UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA

FACULTAD DE PRODUCCIÓN Y SERVICIOS

ESCUELA PROFESIONAL DE INGENIERÍA EN TELECOMUNICACIONES

PROYECTO: Modulacion Adaptativa QAM

CURSO:

Redes Inalambricas

PRESENTADO POR:

Araca Chambi Yessica Sebastiana
Gomez Bobadilla Nadia Medalith
Huachaca Jara Xiomara Estephany
Quispe Chise Bryan Mario
Ydme Zea Jhon Paul

DOCENTE:

Dr. Alexander Hilario Tacuri

AREQUIPA - PERU
2024

Índice

Índice de figuras

PROYECTO: Modulacion Adaptativa QAM

1. Introduccion

En este proyecto, se aborda el diseño y desarrollo de un sistema de comunicaciones inalámbrico basado en la modulación adaptativa QAM (Modulación de Amplitud en Cuadratura), utilizando la tecnología SDR BladeRF. El objetivo primordial radica en establecer un sistema capaz de transmitir y recibir imágenes de manera eficiente, empleando diferentes esquemas de modulación, como 4-QAM, 8-QAM y 16-QAM. A través de esta iniciativa, se busca explorar las potencialidades de la modulación adaptativa QAM en el contexto de las comunicaciones inalámbricas, con miras a mejorar la flexibilidad y el rendimiento de los sistemas de transmisión de imágenes.

2. Objetivos

2.1. Objetivo general

- Diseñar un sistema de comunicaciones inalámbrico básico para la transmicion y recepcion de imagenes usando la Modulacion Adaptativa QAM.

2.2. Objetivos especificos

- Desarollar un transmisor para el envio de una imagen con cualquier tamaño y cualquier modulacion(4-QAM, 8-QAM, 16-QAM).
- Desarrollar receptor que decodifique el paquete segun el tamaño y modulacion usada en el transmisor y muestre el resultado.

3. DESARROLLO DEL PROYECTO

3.1. TRANSMISOR

En esta parte del proyecto se implementara un sistema de comunicaciones usando la modulación QAM. El objetivo es enviar una imagen de una computadora hacia otra utilizando el SDR. Los pasos para el diseño del sistema son los siguientes:

3.1.1. GENERACION DEL PREAMBULO

En el siguiente codigo se esta creando un objeto BarkerCode que genera una secuencia de código Barker de longitud 13 con 16 muestras por trama. Las secuencias de código Barker son conjuntos de números que nos permiten la sincronización y la detección de la nuestra señal. Luego se esta generando la secuencia de código Barker para luego repetirla 4 veces para formar el preámbulo de la señal. El preámbulo es una secuencia conocida que se envía antes de los datos reales para ayudar al receptor a sincronizarse con la señal.

```
%%%%%%%%%%%%%%%%%%%%%%%%   Preámbulo   %%%%%%%%%%%%%%%%%%%%%%%%%

barker = comm.BarkerCode(... % For preamble
    'Length',13,'SamplesPerFrame',16)
preamble = barker();
preambul=vertcat(preamble, preamble,preamble,preamble);
```

3.1.2. GENERACION DE LA MODULACION

Aquí generamos la modulacion con una variable M, seguidamente se lee la imagen del archivo $'FUE.jpg'$ y se almacena en la variable $YourImage$. La imagen se almacena como una matriz. En estas líneas, estás usando la función *vertcat* para concatenar verticalmente el vector x

```
%%%%%%%%%%%%%%%%%%   Generacion de la Modulación   %%%%%%%%%%%%%%%%

M=16; %orden de modulacion
YourImage=imread('TIF.jpg'); % Leer imágenes desde archivos
sizeOfImage = size(YourImage); % Tamaño de la imagen
x=YourImage(:); % Conversión vector columna
signal=vertcat(x); % Imagen + tamaño
```

3.1.3. GENERACION DE LA SECUENCIA DE BITS

En la variable de **bits** se esta convirtiendo cada elemento de la señal en un conjunto de 8 bits y luego convirtiendo esos bytes en una secuencia de bits. Vamos a desglosarlo:

- **typecast(signal, 'uint8'):** Esta función convierte la señal en un array de uint8 (números enteros sin signo de 8 bits). Esto se hace porque la función dec2bin requiere números enteros.
- **dec2bin(typecast(signal, 'uint8'), 8):** Esta función convierte cada número entero en una cadena de caracteres que representa su valor binario. El segundo argumento, 8, especifica que queremos una cadena de 8 caracteres, es decir, un byte.

- **dec2bin(...) - '0':**Este truco se utiliza para convertir la cadena de caracteres en un array de números. En MATLAB, restar '0' a una cadena de caracteres que contiene dígitos numéricos da como resultado un array de números.
- **(dec2bin(...) - '0').':** La función de transposición (.') se utiliza para cambiar la orientación del array de números de horizontal a vertical.
- **reshape((dec2bin(...) - '0').', 1, []):**Finalmente, la función reshape se utiliza para convertir el array 2D en un vector fila.

```
1
2 %%%%%%%%%%%%%%%%%%%%%%%%   Secuencia de bits   %%%%%%%%%%%%%%%%%%%%%%
3
4 bits = reshape((dec2bin(typecast(signal, 'uint8'), 8) - '0').', 1,
     []); % Secuencia de bits de la imagen
5 % Longitud de imagen
6 imglength=length(bits);
```

3.1.4. GENERACION DE LA SECUENCIA DE BITS

En la variable **t1,t2,t3** se esta obteniendo las dimensiones de la imagen original. $sizeOfImage(1)$ representa la altura de la imagen, $sizeOfImage(2)$ la anchura y $sizeOfImage(3)$ la cantidad de canales de color (normalmente 3 para las imágenes RGB).

En la variable **tamano 1, tamaño 2, tamaño 3** se estan convirtiendo las dimensiones de la imagen a representaciones binarias de 3 bytes (24 bits). La función $dec2bin$ convierte un número decimal a una cadena de caracteres binaria. El primer argumento es el número decimal que quieres convertir, y el segundo argumento es el número de bits que quieres que tenga la cadena resultante. Por lo tanto, $dec2bin(t1, 3*8)$ convierte la altura de la imagen t1 a una cadena de 24 bits.

```
1 %%%%%%%%%%%%%%%%%%%%%%%%  Dimensiones  %%%%%%%%%%%%%%%%%%%%%%%%%%
2 t1=sizeOfImage(1);
3 t2=sizeOfImage(2);
4 t3=sizeOfImage(3);
5 tamano1=dec2bin(t1,3*8)-'0';
6 tamano2=dec2bin(t2,3*8)-'0';
7 tamano3=dec2bin(t3,3*8)-'0';
8 tfinal=[tamano1';tamano2';tamano3'];
```

3.1.5. ALMACENAMIENTO DE BITS EN 3 BYTES

La explicacion de las siguientes lineas para almacenar los bits se da de la siguiente manera:

- **info=dec2bin(imglength,3*8)-'0';:** Convierte la longitud de la imagen (imglength) en una representación binaria. El segundo argumento (3*8) especifica la longitud mínima de la cadena de bits. '0' al final se usa para convertir la cadena de caracteres binarios en una matriz de números binarios.
- **modulac=dec2bin(M,3*8)-'0';:** Hace lo mismo que la línea anterior, pero para la variable M.
- $senal_digital = [bits'];$: Esta línea está creando una matriz llamada $senal_digital$ que contiene los valores de la matriz bits transpuesta (eso es lo que hace el apóstrofe).

```
%%%%%%%%%%%%%%%  Almacenamos los bits en 3 bytes  %%%%%%%%%%%%%%%%%%%

info=dec2bin(imglength,3*8)-'0'; %Longitud de los bits en 3 bytes
   imagen
modulac=dec2bin(M,3*8)-'0';
%Almacenamos todo en un vector
senal_digital=[bits'];
```

3.1.6. APLICACION DE LA MODULACION

En esta parte del codigo se esta usando la función *qammod* para modular la señal digital. *qammod* es una función de MATLAB que realiza la modulación por amplitud en cuadratura (QAM), que es un tipo de modulación digital. Los argumentos de *qammod* son los siguientes:

- **data:** Esta es la señal que se quiere modular. En este caso, es la señal digital que se preparo antes.
- **M:** Este es el orden de la modulación QAM.
- **InputType='bit':** Este argumento opcional que especifica que la señal de entrada es una secuencia de bits. Si no se especifica, *qammod* asume que la entrada es una secuencia de enteros.
- **UnitAveragePower=true:** Este argumento opcional especifica que se quiere que la señal modulada tenga una potencia promedio de 1. Esto es útil para controlar la potencia de la señal que estás transmitiendo.

Por lo tanto, después de estas líneas de código, txSig será tu señal modulada, lista para ser transmitida.

```

%%%%%%%%%%%%%%%  Aplicacion de la Modulacion %%%%%%%%%%%%%%%%%%%%%%%%

data=senal_digital;
txSig = qammod(data,M, ...
    InputType='bit', ...
    UnitAveragePower=true);
```

3.1.7. CREACIÓN DE 'TRUSH'

1. $datos\_aleatorios = randi([0,1],2000,1)$; Esta línea genera un vector de 2000 números aleatorios que son 0 o 1. Este vector se llama "datos_aleatorios".
2. $informacion = [info'; tfinal; modulac']$; Aquí estás creando un vector llamado ïnformacion"que consiste en las transposiciones de ïnfo", "tfinalz "modulac". La transposición ('), es utilizada para convertir filas en columnas y viceversa.
3. $informacion\_modulada = pskmod(informacion,2)$; Aquí estás utilizando la función "pskmod"para modular la información usando la modulación por desplazamiento de fase (PSK). El número 2 indica que estás utilizando una PSK binaria.
4. $trush = pskmod(datos\_aleatorios,2)$; Similar a la línea anterior, estás modulando los "datos aleatoriosçon una PSK binaria.
5. $scatterplot(txSig)$; Esta línea crea un gráfico de dispersión de "txSig".

```
%%%%%%%%%%%%%%%%%%%%%%%%  Creacion del TRUSH %%%%%%%%%%%%%%%%%%%%%%%%

datos\_aleatorios=randi([0, 1], 2000, 1);
informacion=[info';tfinal;modulac']; %5 segmentos de 24 bits(96 bits)
informacion_modulada=pskmod(informacion,2);
trush=pskmod(datos_aleatorios,2);
scatterplot(txSig);
%datas=[preambul;txSig];
datas=[trush;preambul;informacion_modulada;txSig;trush];
figure
stem(datas)
figure
stem(preambul)
```

3.1.8. UTILIZACION DEL PULSE SHAPING PARA EL FILTRO TRANSMISOR

En esta parte hacemos uso de un filtro de coseno elevado para modelar tus pulsos. La línea de código $txfilter = comm.RaisedCosineTransmitFilter(...)$ crea un filtro de transmisión de coseno elevado. Luego, se aplica este filtro a los datos con la línea $txSignal = txfilter(datas)$;. Esto genera una señal que tiene un ancho de banda y una potencia más eficientes, y minimizar el ISI.

```

%%%%%%%%%%%%%%  Pulse Shaping para el Filtro %%%%%%%%%%%%%%%%%%

span=10;
rolloff=0.5;
sps=4;
R = 1000;
Fs = R * sps;

%%  Filtro transmisor
txfilter=comm.RaisedCosineTransmitFilter(...
    Shape='Square root',...
    RolloffFactor=rolloff,...
    FilterSpanInSymbols=span,...
    OutputSamplesPerSymbol=sps);
figure
impz(txfilter.coeffs.Numerator)
txSignal=txfilter(datas);
scatterplot(txSignal,sps)
```

3.1.9. GUARDAR EL ARCHIVO EN FORMATO $.sc16q11$

En esta línea de código se guarda la señal transmitida (txSignal) en un archivo llamado "FUE.sc16q11.[en] la ruta especificada, luego para una mejor visualizacion se realizan las respectivas graficas.

```

%%%%%%%%%%%  Guardado Del Archivo en formato .sc16q11 %%%%%%%%%%%%

save_sc16q11("C:\Users\ASUS\Documents\MATLAB\BPSKFinal2\imagen\TIF.
    sc16q11",txSignal);
signal=load_sc16q11("C:\Users\ASUS\Documents\MATLAB\BPSKFinal2\imagen
    \TIF.sc16q11");
scatterplot(signal,sps)
figure()
```

```
plot(real(signal))
hold on
plot(imag(signal))
title('Señal x(n) real e imaginaria');
xlabel('n')
grid on

scatterplot(txSignal,sps)
```

3.2. RECEPTOR

3.2.1. Recepción con el BladeRF

```
C:\Users\estep>bladeRF-cli -i
bladeRF> set frequency rx 850M

  RX1 Frequency:  849999998 Hz (Range: [70000000, 6000000000])

bladeRF> set bandwidth rx 4M

  RX1 Bandwidth:   4000000 Hz (Range: [200000, 56000000])

bladeRF> set samplerate rx 8M

  Setting RX1 sample rate - req:   8000000 0/1Hz, actual:   8000000 0/1Hz

bladeRF> set agc off

  RX1 AGC: Disabled
  RX2 AGC: Disabled

bladeRF> set gain rx  30

Note: This change will not be visible until the channel is enabled.
Setting RX1 overall gain to 30 dB
  Gain RX1 overall:   60 dB (Range: [-16, 60])
              full:   77 dB (Range: [1, 77])

bladeRF> rx config file=C:\Users\estep\OneDrive\Escritorio\MATLAB_CODES\R_16.sc16q11 f
bladeRF> rx start
bladeRF> rx

  State: Running
  Channels: RX1
  Last error: None
  File: C:\Users\estep\OneDrive\Escritorio\MATLAB_CODES\R_16.sc16q11
  File format: SC16 Q11, Binary
  # Samples: infinite
  # Buffers: 32
  # Samples per buffer: 32768
  # Transfers: 16
  Timeout (ms): 1000

bladeRF> rx stop
```

3.2.2. Sincronización de frecuencia

Es necesaria implementarla para compensar las desviaciones de frecuencia entre el transmisor y el receptor, esta compensación es necesaria porque en una señal modulada, las diferencias de frecuencia pueden causar un desplazamiento espectral, lo que dificulta la demodulación y la recuperación de los datos. Con esta sincronización se puede corregir diferencias en la frecuencia portadora ocasionados por cambios en la frecuencia del canal.

```
%% SINCRONIZACIÓN DE FRECUENCIA
compensacion=comm.CoarseFrequencyCompensator( ...
```

```
Modulation='QAM', ...
SampleRate=Fs, ...
FrequencyResolution=11);                    % Compensación de
   frecuencia 0.0001
[compensado,estimate]=compensacion(txSig);
freqCompInfo = info(compensacion)           % Información del
   compensado
disp("La Compensación en Frecuencia es de: " )
disp(estimate)
figure
plot(real(compensado))
hold on
plot(imag(compensado))
scatterplot(compensado,4)
```

3.2.3. Sincronización de tiempo

Es necesaria implementarla para para sincronizar en el tiempo los símbolos recibidos con el reloj del receptor. En modulaciones como QAM, los símbolos deben ser muestreados en el momento adecuado para evitar la interferencia entre símbolos adyacentes garantizando que el receptor pueda distinguir claramente los límites de los símbolos y decodificarlos correctamente.

```
%% SINCRONIZACIÓN DE TIEMPO
timingSync=comm.SymbolSynchronizer(...      % Función
   SymbolSynchronizer
Modulation="PAM/PSK/QAM",...
SamplesPerSymbol=2,...
NormalizedLoopBandwidth=0.001...
,...
DampingFactor=10,...
TimingErrorDetector="Mueller-Muller (decision-directed)"...
);
syncSimbolos=timingSync(simbolos);           % Corregir Retardo
scatterplot(syncSimbolos)
```

3.2.4. Sincronización de fase

Es necesaria implementarla para para sincronizar la fase de la portadora recibida con la fase de referencia en el receptor. En QAM, la fase de la portadora es crucial para distinguir entre los diferentes símbolos modulados en fase garantizando el receptor pueda demodular correctamente los símbolos y recuperar los datos transmitidos.

```
%% SINCRONIZACIÓN DE FASE
carrierSync=comm.CarrierSynchronizer(...      % Función
   CarrierSynchronizer
SamplesPerSymbol=1,...
ModulationPhaseOffset="Auto",...
NormalizedLoopBandwidth=0.0031 ...
,...
Modulation='QAM',...
DampingFactor=0.8 ...
);
CarriersyncSimbolos=carrierSync(syncSimbolos); % Corregir en fase
scatterplot(CarriersyncSimbolos,1)
```

Con este y los pasos anteiores ya deberíamos tener una constelación semejante a 4qam, 8qam o 16qam, las implementadas en este proyecto.

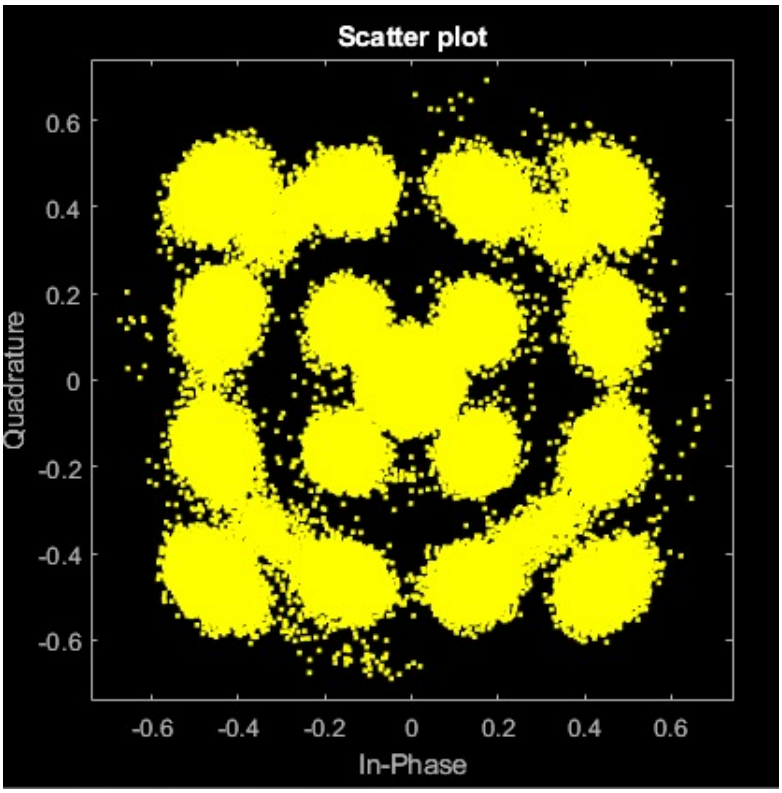

Figura 1: Sincronización de señal 16QAM

3.2.5. Detección del preámbulo

Al recepcionar la señal por la gran cantidad de muestras no podemos identificar el punto de inicio de nuestra señal, lo que necesitamos para poder seleccionar los simbolos y bits que representan a la imagen, es por esto que se transmite un preámbulo, una secuencia de bits conocida por transmisor y el receptor la ubicación de la señal en tiempo.
Como vimos en el transmisor hemos implementado la función BarkerCode para obtener estos simbolos conocidos, en el receptor nuevamente la implementaremos y para ubicar este código utilizaremos la función *comm.PreambleDetector*. En esta función ingresamos el valor conocido, el tipo de dato que es y el threshold.

```
%% DETECCIÓN DEL PREÁMBULO
barker = comm.BarkerCode(...
    'Length',13,'SamplesPerFrame',16)
preamble = barker();
preambul=[preamble;preamble;preamble;preamble];
thr = 0.89*sum(abs(preambul).^2);
demo=pskdemod(CarriersyncSimbolos,2);      % Demodulacion en BPSK de
    la señal sincronizaca
mo=pskmod(demo,2);                         % Modulacion en BPSK
prbdet= comm.PreambleDetector(Preamble=preambul,Detections='All',
    Input='Symbol',Threshold=thr);
[idx,detmet] = prbdet(mo);                 % Detectamos el indice de
    la señal
```

3.2.6. Extraer la información de la imagen transmitida

Después de sincronizar y ubicar la señal recibida, hay que demodular la señal con la función *bpskdemod*, porque los bits de información fueron modulados en bpsk.

Después necesitamos saber si la modulación de la imagen es 4qam, 8qam o 16qam, para la identificación del tipo de modulación se revisa los bits de información, los que se transmitieron antes de los bits de imagen y que representan a las dimensiones de la imagen y al número de modulación, este ultimo valor será guardado como la variable M y las dimensiones serán guardadas en 3 variables, t1, t2 y t3. Estos bits fueron seleccionados considerando la ubicación del ultimo valor del preambulo y tomando en cuenta que cada bloque de información esta compuesto por 3 bytes, por lo que se seleccionaron de 24 en 24 bits.

```
%% EXTRAER TAMAÑOS DE LA IMAGEN
inicio=idx-63;                                % Establecemos el inicio
   del preámbulo
inicio_info=idx+1;                            % Inicio de los bits de
   información
info_symb=CarriersyncSimbolos(inicio_info:inicio_info+119); %
   Extraemos el numero de muestras que contiene la información
info_bits=pskdemod(info_symb,2);              % Demodulamos el preambulo
    e información
leng_bits=info_bits(1:24);                    % Establecemos la longitud
    de bits de la imagen
%% EXTRAER LAS DIMENSIONES DE LA IMAGEN RECEPCIONADA
bits_t1=info_bits(25:48);                    % Extraemos las dimensiones
    de la imagen recepcionada
bits_t2=info_bits(49:72);
bits_t3=info_bits(73:96);
modulac=info_bits(97:120);                    % Extraemos el orden de
   modulacion 4-QAM, 8-QAM Y 16-QAM
long_imag=bin2dec(char(leng_bits'+'0'));     % Realizamos la conversión
    con bin2dec, e imprimimos los valores
t1=bin2dec(char(bits_t1'+'0'));
t2=bin2dec(char(bits_t2'+'0'));
t3=bin2dec(char(bits_t3'+'0'));
M=bin2dec(char(modulac'+'0'));               % Realizamos la conversión
   con bin2dec, e imprimimos los el orden de modulación
data_imag=CarriersyncSimbolos(inicio_info+120:end); % Extraemos el
   paquete de información de la señal
scatterplot(data_imag)
```

3.2.7. Demodulación QAM

```
bits=qamdemod(data_imag,M,OutputType="bit", UnitAveragePower=false);
   %
scatterplot(data_imag)
imag_symb=bits(1:long_imag);              % Selección de la imagen
   recepción
```

3.2.8. Reconstrucción de imagen

```
%% RECONSTRUCCIÓN DE IMAGEN
reconstructed=reshape(typecast(uint8(bin2dec(char(reshape(imag_symb
   (:), 8,[])+'0').')), 'uint8'), [t1,t2,t3]);
disp("Imagen Reconstruida con éxito!!!")
```

3.2.9. Resultados obtenidos

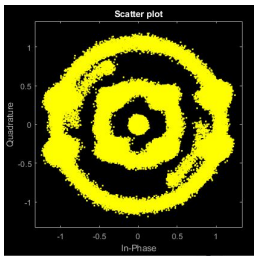

Figura 2: Modulacion 8QAM

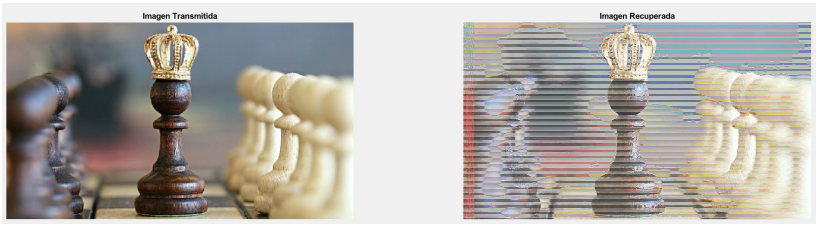

Figura 3: Imagen Recepcionada

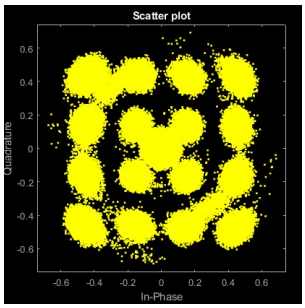

Figura 4: Modulacion 16QAM

Figura 5: Imagen Recepcionada

4. Conclusiones

- Se logro la implementación de un sistema de comunicaciones inalámbrico Tx y Rx basado en la modulación adaptativa QAM utilizando tecnología SDR BladeRF.Su factibilidad ademas de ofrecer un enfoque versátil para la transmisión y recepción de imágenes.
- La utilización de diferentes esquemas de modulación QAM, como 4-QAM, 8-QAM y 16-QAM, proporciona flexibilidad en la transmisión, permitiendo adaptar la modulación según las condiciones de la red y los requisitos de calidad de la imagen.
- El código del receptor demuestra una implementación sólida y efectiva de los procesos necesarios para recibir y decodificar imágenes transmitidas a través de un sistema de comunicaciones inalámbricas utilizando modulación adaptativa QAM.

5. Bibliografía

- Arevalo, N. E. Transmisión y recepción de imagen con modulación QPSK usando radio definido por software.
- PDF Documentation for MATLAB `https://www.mathworks.com/help/pdf_doc/matlab/index.html`
- Cory Beard and William Stallings. Wireless Communication Networks and Systems. Pearson, 2018. David Tse and Pramod Viswanath. Fundamentals of Wireless Communication.
- W Tomasi, G M Hernandez y V G Pozo. Sistemas de comunicaciones electronicas. Pearson Educacion, 2003. isbn: 9789702603160.

6. Anexos

6.1. Anexo 1:Transmisor

```
%% CREACIÓN DEL PREAMBULO
clear all
clc
close all
barker = comm.BarkerCode(... % For preamble
    'Length',13,'SamplesPerFrame',16)
preamble = barker();
preambul=vertcat(preamble, preamble,preamble,preamble);
M=8;                                % Orden de modulacion
YourImage=imread('ajedrez.jpg');    % Leer imágenes desde archivos
sizeOfImage = size(YourImage);      % Tamaño de la imagen
x=YourImage(:);
signal=vertcat(x);
bits = reshape((dec2bin(typecast(signal, 'uint8'), 8) - '0').', 1,
   []); % Secuencia de bits de la imagen
imglength=length(bits);             % Longitud de imagen
t1=sizeOfImage(1);                  % Dimensiones de la imagen
t2=sizeOfImage(2);
t3=sizeOfImage(3);
```

```
% Conversión a bits de dimensiones en 3 bytes
tamano1=dec2bin(t1,3*8)-'0';
tamano2=dec2bin(t2,3*8)-'0';
tamano3=dec2bin(t3,3*8)-'0';
tfinal=[tamano1';tamano2';tamano3']; % Matriz concatenada con
   dimensiones de la imagen
info=dec2bin(imglength,3*8)-'0';     % Longitud de los bits de la
   imagen en 3 bytes
modulac=dec2bin(M,3*8)-'0';          % Orden de la modulación 4-QAM,
   8-QAM O 16-QAM en 3 bytes
senal_digital=[bits'];
data=senal_digital;                  % Aplicamos la modulación 4-QAM,
    8-QAM O 16-QAM a la información de la imagen
txSig = qammod(data,M, ...
    InputType='bit', ...
    UnitAveragePower=true);
%% PROCESAMIENTO DE DATOS DE LA IMAGEN
datos_aleatorios=randi([0, 1], 2000, 1);
informacion=[info';tfinal;modulac'];  % Cinco segmentos de 24 bits(96
    bits)
informacion_modulada=pskmod(informacion,2);
trush=pskmod(datos_aleatorios,2);
scatterplot(txSig);
datas=[trush;preambul;informacion_modulada;txSig;trush]; % Paquete de
    información total
figure                               % Graficamos la información
stem(datas)
figure
stem(preambul)
%% PARÁMETROS PARA EL "PULSE SHAPING"
span=10;rolloff=0.5;sps=4;R = 1000;Fs = R * sps;
%% FILTRO TRANSMISOR
txfilter=comm.RaisedCosineTransmitFilter(...
    Shape='Square root',...
    RolloffFactor=rolloff,...
    FilterSpanInSymbols=span,...
    OutputSamplesPerSymbol=sps);
figure                           % Gráficamos el filtro
impz(txfilter.coeffs.Numerator)
txSignal=txfilter(datas);
scatterplot(txSignal,sps)
%% GUARDAR ARCHIVO BIN
save_sc16q11("C:\Users\bryan\Desktop\matlab\real\proifel.sc16q11",
   txSignal);%banderas_peru_prueba.sc16q11 no funciona
%% CARGAR SEÑAL TRANSMITIDA
signal=load_sc16q11("C:\Users\bryan\Desktop\matlab\real\proifel.
   sc16q11");
scatterplot(signal,sps)
figure()
plot(real(signal))
hold on
plot(imag(signal))
title('Señal x(n) real e imaginaria');
xlabel('n')
grid on
scatterplot(txSignal,sps)
```

6.2. Anexo 2:Receptor

```
1
2  % DEFINICIÓN DE PARÁMETROS
3  clear all;close all;clc;
4  span=10;rolloff=0.5;sps=4;R=1000;Fs = R * sps;
5  % Cargar el archivo recepcionado por medio del SDR
6  txSig=load_sc16q11("C:\Users\estep\OneDrive\Escritorio\MATLAB_CODES\
       R_16.sc16q11");
7  % Gráfica de la señal recepcionada por medio del SDR
8  figure(2)
9  plot(real(txSig))
10 hold on
11 plot(imag(txSig))
12 xlabel('n')
13 grid on
14 scatterplot(txSig,sps)
15
16 %% SINCRONIZACIÓN DE FRECUENCIA
17 compensacion=comm.CoarseFrequencyCompensator( ...
18 Modulation='QAM', ...
19 SampleRate=Fs, ...
20 FrequencyResolution=10);                    % Compensación de
       frecuencia 0.0001
21 [compensado,estimate]=compensacion(txSig);
22 freqCompInfo = info(compensacion)           % Información del
       compensado
23 disp("La Compensación en Frecuencia es de: " )
24 disp(estimate)
25 figure
26 plot(real(compensado))
27 hold on
28 plot(imag(compensado))
29 scatterplot(compensado,4)
30 %% FILTRADO DE LA SEÑAL RECEPCIONADA
31 filtro_receptor=comm.RaisedCosineReceiveFilter(...
32 Shape="Square root",...
33 RolloffFactor=0.5,...
34 FilterSpanInSymbols=span,...
35 InputSamplesPerSymbol=sps,...
36 DecimationFactor=2);                        % Diezmado por un
       factor de dos
37 simbolos=filtro_receptor(compensado);       % Aplicación de
       Filtrado
38 scatterplot(simbolos,2)                     % Normalización
39 %% SINCRONIZACIÓN DE TIEMPO
40 timingSync=comm.SymbolSynchronizer(...      % Función
       SymbolSynchronizer
41 Modulation="PAM/PSK/QAM",...
42 SamplesPerSymbol=2,...
43 NormalizedLoopBandwidth=0.004...
44 ,...
45 DampingFactor=11,...
46 TimingErrorDetector="Mueller-Muller (decision-directed)"...
47 );
48 syncSimbolos=timingSync(simbolos);           % Corregir Retardo
49 scatterplot(syncSimbolos)
50 %% SINCRONIZACIÓN DE FASE
51 carrierSync=comm.CarrierSynchronizer(...     % Función
       CarrierSynchronizer
52 SamplesPerSymbol=1,...
53 ModulationPhaseOffset="Auto",...
```

```
NormalizedLoopBandwidth=0.0033 ...
,...
Modulation='QAM',...
DampingFactor=0.8 ...
);
CarriersyncSimbolos=carrierSync(syncSimbolos); % Corregir en fase
scatterplot(CarriersyncSimbolos,1)
%% DETECCIÓN DEL PREÁMBULO
barker = comm.BarkerCode(...
   'Length',13,'SamplesPerFrame',16)
preamble = barker();
preambul=[preamble;preamble;preamble;preamble];
thr = 0.89*sum(abs(preambul).^2);
demo=pskdemod(CarriersyncSimbolos,2);      % Demodulacion en BPSK de
   la señal sincronizaca
mo=pskmod(demo,2);                         % Modulacion en BPSK
prbdet= comm.PreambleDetector(Preamble=preambul,Detections='All',
   Input='Symbol',Threshold=thr);
[idx,detmet] = prbdet(mo);                 % Detectamos el indice de
   la señal
scatterplot(real(CarriersyncSimbolos))
%% EXTRAER TAMAÑOS DE LA IMAGEN
inicio=idx-63;                              % Establecemos el inicio
   del preámbulo
inicio_info=idx+1;                          % Inicio de los bits de
   información
info_symb=CarriersyncSimbolos(inicio_info:inicio_info+119); %
   Extraemos el numero de muestras que contiene la información
info_bits=pskdemod(1*info_symb,2);           % Demodulamos el
   preambulo e información
leng_bits=info_bits(1:24);                  % Establecemos la longitud
    de bits de la imagen
%% EXTRAER LAS DIMENSIONES DE LA IMAGEN RECEPCIONADA
bits_t1=info_bits(25:48);                   % Extraemos las dimensiones
    de la imagen recepcionada
bits_t2=info_bits(49:72);
bits_t3=info_bits(73:96);
modulac=info_bits(97:120);                  % Extraemos el orden de
   modulacion 4-QAM, 8-QAM Y 16-QAM
long_imag=bin2dec(char(leng_bits'+'0'));    % Realizamos la conversión
    con bin2dec, e imprimimos los valores
t1=bin2dec(char(bits_t1'+'0'));
t2=bin2dec(char(bits_t2'+'0'));
t3=bin2dec(char(bits_t3'+'0'));
M=bin2dec(char(modulac'+'0'));              % Realizamos la conversión
   con bin2dec, e imprimimos los el orden de modulación
data_imag=CarriersyncSimbolos(inicio_info+120:end); % Extraemos el
   paquete de información de la señal
scatterplot(data_imag)
%% DEMODULACION QAM ADAPTATIVA
bits=qamdemod(data_imag,M,OutputType="bit", UnitAveragePower=false);
   %
scatterplot(data_imag)
imag_symb=bits(1:long_imag);           % Selección de la imagen
   recepción
%% CONSTRUCCIÓN DE LA IMAGEN ORIGINAL
YourImage=imread('ajedrez.jpg');          % Leer imágenes desde
   archivos
sizeOfImage = size(YourImage);         % Tamaño de la imagen
x=YourImage(:); w=sizeOfImage';        % Conversión vector columna
```

```
signalimage=vertcat(x);                    % Imagen + tamaño
bitsimage = reshape((dec2bin(typecast(signalimage, 'uint8'), 8) - '0'
    ).', 1, []); % Secuencia de bits de la imagen
data=bitsimage';
L=length(data)

%% RECONSTRUCCIÓN DE IMAGEN
reconstructed=reshape(typecast(uint8(bin2dec(char(reshape(imag_symb
    (:), 8,[])+'0').')), 'uint8'), [t1,t2,t3]);
disp("Imagen Reconstruida con éxito!!!")
%% COMPARACIÓN DE LA IMAGEN ORIGINAL VS IMAGEN RECUPERADA
figure
subplot(1,2,1)
imshow(YourImage)
title("Imagen Transmitida")
subplot(1,2,2)
imshow(reconstructed)
title("Imagen Recuperada")
```

6.3. Anexo 3:Pruebas

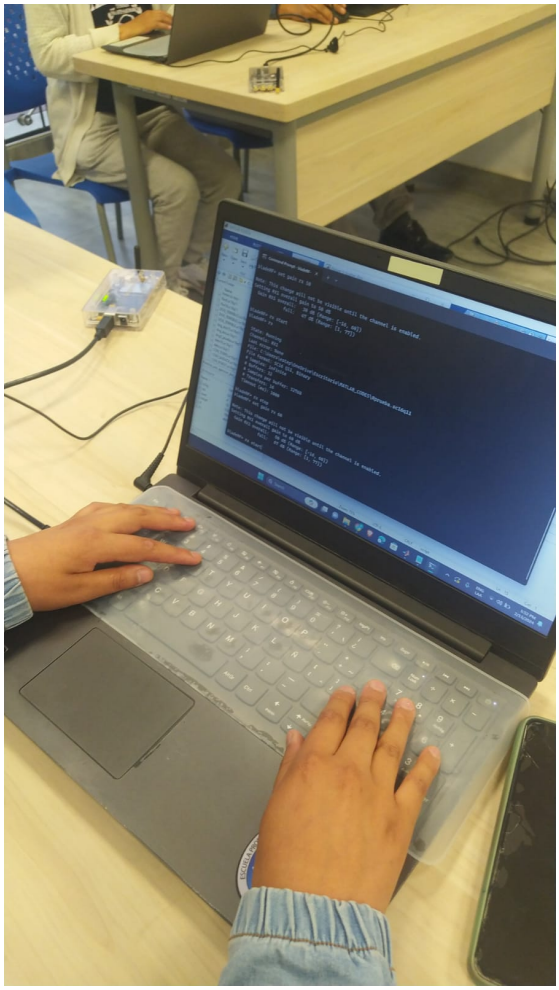

Figura 6: Envio y recepcion usando SDR BladeRF