, Di, trellisDemT)

Es la función encargada de determinar el bloque de transporte decodificado **H** a partir de las ráfagas alteradas de los bits sistemáticos (**Ds**), bits de paridad (**Dp**) y entrelazados (**Di**). Utiliza el mismo esquema de entrelazado que la función **TurboCoding2**, por lo que es necesario la variable **x_posicion**, y la terminación de trellis ya demodulada (**trellisDemT**).

[BER, BER2] = AAM_BER (x_dataCRC, H, x2DemT)

Calcula la tasa de bits errados entre los bits transmitidos y los bits recibidos. La variable **BER** representa la comparación entre los variables **x_dataCRC** y **H**, determinando la tasa de bits errados usando la codificación de canal, mientras que **BER2** es la tasa de bits errados mostrada cuando no se usa dicho mecanismo.

A continuación, se presentan los códigos que describen el funcionamiento del modelo de simulación.

```
function [BER,BER2]= Prefacio
%En esta instancia se correra el codigo y se veran la comparacion
final
%entre la primera rafaga de bits con el CRC y la ultima rafaga al
salir de
%la funcion de TurboDecoding

Entrada=input('Indique el tipo de modulacion a emplear (QPSK=1,
8QAM=2, 16QAM=3) ');

switch Entrada
    %Caso de QPSK
    case 1
        L=1000;
        x=round(rand(1,L));
        x2=x';
    %Caso de 8QAM
    case 2
        L=1512;
        x=[round(rand(1,L-12)) zeros(1,12)];
        x2=x(1:length(x)-12)';
    %Caso de 16QAM
    case 3
        L=2024;
        x=[round(rand(1,L-24)) zeros(1,24)];
        x2=x(1:length(x)-24)';
    otherwise
    Mensaje=char('TIPO DE MODULACION NO VÁLIDA (DEBE ESCOGER ENTRE
QPSK=1, 8QAM=2, 16QAM=3');
    display (Mensaje);
end
%A continuacion se procede a realizar como tal el proceso de Channel
%Coding, conformado por 9 bloques en total, 5 referentes al
transmisor y
%4 referentes a la recepcion

[x_dataCRC]=CRC_attachment(x);
[x_posicion]=EntrelazadoTC(x_dataCRC);
[dataSist,dataParity,dataInterlevd,trellis]=TurboCoding2(x_dataCRC,x
_posicion);
[Vk1,Vk2,Vk3,p]=SubBlockInterleaver(dataSist,dataParity,dataInterlev
d);
[Wk,LB]=BitCollector(Vk1,Vk2,Vk3);
[Ek]=BitSelection(Wk,LB);
[EkMod,WkMod,trellisMod,x2Mod] = Modulator(Ek,Wk,trellis,x2);
%Terminada la etapa como tal de la codificacion de canal entre el
canal de
%transporte y el canal fisico, se procede a simular el proceso de
%simulacion
```

```
[TOFDM,NSy]=AAA_Preliminares;
[Bit,NBits_Group,Pulse,NP_Bit ] = AAB_Entrada_Datos(Ek,TOFDM,NSy);
 Symbolo=EkMod';
%AAC_Mapeo_Simbolos(Bit,NBits_Group,Pulse,NSy,NP_Bit);
 Symbolo2=x2Mod';
[Espectro,Espectro_pil,Espectro2,Espectro_pil2] =
AAD_Agrega_Piloto(Bit,Symbolo,Symbolo2,NSy);
[Espectro_ofdm_conDC,Espectro_ofdm_conDC2] =
AAE_Mapeo_Sub(Bit,Espectro,Espectro2,NSy,Espectro_pil,Espectro_pil2)
;
[s_ofdm,s_ofdm2] =
AAF_Transformada_Inversa(Bit,Espectro_ofdm_conDC,Espectro_ofdm_conDC
2,NSy,Espectro_pil);
[s_OFDM_analog,s_OFDM_analog2,NN,t_ofdm_PC] =
AAG_Agrega_PC(Bit,s_ofdm,s_ofdm2,NSy,Espectro_pil);

%A continuacion se procede a simular varios canales ruidosos, la
unica
%rafaga de interes que pasara por estos canales sera la que
corresponde a
%los bits de informacion. Los otros conjuntos de bits se comunican
con el
%receptor mediante canales de control, lo cual se simplifico
mediante el
%uso del comando AWGN
[senalmulti,Rsenalmulti,senalmulti2,Rsenalmulti2,NP]=AAH_Canal(Bit,s
_OFDM_analog,s_OFDM_analog2,NN,t_ofdm_PC,NSy,Espectro_pil);
[RSisPC,SisPC,RSisPC2,SisPC2]
=AAI_Elimina_PC(Bit,senalmulti,Rsenalmulti,senalmulti2,Rsenalmulti2,
NSy,Espectro_pil);
[CROrig,SROrig,SR_Pilot,CR_Pilot,CROrig2,SROrig2,SR_Pilot2,CR_Pilot2
] =
AAJ_Transformada_Directa(Bit,senalmulti,Rsenalmulti,NSy,Espectro_pil
,RSisPC,SisPC,RSisPC2,SisPC2);
[Pilotos,RPilotos,TxPilotos,Pilotos2,RPilotos2,TxPilotos2]
=AAK_Extrae_Pilotos(Bit,CROrig,SROrig,SR_Pilot,CR_Pilot,NSy,Espectro
_pil,CROrig2,SROrig2,SR_Pilot2,CR_Pilot2,Espectro_pil2);
[simbolos_sinecua,simbolos_sinecua2]=
AAL_Ecualizador(Pilotos,RPilotos,TxPilotos,Bit,CROrig,SROrig,SR_Pilo
t,CR_Pilot,NSy,Espectro_pil,CROrig2);
%Estos son los bits que se comunican mediante canales de control
WkRec=awgn(WkMod,5);
trellisRec=awgn(trellisMod,5);
%A partir de este momento se procede a simular o modelar el proceso
de
%recepcion, comenzando a primera instancia por el demodulador
[EkDemT,WkDemT,trellisDemT,x2DemT] =
Demodulator(Ek,simbolos_sinecua,WkRec,trellisRec,simbolos_sinecua2);
[F]=Inverse_BitSelection(EkDemT,WkDemT);
[v1,v2,v3]=Inverse_Buffer(F);
[Ds,Dp,Di]=SubBlockDeinterleaver(v1,v2,v3,p);
[H]=TurboDecoding(x_posicion,Ds,Dp,Di,trellisDemT);
[BER,BER2]=AAM_BER(x_dataCRC,H,x2DemT);
end
```

```
function [x_dataCRC] = CRC_attachment(x)
%se procede a seleccionar la longitud del vector de entrada
%L=16; %Tipo de prueba
%L=1000; %(normalizada, seria 1000)
%L=1512; %(normalizada, seria 1500)
%L=2024; %(normalizada, seria 2000)

%x= round(rand([1,L]));
%Siempre seran 24 bits de paridad, regidos por el polinomio
generador
Gcrc24A=[1 1 0 1 1 1 1 1 0 0 1 1 0 0 1 0 0 1 1 0 0 0 0 1 1];

x_array=[zeros(1,24) x];
rem=binary_rem(x_array,Gcrc24A);

x_dataCRC=[x rem];
dim=length(x_dataCRC);
%TurboCoding2(x_dataCRC);
end



function x_posicion= EntrelazadoTC(x_dataCRC)
%x_dataCRC= round(rand(1,40));
%x_dataCRC= round(rand(1,1024));
%x_dataCRC= round(rand(1,1536));
%x_dataCRC= round(rand(1,2048));
%[x_dataCRC,K]=CRC_attachment;
K=length(x_dataCRC);
if K==1024 || K==2048
   f1=31;f2=64;
elseif K==1536
   f1=71;f2=48;
elseif K==40
   f1=3;f2=10;
end

Ve=zeros(1,K);
Orden=zeros(1,K);
for i=0:K-1
  c= mod((f1*i+ f2*(i^2)),K);
  Ve(c+1) = x_dataCRC(i+1);
  Orden(i+1)= c+1;
end
  x_interleaved= Ve;
  x_posicion= Orden;

end
```

```
function [dataSist,dataParity,dataInterlevd,trellis]=
TurboCoding2(x_dataCRC,x_posicion)
%x_dataCRC= round(rand(1,40));
%x_dataCRC= round(rand(1,1024));
%x_dataCRC= round(rand(1,1536));
%x_dataCRC= round(rand(1,2048));
%x_dataCRC=CRC_attachment;
%x_posicion=EntrelazadoTC;
hTenc=comm.TurboEncoder('TrellisStructure',poly2trellis(4,...
    [13,15],13),'InterleaverIndices',x_posicion);
dataCod=step(hTenc,x_dataCRC');
largo=length(x_dataCRC);
t=length(dataCod);
g=dataCod';
trellis=g(t-11:t);

Sist=[dataCod(1) zeros(1,(largo-1))];
Parity=[dataCod(2) zeros(1,(largo-1))];
Interlevd= zeros(1,largo);
k=2;
for i=4:3:(t-12-2)
    Sist(mod(i,k)+2)=dataCod(i);
    k=k+1;
end
k=2;
for j=5:3:(t-12-1)
    Parity(mod(j,k)+1)= dataCod(j);
    k=k+1;
end

for p=3:3:(t-12)
    Interlevd(p/3)=dataCod(p);
end
    dataSist=Sist;
    dataParity=Parity;
    dataInterlevd=Interlevd;
end
```

```
function [Vk1,Vk2,Vk3,p]=
SubBlockInterleaver(dataSist,dataParity,dataInterlevd)
%[g,dataSist,dataParity,dataInterlevd,trellis,x_dataCRC]=
TurboCoding2;
l=length(dataSist);

if l==40
    dataSist2=[dataSist zeros(1,(32*ceil(l/32))-l)];
    dataParity2=[dataParity zeros(1,(32*ceil(l/32))-l)];
    %dataInterlevd2=[dataInterlevd zeros(1,(32*ceil(l/32))-l)];
    MatrizA=(reshape(dataSist2,32,2)');
    MatrizB=(reshape(dataParity2,32,2)');
    %MatrizC=(reshape(dataInterlevd2,32,2)');

elseif l==1024
    MatrizA=(reshape(dataSist,32,32)');
    MatrizB=(reshape(dataParity,32,32)');
    %MatrizC=(reshape(dataInterlevd,32,32)');

elseif l==1536
    MatrizA=(reshape(dataSist,32,48)');
    MatrizB=(reshape(dataParity,32,48)');
    %MatrizC=(reshape(dataInterlevd,32,48)');
else
    MatrizA=(reshape(dataSist,32,64)');
    MatrizB=(reshape(dataParity,32,64)');
    %MatrizC=(reshape(dataInterlevd,32,64)');
end
%Se inicializan matrices para que no se pierda ningun dato con la
%permutaci[on
if l==40
    MatrizAA=zeros(2,32);
    MatrizBB=zeros(2,32);

elseif l==1024
    MatrizAA=zeros(32);
    MatrizBB=zeros(32);

elseif l== 1536
    MatrizAA=zeros(48,32);
    MatrizBB=zeros(48,32);

else
    MatrizAA=zeros(64,32);
    MatrizBB=zeros(64,32);

end
%Ahora se realiza el esquema de permutacion de las columnas de las
%matrices MatrizA y MAtrizB
MatrizAA(:,1)=MatrizA(:,1); MatrizAA(:,2)=MatrizA(:,17);
MatrizAA(:,3)=MatrizA(:,9);
MatrizAA(:,4)=MatrizA(:,25); MatrizAA(:,5)=MatrizA(:,5);
MatrizAA(:,6)=MatrizA(:,21);
```

```
MatrizAA(:,7)=MatrizA(:,13); MatrizAA(:,8)=MatrizA(:,29);
MatrizAA(:,9)=MatrizA(:,3);
MatrizAA(:,10)=MatrizA(:,19); MatrizAA(:,11)=MatrizA(:,11);
MatrizAA(:,12)=MatrizA(:,27);
MatrizAA(:,13)=MatrizA(:,7); MatrizAA(:,14)=MatrizA(:,23);
MatrizAA(:,15)=MatrizA(:,15);
MatrizAA(:,16)=MatrizA(:,31); MatrizAA(:,17)=MatrizA(:,2);
MatrizAA(:,18)=MatrizA(:,18);
MatrizAA(:,19)=MatrizA(:,10); MatrizAA(:,20)=MatrizA(:,26);
MatrizAA(:,21)=MatrizA(:,6);
MatrizAA(:,22)=MatrizA(:,22); MatrizAA(:,23)=MatrizA(:,14);
MatrizAA(:,24)=MatrizA(:,30);
MatrizAA(:,25)=MatrizA(:,4); MatrizAA(:,26)=MatrizA(:,20);
MatrizAA(:,27)=MatrizA(:,12);
MatrizAA(:,28)=MatrizA(:,28); MatrizAA(:,29)=MatrizA(:,8);
MatrizAA(:,30)=MatrizA(:,24);
MatrizAA(:,31)=MatrizA(:,16); MatrizAA(:,32)=MatrizA(:,32);

MatrizBB(:,1)=MatrizB(:,1); MatrizBB(:,2)=MatrizB(:,17);
MatrizBB(:,3)=MatrizB(:,9);
MatrizBB(:,4)=MatrizB(:,25); MatrizBB(:,5)=MatrizB(:,5);
MatrizBB(:,6)=MatrizB(:,21);
MatrizBB(:,7)=MatrizB(:,13); MatrizBB(:,8)=MatrizB(:,29);
MatrizBB(:,9)=MatrizB(:,3);
MatrizBB(:,10)=MatrizB(:,19); MatrizBB(:,11)=MatrizB(:,11);
MatrizBB(:,12)=MatrizB(:,27);
MatrizBB(:,13)=MatrizB(:,7); MatrizBB(:,14)=MatrizB(:,23);
MatrizBB(:,15)=MatrizB(:,15);
MatrizBB(:,16)=MatrizB(:,31); MatrizBB(:,17)=MatrizB(:,2);
MatrizBB(:,18)=MatrizB(:,18);
MatrizBB(:,19)=MatrizB(:,10); MatrizBB(:,20)=MatrizB(:,26);
MatrizBB(:,21)=MatrizB(:,6);
MatrizBB(:,22)=MatrizB(:,22); MatrizBB(:,23)=MatrizB(:,14);
MatrizBB(:,24)=MatrizB(:,30);
MatrizBB(:,25)=MatrizB(:,4); MatrizBB(:,26)=MatrizB(:,20);
MatrizBB(:,27)=MatrizB(:,12);
MatrizBB(:,28)=MatrizB(:,28); MatrizBB(:,29)=MatrizB(:,8);
MatrizBB(:,30)=MatrizB(:,24);
MatrizBB(:,31)=MatrizB(:,16); MatrizBB(:,32)=MatrizB(:,32);

A=MatrizAA; B=MatrizBB; %C=MatrizC;
p=size(MatrizAA);
perm=[0 16 8 24 4 20 12 28 2 18 10 26 6 22 14 30 1 17 9 25 5 21 13
29 3 19 11 27 7 23 15 31];
%Las dos primeras rafagas de bits seran
Vk1= reshape(A,1,p(1,1)*p(1,2));
Vk2= reshape(B,1,p(1,1)*p(1,2));

%La tercera rafaga de bits sera
%V3=zeros(1,p(1,1)*p(1,2));
%k=1;
%for j=1:p(1,1)
    %for i=1:32
        %V3(k)=C(j,i);
```

```
        %k=k+1;
    %end
%end
%V33=reshape(C',1,(p(1,1)*p(1,2)));
V33dos=zeros(1,length(dataInterlevd));
    for i=0:length(dataInterlevd)-1

valor=mod(perm(floor(i/p(1,1))+1)+p(1,2)*(mod(i,p(1,1)))+1,(p(1,1)*p
(1,2)));
       V33dos(i+1)=dataInterlevd(valor+1);
    end
     %for i=0:length(dataInterlevd)-1

%V33dos(mod(perm(floor(i/p(1,1))+1)+p(1,2)*(mod(i,p(1,1)))+1,(p(1,1)
*p(1,2)))+1)=dataInterlevd(i+1);
     %end
Vk3=V33dos;
end



function [Wk,LB] = BitCollector(Vk1,Vk2,Vk3)
%[g,Vk1,Vk2,Vk3,trellis,x_dataCRC]=SubBlockInterleaver;
Kpi=length(Vk1);
Buffer=zeros(1,3*Kpi);

for i=0:Kpi-1
    Buffer(i+1)=Vk1(i+1);
    Buffer((Kpi+1)+2*i)=Vk2(i+1);
    Buffer((Kpi+1)+(2*i+1))=Vk3(i+1);
end

Wk=Buffer;
LB=length(Buffer);
end









function [Ek] = BitSelection(Wk,LB)
%[g,Wk,LB,trellis,x_dataCRC]=BitCollector;

if LB==192
    rows=2;
    E=40;
elseif LB==3072
    rows=32;
    E=1000;
```

```
elseif LB==4608
    rows=48;
    E=1500;
else
    rows=64;
    E=2000;
end
rvidx=0;
%rvidx=1;
%rvidx=2;
%rvidx=3;
K0=rows*(2*ceil(LB/(8*rows))*rvidx+2);
k=0;
j=0;
e=zeros(1,E);
while k<E
    %if Wk(mod(K0+j,LB)+1)~=
        e(k+1)=Wk(mod(K0+j,LB)+1);
        k=k+1;
    %end
    j=j+1;
end
    Ek=e;
end



function [EkMod,WkMod,trellisMod,x2Mod] =
Modulator(Ek,Wk,trellis,x2)
hModQPSK=comm.QPSKModulator('BitInput',true);
hMod8QAM=comm.RectangularQAMModulator('ModulationOrder',8,'BitInput'
,true);
hMod16QAM=comm.RectangularQAMModulator('ModulationOrder',16,'BitInpu
t',true);

if length(Ek)==1000
    EkMod=step(hModQPSK,Ek');
    WkMod=step(hModQPSK,Wk');
    trellisMod=step(hModQPSK,trellis');
    x2Mod=step(hModQPSK,x2);
elseif length(Ek)==1500
    EkMod=step(hMod8QAM,Ek');
    WkMod=step(hMod8QAM,Wk');
    trellisMod=step(hMod8QAM,trellis');
    x2Mod=step(hMod8QAM,x2);
else
    EkMod=step(hMod16QAM,Ek');
    WkMod=step(hMod16QAM,Wk');
    trellisMod=step(hMod16QAM,trellis');
    x2Mod=step(hMod16QAM,x2);
end

end
```

```
function [TOFDM,NSy]=AAA_Preliminares
% clc;
% clear;
% close all;

%n=input('INDIQUE EL NÚMERO DE RECURSOS ASIGNADOS AL USUARIO EN ESTE
"SLOT"
%(DE 1 A 6 MAX): ');
n=50;  %numero de recursos para 500 portadoras de datos, 1024
subportadoras en total, 100 pilotos en en ensamble de datos, 424
subportadoras de relleno
Modul=input('INDIQUE EL TIPO DE MODULACION (QPSK=1, 8QAM=2, 16QAM=3)
');

% duración del símbolo OFDM sin el prefijo cíclico
TOFDM=66.667*10^(-6);

switch Modul
    %case 'QPSK'
    case 1
        NSy=4;
   % case '8QAM'
    case 2
        NSy=8;
   % case '16QAM'
    case 3
        NSy=16;

    otherwise
    Mensaje=char('TIPO DE MODULACION NO VÁLIDA (DEBE ESCOGER ENTRE
QPSK=1, 8QAM=2, 16QAM=3');
    display (Mensaje);
end;
end
% se transfieren a la rutina Datos_Entrada, la duración del símbolo
OFDM,
% el número de símbolos del sistema de modulación y el número de
recursos
% asignados al usuario

%AAB_Entrada_Datos(TOFDM,NSy)



function [Bit,NBits_Group,Pulse,NP_Bit ] =
AAB_Entrada_Datos(Ek,TOFDM,NSy)
%[Ek]=Prefacio;
    %Número puntos de muestreo de la senal en el período TOFDM de
    %observación (en algunos caso es necesario modificarlo para
garantizar
    %un entero como número de puntos de muestreo por bit NP_Bit).
```

```
    % Dependiendo del número de recursos asignados n, se establece
el número
    % de subportadoras asignadas al usuario, sin los pilotos, y la
    % modificación de NP donde es necesario.

 Sub_User=500;

 switch NSy

     case 4  %500*2 bits
         NP=1000*5;  % numero de puntos solo para graficar el tren
de pulsos de entrada=n de bits*5 muestras

     case 8  %500*3 bits
         NP=1500*5;

     case 16  %500*4 bits
         NP=2000*5;

 end



    %Amplitud Am de los pulsos
Am=5;

    %Número de bits NBits_Group que se han agrupado de acuerdo al
sistema
    %de modulación escogido
NBits_Group=log2(NSy);

    %Número total de bits Nbits_por_OFDM correspondiente al símbolo
OFDM
Nbits_por_OFDM=NBits_Group*Sub_User;

    %Número puntos de muestreo por bit NP_Bit: es igual al número
total de puntos
    % de muestreo NP dividido por el número total de bits
Nbits_por_OFDM
    %correspondiente al esquema de modulación seleccionado.
NP_Bit=(NP/(Nbits_por_OFDM));

    % Se genera la secuencia de pulsos aleatorios a ser modulados.
Cada pulso
    % tendrá un número de puntos de muestreo igual a NP_Bit
    % que se almacenarán en los elementos del vector Pulse.
    c=1;
    Bit=Ek;
   % Bit=randi([0,1],1,Nbits_por_OFDM);
 % Bit=round(rand(1,Nbits_por_OFDM));
 % Bit=ones(1,Nbits_por_OFDM);
```

```
for k=1:Nbits_por_OFDM
    % c apunta a la posición del vector Pulse que representa en cada
lazo k
    % el comienzo del pulso aleatorio de amplitud +-Am.
    if Bit(1,k)==1, Pulse(1,c:k*NP_Bit)=Am; else
Pulse(1,c:k*NP_Bit)=-Am;end;
    c=c+NP_Bit;
end;

    %   Dt es el período de muestreo de los pulsos generados, igual
al
    %   periodo de observación TOFDM dividido por el número de
elementos
    %   del vector Pulse
Dt=TOFDM/length(Pulse);
t=0:Dt:TOFDM-Dt;

figure(1)
plot(t,Pulse)
ylim([-Am*1.5 Am*1.5])
xlim([0 TOFDM])
xlabel('t')
title('TREN DE BITS DE ENTRADA')
grid on

    %Se procede a pasar la secuencia de pulsos al mapeador
 %AAC_Mapeo_Simbolos(Bit,NBits_Group,Pulse,NSy,NP_Bit)

End

function [Espectro,Espectro_pil,Espectro2,Espectro_pil2] =
AAD_Agrega_Piloto(Bit,Symbolo,Symbolo2,NSy)

Sub_User=500;
n=50;
NPFFT=Sub_User;
f=(0:NPFFT-1)*15*10^3;
f1=(0:(Sub_User+2*n)-1)*15*10^3;
n=50;

%Se establece un factor de peso para las portadoras piloto segun el
%sistema de modulacion escogido

switch NSy
    case 4
           % Ypil=1.2;
            Ypil=5.0;
    case 8
            Ypil=2*1.2;
    case 16
            Ypil=2*1.2;
end;
    % Se efectua la FFT de y
```

```
%Y=(1/NPFFT)*fft(Symbolo,NPFFT);

    %...se agregan las portadoras piloto

%TABLA DE PORTADORAS PILOTO
pil(1,1)=(1+1j)*Ypil;
pil(1,2)=(-1+1j)*Ypil;
pil(1,3)=(-1-1j)*Ypil;
pil(1,4)=(1-1j)*Ypil;

% pil(1,1)=(1+1j)*Ypil;
% pil(1,2)=(1+1j)*Ypil;
% pil(1,3)=(1+1j)*Ypil;
% pil(1,4)=(1+1j)*Ypil;
% pil(1,5)=(1+1j)*Ypil;
% pil(1,6)=(-1+1j)*Ypil;
% pil(1,7)=(-1-1j)*Ypil;
% pil(1,8)=(1-1j)*Ypil;
% pil(1,9)=(1+1j)*Ypil;
% pil(1,10)=(-1+1j)*Ypil;
% pil(1,11)=(-1-1j)*Ypil;
% pil(1,12)=(1-1j)*Ypil;
% pil(1,13)=(1+1j)*Ypil;
% pil(1,14)=(-1+1j)*Ypil;
% pil(1,15)=(-1-1j)*Ypil;
% pil(1,16)=(1-1j)*Ypil;

Espectro=zeros(1,12*n); %porque cada recurso posee 12 portadoras
    %Se genera aleatoriamente el índice del vector pil
%indice=randi([1 2*n],1,2*n);
%indice=randi([1 4],1,2*n);
indice=round(1+rand(1,2*n)*3);
    %Se ubican las portadoras piloto de 6 en 6 posiciones en el
vector Espectro
    %que tiene una longitud 2*n mayor que los vectores Symbolo y Y
con las
    %subportadoras de información y transformados respectivamente.
    %En cada paso k se ubica una portadora seleccionada al azar de
la TABLA
    %DE PORTADORAS PILOTO. La aleatoriedad de la portadora piloto
    %seleccionada depende de la aleatoriedad del índice utilizado.
k=1;
for w=1:6:Sub_User+2*n
    Espectro(1,w)=pil(1,indice(1,k));
    k=k+1;
end;
Espectro_pil=Espectro; %%%% Cuidado, hasta este momento el vector
espectro esta formado por solo las pilotos


    %se rellenan los demás elementos de Espectro con los simbolos
complejos de
```

```
    %transformado o precodificados, contenidos en Y. El llenado se
efectúa
    %evitando las posiciones de las portadoras piloto.
k=1;
for g=2:6:Sub_User+2*n
    Espectro(1,g:g+4)=Symbolo(1,k:k+4);
    k=k+5;
end;

% ...y se grafica

figure(4)
subplot(2,1,1)
stem(f1,real(Espectro),'.')
xlim([-15*10^3 (Sub_User+2*n)*15*10^3])
ylim([min(real(Espectro))-1 max(real(Espectro))+1])
xlabel('f')
title('Parte Real símbolos con pilotos')
subplot(2,1,2)
stem(f1,imag(Espectro),'r','.')
xlim([-15*10^3 (Sub_User+2*n)*15*10^3])
ylim([min(imag(Espectro))-1 max(imag(Espectro))+1])
xlabel('f')
title('Parte Imaginaria símbolos con pilotos')

%Los simbolos precodificados se presentan a la entrada del
dispositivo de
%mapeo de las subportadoras.
%AAE_Mapeo_Sub(Bit,Espectro,NSy,Espectro_pil)

%////////////////////////////////////////////////////////////////////
////
%////////////////////////////////////////////////////////////////////
////
%////////////////////////////////////////////////////////////////////
////
%/////////RAFAGA SIN METODO DE
CORRECCION//////////////////////////////
%////////////////////////////////////////////////////////////////////
////
%////////////////////////////////////////////////////////////////////
////

Sub_User=500;
n=50;
NPFFT=Sub_User;
f=(0:NPFFT-1)*15*10^3;
f1=(0:(Sub_User+2*n)-1)*15*10^3;
n=50;

%Se establece un factor de peso para las portadoras piloto segun el
%sistema de modulacion escogido
% switch NSy
```

```
%     case 4
%            % Ypil=1.2;
%             Ypil=5.0;
%     case 8
%             Ypil=2*1.2;
%     case 16
%             Ypil=2*1.2;
% end;
    % Se efectua la FFT de y
%Y=(1/NPFFT)*fft(Symbolo,NPFFT);

    %...se agregan las portadoras piloto

%TABLA DE PORTADORAS PILOTO
pil(1,1)=(1+1j)*Ypil;
pil(1,2)=(-1+1j)*Ypil;
pil(1,3)=(-1-1j)*Ypil;
pil(1,4)=(1-1j)*Ypil;

% pil(1,1)=(1+1j)*Ypil;
% pil(1,2)=(1+1j)*Ypil;
% pil(1,3)=(1+1j)*Ypil;
% pil(1,4)=(1+1j)*Ypil;
% pil(1,5)=(1+1j)*Ypil;
% pil(1,6)=(-1+1j)*Ypil;
% pil(1,7)=(-1-1j)*Ypil;
% pil(1,8)=(1-1j)*Ypil;
% pil(1,9)=(1+1j)*Ypil;
% pil(1,10)=(-1+1j)*Ypil;
% pil(1,11)=(-1-1j)*Ypil;
% pil(1,12)=(1-1j)*Ypil;
% pil(1,13)=(1+1j)*Ypil;
% pil(1,14)=(-1+1j)*Ypil;
% pil(1,15)=(-1-1j)*Ypil;
% pil(1,16)=(1-1j)*Ypil;

Espectro2=zeros(1,12*n); %porque cada recurso posee 12 portadoras
    %Se genera aleatoriamente el índice del vector pil
%indice=randi([1 2*n],1,2*n);
%indice=randi([1 4],1,2*n);
indice=round(1+rand(1,2*n)*3);
    %Se ubican las portadoras piloto de 6 en 6 posiciones en el
vector Espectro
    %que tiene una longitud 2*n mayor que los vectores Symbolo y Y
con las
    %subportadoras de información y transformados respectivamente.
    %En cada paso k se ubica una portadora seleccionada al azar de
la TABLA
    %DE PORTADORAS PILOTO. La aleatoriedad de la portadora piloto
    %seleccionada depende de la aleatoriedad del índice utilizado.
k=1;
for w=1:6:Sub_User+2*n
    Espectro2(1,w)=pil(1,indice(1,k));
    k=k+1;
```

```
end;
Espectro_pil2=Espectro2; %%%% Cuidado, hasta este momento el vector
espectro esta formado por solo las pilotos


    %se rellenan los demás elementos de Espectro con los simbolos
complejos de
    %transformado o precodificados, contenidos en Y. El llenado se
efectúa
    %evitando las posiciones de las portadoras piloto.
k=1;
for g=2:6:Sub_User+2*n
    Espectro2(1,g:g+4)=Symbolo2(1,k:k+4);
    k=k+5;
end;

% ...y se grafica

figure(5)
subplot(2,1,1)
stem(f1,real(Espectro2),'g','.')
xlim([-15*10^3 (Sub_User+2*n)*15*10^3])
ylim([min(real(Espectro2))-1 max(real(Espectro2))+1])
xlabel('f')
title('Parte Real símbolos sin channel coding con pilotos')
subplot(2,1,2)
stem(f1,imag(Espectro2),'y','.')
xlim([-15*10^3 (Sub_User+2*n)*15*10^3])
ylim([min(imag(Espectro2))-1 max(imag(Espectro2))+1])
xlabel('f')
title('Parte Imaginaria símbolos sin channel coding con pilotos')


end



function [Espectro_ofdm_conDC,Espectro_ofdm_conDC2] =
AAE_Mapeo_Sub(Bit,Espectro,Espectro2,NSy,Espectro_pil,Espectro_pil2)
%
% %Por el canal de bajada se envía al usuario cuáles bloques de
recursos
% %va a utilizar en ese particular "slot". En nuestra simulación los
bloques
% %se asignarán al azar, al igual que se ha asignado al azar el
número n de los
% %mismos.

 n=50;
 f=(0:1023)*15000;
```

```
Espectro_ofdm_conDC=[zeros(1,211) Espectro(1:300) 0
Espectro(301:600) zeros(1,212)];
Espectro_ofdm_conDC2=[zeros(1,211) Espectro2(1:300) 0
Espectro2(301:600) zeros(1,212)];

%Se procede a graficar.
figure(6)
subplot(2,1,1)
stem(f,real(Espectro_ofdm_conDC),'.')
% xlim([-1 NPIFFT+1])
% ylim([-1.25 1.25])
xlabel('f')
title('Parte Real FFT despues del mapeo')
subplot(2,1,2)
stem(f,imag(Espectro_ofdm_conDC),'r','.')
% xlim([-1 NPIFFT+1])
% ylim([-1.25 1.25])
xlabel('f')
title('Parte Imaginaria FFT despues del mapeo')

figure(7)
subplot(2,1,1)
stem(f,real(Espectro_ofdm_conDC2),'g','.')
% xlim([-1 NPIFFT+1])
% ylim([-1.25 1.25])
xlabel('f')
title('Parte Real FFT sin channel coding despues del mapeo')
subplot(2,1,2)
stem(f,imag(Espectro_ofdm_conDC2),'y','.')
% xlim([-1 NPIFFT+1])
% ylim([-1.25 1.25])
xlabel('f')
title('Parte Imaginaria FFT sin channel coding despues del mapeo')



%AAF_Transformada_Inversa(Bit,Espectro_ofdm_conDC,NSy,Espectro_pil)
end



function [s_ofdm,s_ofdm2] =
AAF_Transformada_Inversa(Bit,Espectro_ofdm_conDC,Espectro_ofdm_conDC
2,NSy,Espectro_pil)

Tofdm=66.667*10^(-6);
NPIFFT=1024;
s_ofdm=NPIFFT*ifft(Espectro_ofdm_conDC,NPIFFT);  %Se calcula la ifft
del vector
                                                 %s_ofdm es el símbolo
OFDM en el dominio
                                                 %del tiempo
```

```
s_ofdm2=NPIFFT*ifft(Espectro_ofdm_conDC2,NPIFFT);  %Se calcula la
ifft del vector
                                                   %s_ofdm es el símbolo
OFDM en el dominio
                                                   %del tiempo

t=0:Tofdm/NPIFFT:Tofdm-(Tofdm/NPIFFT);

figure (8)                          %Gráficos de la parte real e
imaginaria
subplot(2,1,1)                      %del símbolo OFDM
plot(t,real(s_ofdm))
xlim([0 Tofdm-(Tofdm/NPIFFT)])
xlabel('t')
title('Parte Real del símbolo OFDM')
subplot(2,1,2)
plot(t,imag(s_ofdm),'r')
xlim([0 Tofdm-(Tofdm/NPIFFT)])
xlabel('t')
title('Parte Imaginaria del símbolo OFDM')

figure (8)                          %Gráficos de la parte real e
imaginaria
subplot(2,1,1)                      %del símbolo OFDM
plot(t,real(s_ofdm))
xlim([0 Tofdm-(Tofdm/NPIFFT)])
xlabel('t')
title('Parte Real del símbolo OFDM')
subplot(2,1,2)
plot(t,imag(s_ofdm),'r')
xlim([0 Tofdm-(Tofdm/NPIFFT)])
xlabel('t')
title('Parte Imaginaria del símbolo OFDM')


figure (9)                          %Gráficos de la parte real e
imaginaria
subplot(2,1,1)                      %del símbolo OFDM
plot(t,real(s_ofdm2),'g')
xlim([0 Tofdm-(Tofdm/NPIFFT)])
xlabel('t')
title('Parte Real del símbolo OFDM sin channel coding')
subplot(2,1,2)
plot(t,imag(s_ofdm2),'y')
xlim([0 Tofdm-(Tofdm/NPIFFT)])
xlabel('t')
title('Parte Imaginaria del símbolo OFDM sin channel coding')

%AAG_Agrega_PC(Bit,s_ofdm,NSy,Espectro_pil)

end
```

```
function [s_OFDM_analog,s_OFDM_analog2,NN,t_ofdm_PC] =
AAG_Agrega_PC(Bit,s_ofdm,s_ofdm2,NSy,Espectro_pil)

Tofdm=66.667*10^(-6);
t_PC=4.6875*10^(-6);
t_ofdm_PC=Tofdm+t_PC;
NPIFFT=1024;

longPC=9;           %Se agrega el prefijo cíclico de 9 puntos
(4,6875 useg)
NPRF=NPIFFT+longPC;
s_ofdm_PC=zeros(NPRF,1);
PC(1:longPC)=s_ofdm(NPIFFT-8:NPIFFT);
s_ofdm_PC(1:longPC)=PC;
s_ofdm_PC(longPC+1:NPRF)=s_ofdm;

t=0:t_ofdm_PC/NPRF:t_ofdm_PC-(t_ofdm_PC/NPRF);


figure (10)                         %Gráficos de la parte real e
imaginaria
subplot(2,1,1)                     %del símbolo OFDM con el Prefijo
Cíclico PC
plot(t,real(s_ofdm_PC))
xlim([0 t_ofdm_PC-(t_ofdm_PC/NPRF)])
xlabel('t')
title('Parte Real del símbolo OFDM con Prefijo Cíclico')
subplot(2,1,2)
plot(t,imag(s_ofdm_PC),'r')
xlim([0 t_ofdm_PC-(t_ofdm_PC/NPRF)])
xlabel('t')
title('Parte Imaginaria del símbolo OFDM con Prefijo Cíclico')

%sobremuestreo de la señal
%dtm=5*10^(-7); %periodo del sobremuestreo
dtm=t_ofdm_PC/1033/1;
NN=t_ofdm_PC/dtm; % numero de muestras de la señal sobremuestreada
tn=0:dtm:t_ofdm_PC-dtm; %nuevo eje sobremuestreado

s_OFDM_analog=spline(t,s_ofdm_PC,tn);

%AAH_Canal(Bit,s_OFDM_analog,NN,t_ofdm_PC,NSy,Espectro_pil)

%////////////////////////////////////////////////////////////////////
///
%////////////////////////////////////////////////////////////////////
///
%////////////////////BITS SIN CHANNEL
CODING/////////////////////////
%////////////////////////////////////////////////////////////////////
///
%////////////////////////////////////////////////////////////////////
///
```

```
longPC=9;           %Se agrega el prefijo cíclico de 9 puntos
(4,6875 useg)
NPRF=NPIFFT+longPC;
s_ofdm_PC2=zeros(NPRF,1);
PC(1:longPC)=s_ofdm(NPIFFT-8:NPIFFT);
s_ofdm_PC2(1:longPC)=PC;
s_ofdm_PC2(longPC+1:NPRF)=s_ofdm2;

t=0:t_ofdm_PC/NPRF:t_ofdm_PC-(t_ofdm_PC/NPRF);


figure (11)                         %Gráficos de la parte real e
imaginaria
subplot(2,1,1)                     %del símbolo OFDM con el Prefijo
Cíclico PC
plot(t,real(s_ofdm_PC2),'g')
xlim([0 t_ofdm_PC-(t_ofdm_PC/NPRF)])
xlabel('t')
title('Parte Real del símbolo OFDM sin channel coding con Prefijo
Cíclico')
subplot(2,1,2)
plot(t,imag(s_ofdm_PC2),'y')
xlim([0 t_ofdm_PC-(t_ofdm_PC/NPRF)])
xlabel('t')
title('Parte Imaginaria del símbolo OFDM sin channel coding con
Prefijo Cíclico')

%sobremuestreo de la señal
%dtm=5*10^(-7); %periodo del sobremuestreo
dtm=t_ofdm_PC/1033/1;
NN=t_ofdm_PC/dtm; % numero de muestras de la señal sobremuestreada
tn=0:dtm:t_ofdm_PC-dtm; %nuevo eje sobremuestreado

s_OFDM_analog2=spline(t,s_ofdm_PC2,tn);


end



function
[senalmulti,Rsenalmulti,senalmulti2,Rsenalmulti2,NP]=AAH_Canal(Bit,s
_OFDM_analog,s_OFDM_analog2,NN,t_ofdm_PC,NSy,Espectro_pil)

%Número de muestras
NP=NN;   %esta tomando en cuenta las 9 muestras que agrega el
prefijo ciclico y el sobremuestreo
%Período de muestreo
Dt=t_ofdm_PC/NP;
%Delta paa respuesta impulsiva
deltadi=[1,zeros(1,NP-1)];
```

```
%eje de tiempo
t=(0:NP-1)*Dt;
snr=3;

tipo=input('Indique el tipo de canal (1 al 4) ') ; % con esta
instruccion el usuario decide el tipo de canal
[respimpulsivat,senalmulti]=eligecanal(Dt,deltadi,s_OFDM_analog,tipo
);
Rsenalmulti = awgn(senalmulti,snr,'measured');  %Señal con ruido que
paso por el canal en tiempo

%Respuesta del canal Raleigh ruidoso en el dominio de la
%frecuencia
%NpFFT=2^nextpow2(NP);
NpFFT=NP;
f=(0:NpFFT-1)*(1/(t_ofdm_PC));


respimpulsivafrec=fft(respimpulsivat,NpFFT)*(1/NpFFT); %Respuesta en
frecuencia canal
FRsenalmulti=fft(Rsenalmulti,NpFFT)*(1/NpFFT); %Señal con ruido que
paso por el canal en frecuencia
Fsenalmulti=fft(senalmulti,NpFFT)*(1/NpFFT);    %Señal sin ruido que
paso por el canal en frecuencia

%grafica respuesta al impulso del canal
figure (12)

stem (t,abs(respimpulsivat));
title('Respuesta impulsiva del canal')

figure (13)

plot (t,abs(senalmulti));
hold on
plot (t,abs(Rsenalmulti),'r');
title('Gráfica respuesta en tiempo señal que pasó a través del
canal')

%graficas en el dominio de la frecuencia

figure (14)
plot (f,abs(respimpulsivafrec));
title('Gráfica respuesta de amplitud del canal')

figure (15)
plot (f,angle(respimpulsivafrec));
title('Gráfica respuesta de fase del canal')

figure (16)

plot (f,abs(Fsenalmulti));
```

```
hold on
%plot (f,abs(Fsenalmulti_ruido),'r');
title('Gráfica respuesta de amplitud en frecuencia de la señal al
pasar por el canal')
%gráficas en tiempo


%AAI_Elimina_PC(Bit,senalmulti,Rsenalmulti,NSy,Espectro_pil,NP)

%////////////////////////////////////////////////////////////////////
//////
%////////////////////////////////////////////////////////////////////
//////
%//////////////////////////////BITS SIN CHANNEL
CODING//////////////////////
%////////////////////////////////////////////////////////////////////
//////
%////////////////////////////////////////////////////////////////////
//////

[respimpulsivat2,senalmulti2]=eligecanal(Dt,deltadi,s_OFDM_analog2,t
ipo);
Rsenalmulti2 = awgn(senalmulti2,snr,'measured');  %Señal con ruido
que paso por el canal en tiempo

%Respuesta del canal Raleigh ruidoso en el dominio de la
%frecuencia
%NpFFT=2^nextpow2(NP);
NpFFT=NP;
f=(0:NpFFT-1)*(1/(t_ofdm_PC));


respimpulsivafrec2=fft(respimpulsivat2,NpFFT)*(1/NpFFT); %Respuesta
en frecuencia canal
FRsenalmulti2=fft(Rsenalmulti2,NpFFT)*(1/NpFFT); %Señal con ruido
que paso por el canal en frecuencia
Fsenalmulti2=fft(senalmulti2,NpFFT)*(1/NpFFT);    %Señal sin ruido
que paso por el canal en frecuencia

%grafica respuesta al impulso del canal
figure (17)

stem (t,abs(respimpulsivat2),'g');
title('Respuesta impulsiva del canal a los bits sin channel coding')

figure (18)

plot (t,abs(senalmulti2));
hold on
plot (t,abs(Rsenalmulti2),'y');
title('Gráfica respuesta en tiempo señal sin channel coding que pasó
a través del canal')
```

```
%graficas en el dominio de la frecuencia

figure (19)
plot (f,abs(respimpulsivafrec2),'g');
title('Gráfica respuesta de amplitud del canal a los bits sin
channel coding')

figure (20)
plot (f,angle(respimpulsivafrec2),'g');
title('Gráfica respuesta de fase del canal a los bits sin channel
coding')

figure (21)

plot (f,abs(Fsenalmulti2),'g');
hold on
%plot (f,abs(Fsenalmulti_ruido),'r');
title('Gráfica respuesta de amplitud en frecuencia de la señal sin
channel coding al pasar por el canal')
%gráficas en tiempo

end



function [RSisPC,SisPC,RSisPC2,SisPC2]
=AAI_Elimina_PC(Bit,senalmulti,Rsenalmulti,senalmulti2,Rsenalmulti2,
NSy,Espectro_pil)
Tofdm=66.667*10^(-6);
t_PC=4.6875*10^(-6);
t_ofdm_PC=Tofdm+t_PC;
deltat=t_ofdm_PC/1033;
NPRF=1033;
% paso de submuestreo o conversion analogica a digital

dtm=5*10^(-8); %periodo del sobremuestreo
NN=t_ofdm_PC/dtm; % numero de muestras de la señal sobremuestreada
tn=0:dtm:t_ofdm_PC-dtm; %nuevo eje sobremuestreado
C=round(deltat/dtm);

senalmulti=decimate(senalmulti,C);
Rsenalmulti=downsample(Rsenalmulti,C);

%eliminación del prefijo cíclico
longPC=9;
NPFFT=1024;  %realmente son 1033 -9 puntos para long PC

RSisPC(1:NPFFT)=Rsenalmulti(longPC+1:NPRF);
SisPC(1:NPFFT)=senalmulti(longPC+1:NPRF);
```

```
%////////////////////////////////////////////////////////////////////
/////
%//////////////////////BITS SIN CHANNEL
CODING//////////////////////////
%////////////////////////////////////////////////////////////////////
/////

senalmulti2=decimate(senalmulti2,C);
Rsenalmulti2=downsample(Rsenalmulti2,C);

%eliminación del prefijo cíclico
longPC=9;
NPFFT=1024;  %realmente son 1033 -9 puntos para long PC

RSisPC2(1:NPFFT)=Rsenalmulti2(longPC+1:NPRF);
SisPC2(1:NPFFT)=senalmulti2(longPC+1:NPRF);

%AAJ_Transformada_Directa(Bit,senalmulti,Rsenalmulti,NSy,Espectro_pi
l,RSisPC,SisPC)
end

function
[CROrig,SROrig,SR_Pilot,CR_Pilot,CROrig2,SROrig2,SR_Pilot2,CR_Pilot2
] =
AAJ_Transformada_Directa(Bit,senalmulti,Rsenalmulti,NSy,Espectro_pil
,RSisPC,SisPC,RSisPC2,SisPC2)



NPFFT=1024;
Ensamble=600;
subSR=SisPC;    %%%%Son complejos
subCR=RSisPC;

wSR=(1/NPFFT)*fft(subSR);       %Trasformada de Fourier de la señal
compleja
wCR=(1/NPFFT)*fft(subCR);       %para recuperar los símbolos m_arios
CODIFICADOS.
f1=(0:Ensamble-1)*15000;

% Demapeo de subportadoras

% con ruido
CROrig=[wCR(212:511) wCR(513:812)];
% sin ruido
SROrig=[wSR(212:511) wSR(513:812)];



figure(22)

subplot(2,1,1)
stem(f1,real(SROrig),'.')
```

```
% xlim([-1 NPFFT+1])
% ylim([-1.25 1.25])
xlabel('f')
title('Parte Real de los símbolos codificados sin ruido')
subplot(2,1,2)
stem(f1,imag(SROrig),'.','r')
% xlim([-1 NPFFT+1])
% ylim([-1.25 1.25])
xlabel('f')
title('Parte Imaginaria de los símbolos codificados sin ruido')

figure(23)

subplot(2,1,1)
stem(f1,real(CROrig),'.')
% xlim([-1 NPFFT+1])
% ylim([-1.25 1.25])
xlabel('f')
title('Parte Real de los símbolos codificados con ruido')
subplot(2,1,2)
stem(f1,imag(CROrig),'.','r')
% xlim([-1 NPFFT+1])
% ylim([-1.25 1.25])
xlabel('f')
title('Parte Imaginaria de los símbolos codificados con ruido')

% Almacenamiento en "Pilot" de las portadoras piloto
SR_Pilot=zeros(1,Ensamble);
CR_Pilot=zeros(1,Ensamble);


c=1:6:Ensamble;
SR_Pilot([1 c])=SROrig([1 c]); % sin ruido
CR_Pilot([1 c])=CROrig([1 c]);  % con ruido


figure(24)                  %Gráfica de las subportadoras piloto
recibidas
subplot(2,1,1)
stem(f1,real(SR_Pilot),'.')
%xlim([-1 Ensamble+1])
xlabel('f')
title('Parte Real de las subportadoras piloto sin ruido recibidas')
subplot(2,1,2)
stem(f1,imag(SR_Pilot),'.','r')
%xlim([-1 Ensamble+1])
xlabel('f')
title('Parte Imaginaria de las subportadoras piloto sin ruido
recibidas')

%////////////////////////////////////////////////////////////////
////
```

```
%////////////////////////////////////////////////////////////////////
////
%//////////////////////////BITS SIN CHANNEL
CODING//////////////////////
%////////////////////////////////////////////////////////////////////
////
%////////////////////////////////////////////////////////////////////
////

subSR2=SisPC2;    %%%%Son complejos
subCR2=RSisPC2;

wSR2=(1/NPFFT)*fft(subSR2);       %Trasformada de Fourier de la
señal compleja
wCR2=(1/NPFFT)*fft(subCR2);       %para recuperar los símbolos
m_arios CODIFICADOS.
f1=(0:Ensamble-1)*15000;

% Demapeo de subportadoras

% con ruido
CROrig2=[wCR2(212:511) wCR2(513:812)];
% sin ruido
SROrig2=[wSR2(212:511) wSR2(513:812)];



figure(25)

subplot(2,1,1)
stem(f1,real(SROrig2),'g','.')
% xlim([-1 NPFFT+1])
% ylim([-1.25 1.25])
xlabel('f')
title('Parte Real de los símbolos sin channel coding codificados sin
ruido')
subplot(2,1,2)
stem(f1,imag(SROrig),'.','y')
% xlim([-1 NPFFT+1])
% ylim([-1.25 1.25])
xlabel('f')
title('Parte Imaginaria de los símbolos sin channel coding
codificados sin ruido')

figure(26)

subplot(2,1,1)
stem(f1,real(CROrig2),'.','g')
% xlim([-1 NPFFT+1])
% ylim([-1.25 1.25])
xlabel('f')
```

```
title('Parte Real de los símbolos sin channel coding codificados con
ruido')
subplot(2,1,2)
stem(f1,imag(CROrig2),'.','y')
% xlim([-1 NPFFT+1])
% ylim([-1.25 1.25])
xlabel('f')
title('Parte Imaginaria de los símbolos sin channel coding
codificados con ruido')

% Almacenamiento en "Pilot" de las portadoras piloto
SR_Pilot2=zeros(1,Ensamble);
CR_Pilot2=zeros(1,Ensamble);


c=1:6:Ensamble;
SR_Pilot2([1 c])=SROrig2([1 c]); % sin ruido
CR_Pilot2([1 c])=CROrig2([1 c]);  % con ruido


figure(27)                  %Gráfica de las subportadoras piloto
recibidas
subplot(2,1,1)
stem(f1,real(SR_Pilot2),'.','g')
%xlim([-1 Ensamble+1])
xlabel('f')
title('Parte Real de las subportadoras piloto sin ruido recibidas
(sin channel coding)')
subplot(2,1,2)
stem(f1,imag(SR_Pilot2),'.','y')
%xlim([-1 Ensamble+1])
xlabel('f')
title('Parte Imaginaria de las subportadoras piloto sin ruido
recibidas (sin channel coding)')



%AAK_Extrae_Pilotos(Bit,CROrig,SROrig,SR_Pilot,CR_Pilot,NSy,Espectro
_pil);

end



function [Pilotos,RPilotos,TxPilotos,Pilotos2,RPilotos2,TxPilotos2]
=AAK_Extrae_Pilotos(Bit,CROrig,SROrig,SR_Pilot,CR_Pilot,NSy,Espectro
_pil,CROrig2,SROrig2,SR_Pilot2,CR_Pilot2,Espectro_pil2)

Ensamble=600;
n=50;
CRSimbolos_Recibidos_Cod=zeros(1,Ensamble-2*n);
SRSimbolos_Recibidos_Cod=zeros(1,Ensamble-2*n);
```

```
t=2;
for c=1:5:Ensamble-2*n;
    CRSimbolos_Recibidos_Cod(1,c:c+4)=CROrig(1,t:t+4);
    SRSimbolos_Recibidos_Cod(1,c:c+4)=SROrig(1,t:t+4);

    t=t+6;
end;

% Pilotos sin ceros

Pilotos=[];
RPilotos=[];

for k=1:6:600
   Pilotos=[Pilotos SR_Pilot(k)];  %vector de pilotos recibidos sin
ruido
   RPilotos=[RPilotos CR_Pilot(k)];   %vector de pilotos recibidos
con ruido
end

TxPilotos=[];  %vector de pilotos transmitidos
for k=1:6:600
   TxPilotos=[TxPilotos Espectro_pil(k)];
end


f1=(212:6:810)*15000;




%figure(16)                  %Gráfica de las pilotos

%plot(f1,abs(Pilotos),'r')

%////////////////////////////////////////////////////////////////////
//////
%//////////////////////BITS SIN CHANNEL
CODING//////////////////////////
%////////////////////////////////////////////////////////////////////
//////

CRSimbolos_Recibidos_Cod2=zeros(1,Ensamble-2*n);
SRSimbolos_Recibidos_Cod2=zeros(1,Ensamble-2*n);

t=2;
for c=1:5:Ensamble-2*n;
    CRSimbolos_Recibidos_Cod2(1,c:c+4)=CROrig2(1,t:t+4);
    SRSimbolos_Recibidos_Cod2(1,c:c+4)=SROrig2(1,t:t+4);
```

```
    t=t+6;
end;

% Pilotos sin ceros

Pilotos2=[];
RPilotos2=[];

for k=1:6:600
   Pilotos2=[Pilotos2 SR_Pilot2(k)];  %vector de pilotos recibidos
sin ruido
   RPilotos2=[RPilotos CR_Pilot2(k)];   %vector de pilotos recibidos
con ruido
end

TxPilotos2=[];  %vector de pilotos transmitidos
for k=1:6:600
   TxPilotos2=[TxPilotos2 Espectro_pil2(k)];
end


f2=(212:6:810)*15000;


%AAL_Ecualizador(Pilotos,RPilotos,TxPilotos,Bit,CROrig,SROrig,SR_Pil
ot,CR_Pilot,NSy,Espectro_pil)

end









function [simbolos_sinecua,simbolos_sinecua2]=
AAL_Ecualizador(Pilotos,RPilotos,TxPilotos,Bit,CROrig,SROrig,SR_Pilo
t,CR_Pilot,NSy,Espectro_pil,CROrig2)

Ensamble=600;
n=50;
f1=(212:6:810)*15000;

%[canalinterp,Rcanalinterp,Hinterp4,Hinterp5,Hinterp6,Hinterp7,Hinte
rp8,Hinterp9,Hinterp10,Hinterp11]...
   % =AAOestimadorLS(Pilotos,RPilotos,TxPilotos);
```

```
%Ecualizacion

Ensamble_sinecualizar=CROrig;              %datos + portadoras sin
ecualizar
% Ensamble_ecualizadoLS=CROrig./canalinterp;  % datos mas potadoras
ecualizado LS
% Ensamble_ecualizado4=CROrig./Hinterp4;   %datos+ portadoras
% Ensamble_ecualizado5=CROrig./Hinterp5;
% Ensamble_ecualizado6=CROrig./Hinterp6;
% Ensamble_ecualizado7=CROrig./Hinterp7;
% Ensamble_ecualizado8=CROrig./Hinterp8;   %datos+ portadoras
% Ensamble_ecualizado9=CROrig./Hinterp9;
% Ensamble_ecualizado10=CROrig./Hinterp10;
% Ensamble_ecualizado11=CROrig./Hinterp11;   %datos+ portadoras


 fint=(212:811)*15000;
% figure(25)
% stem(fint,real(Ensamble_ecualizado9),'.')


%Se extraen los datos ecualizados del ensamble

x=2; % será que se piensa que la primera es una piloto?
for c=1:5:Ensamble-2*n;
    simbolos_sinecua(1,c:c+4)=Ensamble_sinecualizar(1,x:x+4);
%     simbolos_ecua_datosLS(1,c:c+4)=Ensamble_ecualizadoLS(1,x:x+4);
%     simbolos_ecua_datos4(1,c:c+4)=Ensamble_ecualizado4(1,x:x+4);
%     simbolos_ecua_datos5(1,c:c+4)=Ensamble_ecualizado5(1,x:x+4);
%     simbolos_ecua_datos6(1,c:c+4)=Ensamble_ecualizado6(1,x:x+4);
%     simbolos_ecua_datos7(1,c:c+4)=Ensamble_ecualizado7(1,x:x+4);
%     simbolos_ecua_datos8(1,c:c+4)=Ensamble_ecualizado8(1,x:x+4);
%     simbolos_ecua_datos9(1,c:c+4)=Ensamble_ecualizado9(1,x:x+4);
%     simbolos_ecua_datos10(1,c:c+4)=Ensamble_ecualizado10(1,x:x+4);
%     simbolos_ecua_datos11(1,c:c+4)=Ensamble_ecualizado11(1,x:x+4);
%
    fdat(1,c:c+4)=fint(1,x:x+4);  % vector de frecuencias sin
posiciones de portadora
    x=x+6;
end;

%figure(26)
%stem(fdat,real(simbolos_ecua_datos8),'.')
dibujaconstelacion(simbolos_sinecua);
%
dibujaconstelacion(simbolos_sinecua,simbolos_ecua_datosLS,simbolos_e
cua_datos4,simbolos_ecua_datos5,...
%
simbolos_ecua_datos6,simbolos_ecua_datos7,simbolos_ecua_datos8,simbo
los_ecua_datos9,simbolos_ecua_datos10,simbolos_ecua_datos11)
```

```
%////////////////////////////////////////////////////////////////////
///////
%////////////////////////////////////////////////////////////////////
///////
%//////////////////////////////BITS SIN CHANNEL
CODING//////////////////////
%////////////////////////////////////////////////////////////////////
///////
%////////////////////////////////////////////////////////////////////
///////

Ensamble_sinecualizar2=CROrig2;

x=2; % será que se piensa que la primera es una piloto?
for c=1:5:Ensamble-2*n;
    simbolos_sinecua2(1,c:c+4)=Ensamble_sinecualizar2(1,x:x+4);
    fdat(1,c:c+4)=fint(1,x:x+4);  % vector de frecuencias sin
posiciones de portadora
    x=x+6;
end;

dibujaconstelacion(simbolos_sinecua2);

%AAQ_Deteccion(Bit,simbolos_sinecua,NSy);
%
simbolos_ecua_datosLS,simbolos_ecua_datos4,simbolos_ecua_datos5,...
%
simbolos_ecua_datos6,simbolos_ecua_datos7,simbolos_ecua_datos8,simbo
los_ecua_datos9,simbolos_ecua_datos10,simbolos_ecua_datos11,NSy)


end
```

```
function [EkDemT,WkDemT,trellisDemT,x2DemT] =
Demodulator(Ek,simbolos_sinecua,WkRec,trellisRec,simbolos_sinecua2)
hDemQPSK=comm.QPSKDemodulator('BitOutput',true,'DecisionMethod','Log
-likelihood ratio');
hDem8QAM=comm.RectangularQAMDemodulator('ModulationOrder',8,'BitOutp
ut',true,'DecisionMethod','Log-likelihood ratio');
hDem16QAM=comm.RectangularQAMDemodulator('ModulationOrder',16,'BitOu
tput',true,'DecisionMethod','Log-likelihood ratio');

hDemQPSK2=comm.QPSKDemodulator('BitOutput',true,'DecisionMethod','Ha
rd decision');
hDem8QAM2=comm.RectangularQAMDemodulator('ModulationOrder',8,'BitOut
put',true,'DecisionMethod','Hard decision');
hDem16QAM2=comm.RectangularQAMDemodulator('ModulationOrder',16,'BitO
utput',true,'DecisionMethod','Hard decision');
```

```
simbolos_sinecuaT=simbolos_sinecua';
simbolos_sinecuaT2=simbolos_sinecua2';

if length(Ek)==1000
    EkDem=step(hDemQPSK,simbolos_sinecuaT);
    WkDem=step(hDemQPSK,WkRec);
    trellisDem=step(hDemQPSK,trellisRec);
    x2Dem=step(hDemQPSK2,simbolos_sinecuaT2);
elseif length(Ek)==1500
    EkDem=step(hDem8QAM,simbolos_sinecuaT);
    WkDem=step(hDem8QAM,WkRec);
    trellisDem=step(hDem8QAM,trellisRec);
    x2Dem=step(hDem8QAM2,simbolos_sinecuaT2);
else
    EkDem=step(hDem16QAM,simbolos_sinecuaT);
    WkDem=step(hDem16QAM,WkRec);
    trellisDem=step(hDem16QAM,trellisRec);
    x2Dem=step(hDem16QAM2,simbolos_sinecuaT2);
end
EkDemT=EkDem';
WkDemT=WkDem';
trellisDemT=trellisDem';
x2DemT=x2Dem';
end
```

```
function [F] = Inverse_BitSelection(EkDemT,WkDemT)
%[g,e,w,trellis,x_dataCRC]=BitSelection;

if length(EkDemT)==40
    L=192;
    rows=2;
elseif length(EkDemT)==1000
    L=3096;
    rows=32;
elseif length(EkDemT)==1500
    L=4608;
    rows=48;
else
    L=6144;
    rows=64;
end
```

```
rvidx=0;
%rvidx=1;
%rvidx=2;
%rvidx=3;
K0=rows*(2*ceil(L/(8*rows))*rvidx+2);
j=0;
q=WkDemT;
for i=0:length(EkDemT)-1
    q(mod(K0+j,L)+1)=EkDemT(i+1);
    j=j+1;
end
    F=q;

end



function [v1,v2,v3] = Inverse_Buffer(F)
%[g,F,trellis,x_dataCRC]=Inverse_BitSelection;
n=length(F)/3;
vv1=zeros(1,n);
vv2=zeros(1,n);
vv3=zeros(1,n);
for i=0:n-1
    vv1(i+1)=F(i+1);
    vv2(i+1)=F((n+1)+2*i);
    vv3(i+1)=F((n+1)+(2*i+1));
end
v1=vv1;
v2=vv2;
v3=vv3;

end




function [Ds,Dp,Di] = SubBlockDeinterleaver(v1,v2,v3,p)
%[g,v1,v2,v3,trellis,x_dataCRC]=Inverse_Buffer;
if length(v1)==64
    rows=2;
elseif length(v1)==1024
    rows=32;
elseif length(v1)==1536
    rows=48;
else
    rows=64;
end
Ms=reshape(v1,rows,32);
Mp=reshape(v2,rows,32);
perm=[0 16 8 24 4 20 12 28 2 18 10 26 6 22 14 30 1 17 9 25 5 21 13
29 3 19 11 27 7 23 15 31];
mii=zeros(1,length(v1));
for i=0:length(v1)-1
```

```
mii(mod(perm(floor(i/p(1,1))+1)+p(1,2)*(mod(i,p(1,1)))+1,(p(1,1)*p(1
,2)))+1)=v3(i+1);
end
%Mi=reshape(mii,32,rows)';
Dii=mii;
%Se realiza la permutacion inversa que el proceso de
SubBlockInterleaver
MatrizS=zeros(rows,32);
MatrizP=zeros(rows,32);

MatrizS(:,1)=Ms(:,1); MatrizS(:,2)=Ms(:,17); MatrizS(:,3)=Ms(:,9);
MatrizS(:,4)=Ms(:,25); MatrizS(:,5)=Ms(:,5); MatrizS(:,6)=Ms(:,21);
MatrizS(:,7)=Ms(:,13); MatrizS(:,8)=Ms(:,29); MatrizS(:,9)=Ms(:,3);
MatrizS(:,10)=Ms(:,19); MatrizS(:,11)=Ms(:,11);
MatrizS(:,12)=Ms(:,27);
MatrizS(:,13)=Ms(:,7); MatrizS(:,14)=Ms(:,23);
MatrizS(:,15)=Ms(:,15);
MatrizS(:,16)=Ms(:,31); MatrizS(:,17)=Ms(:,2);
MatrizS(:,18)=Ms(:,18);
MatrizS(:,19)=Ms(:,10); MatrizS(:,20)=Ms(:,26);
MatrizS(:,21)=Ms(:,6);
MatrizS(:,22)=Ms(:,22); MatrizS(:,23)=Ms(:,14);
MatrizS(:,24)=Ms(:,30);
MatrizS(:,25)=Ms(:,4); MatrizS(:,26)=Ms(:,20);
MatrizS(:,27)=Ms(:,12);
MatrizS(:,28)=Ms(:,28); MatrizS(:,29)=Ms(:,8);
MatrizS(:,30)=Ms(:,24);
MatrizS(:,31)=Ms(:,16); MatrizS(:,32)=Ms(:,32);

MatrizP(:,1)=Mp(:,1); MatrizP(:,2)=Mp(:,17); MatrizP(:,3)=Mp(:,9);
MatrizP(:,4)=Mp(:,25); MatrizP(:,5)=Mp(:,5); MatrizP(:,6)=Mp(:,21);
MatrizP(:,7)=Mp(:,13); MatrizP(:,8)=Mp(:,29); MatrizP(:,9)=Mp(:,3);
MatrizP(:,10)=Mp(:,19); MatrizP(:,11)=Mp(:,11);
MatrizP(:,12)=Mp(:,27);
MatrizP(:,13)=Mp(:,7); MatrizP(:,14)=Mp(:,23);
MatrizP(:,15)=Mp(:,15);
MatrizP(:,16)=Mp(:,31); MatrizP(:,17)=Mp(:,2);
MatrizP(:,18)=Mp(:,18);
MatrizP(:,19)=Mp(:,10); MatrizP(:,20)=Mp(:,26);
MatrizP(:,21)=Mp(:,6);
MatrizP(:,22)=Mp(:,22); MatrizP(:,23)=Mp(:,14);
MatrizP(:,24)=Mp(:,30);
MatrizP(:,25)=Mp(:,4); MatrizP(:,26)=Mp(:,20);
MatrizP(:,27)=Mp(:,12);
MatrizP(:,28)=Mp(:,28); MatrizP(:,29)=Mp(:,8);
MatrizP(:,30)=Mp(:,24);
MatrizP(:,31)=Mp(:,16); MatrizP(:,32)=Mp(:,32);



d1=reshape(MatrizS',1,rows*32);
d2=reshape(MatrizP',1,rows*32);
```

```
%d3=reshape(Mi',1,rows*32);
d3=Dii;
d11=zeros(1,40);
d22=zeros(1,40);
d33=zeros(1,40);
if length(d3)==64
    for i=1:40
        d11(i)=d1(i);
        d22(i)=d2(i);
        d33(i)=d3(i);
    end
    Ds=d11;
    Dp=d22;
    Di=d33;
else
    Ds=d1;
    Dp=d2;
    Di=d3;
end



end



function [H] = TurboDecoding(x_posicion,Ds,Dp,Di,trellisDemT)
%[g,Ds,Dp,Di,trellis,x_dataCRC]=SubBlockDeinterleaver;
%x_posicion=EntrelazadoTC;
T=length(Ds);
Output=zeros(1,3*T);
k=1;
for i=1:3:(3*T-2)
    Output(i)=Ds(k);
    k=k+1;
end
k=1;
for j=2:3:(3*T-1)
    Output(j)=Dp(k);
    k=k+1;
end
k=1;
for m=3:3:3*T
    Output(m)=Di(k);
    k=k+1;
end
%Es necesario recuperar la terminacion de trellis del TurboCodcaer
DataInput=[Output trellisDemT]'; %trellis];
%DataInput2=DataInput';
hTDec=comm.TurboDecoder('TrellisStructure',poly2trellis(4,...

[13,15],13),'InterleaverIndices',x_posicion,'Algorithm','Max','NumIt
erations',8);
DataDecod=step(hTDec,-DataInput);
```

```
%op=length(DataDecod);
H=DataDecod';
end



function [BER,BER2]= AAM_BER(x_dataCRC,H,x2DemT)
if length(x_dataCRC)==1024
    x2=x_dataCRC(1:length(x_dataCRC)-24);
    xR=H(1:length(H)-24);
elseif length(x_dataCRC)==1536
    x2=x_dataCRC(1:length(x_dataCRC)-36);
    xR=H(1:length(H)-36);
else
    x2=x_dataCRC(1:length(x_dataCRC)-48);
    xR=H(1:length(H)-48);
end
vecBER=xor(x2,xR);
vecBER2=xor(x2,x2DemT);
sumBER=0;
sumBER2=0;
for x=1:length(vecBER)

    sumBER=sumBER+vecBER(x);
end

for x=1:length(vecBER2)

    sumBER2=sumBER2+vecBER2(x);
end

BER=sumBER*100/length(vecBER);
BER2=sumBER2*100/length(vecBER2);
end
```