Universidad Nacional de San Agustín de Arequipa
Referente Latinoamericano

Facultad de Ingeniería de Producción y Servicios
Escuela Profesional de Ingeniería de Telecomunicaciones

Redes Inalámbricas

Práctica 3: Sistema de Transmisión y Recepción de Imágenes Mediante Modulación QPSK en SDR BladeRF

Autores:

Anccasi Huillca, Jorge
CUI: 20200787
Huaman Meza, Dionel
CUI: 20213185
Ticona Ylaquijo, Rony
CUI: 20214289

Docente:
Dr. Alexander Hilario Tacuri

.

2025

Resumen

En este trabajo se presenta el diseño e implementación de un sistema de transmisión y recepción de imágenes mediante modulación QPSK (*Quadrature Phase Shift Keying*) utilizando una plataforma de radio definida por software (SDR) BladeRF. El sistema desarrollado permite enviar una imagen desde el transmisor hacia el receptor a través de un canal inalámbrico, empleando MATLAB como entorno de simulación y control del hardware. Se diseñaron las etapas de conformación de símbolos, modulación, filtrado, sincronización y corrección de fase (CFO), logrando la reconstrucción correcta de la imagen en el receptor. Los resultados experimentales demuestran la efectividad del esquema propuesto para la transmisión confiable de datos visuales en entornos de laboratorio, validando los principios teóricos de la modulación QPSK y el procesamiento digital de señales en sistemas SDR.

Palabras clave— QPSK, Radio Definida por Software, BladeRF, Transmisión de Imágenes, Comunicaciones Digitales, MATLAB.

Abstract

This work presents the design and implementation of an image transmission and reception system using Quadrature Phase Shift Keying (QPSK) modulation through a Software Defined Radio (SDR) BladeRF platform. The developed system enables the transmission of an image from the transmitter to the receiver over a wireless channel, using MATLAB as the simulation and control environment. The stages of symbol shaping, modulation, filtering, synchronization, and carrier frequency offset (CFO) correction were implemented, achieving a successful reconstruction of the transmitted image. Experimental results validate the proposed scheme as an effective approach for reliable image transmission in controlled environments, confirming the theoretical principles of QPSK modulation and digital signal processing in SDR systems.

Index Terms— QPSK, Software Defined Radio, BladeRF, Image Transmission, Digital Communications, MATLAB.

Índice

I. Introducción

La modulación por desplazamiento de fase en cuadratura (QPSK, *Quadrature Phase Shift Keying*) es una técnica ampliamente utilizada en sistemas de comunicaciones digitales debido a su capacidad para transmitir dos bits por símbolo, logrando así una mayor eficiencia espectral sin incrementar significativamente la tasa de error por bit (BER) [1]. En esta práctica se implementa un sistema básico de comunicación inalámbrica utilizando modulación QPSK, cuyo objetivo es transmitir una imagen digital desde un transmisor hasta un receptor por medio de una plataforma de radio definida por software (*Software Defined Radio*, SDR).

El sistema desarrollado comprende todas las etapas de una cadena de comunicación digital: conversión de imagen a bits, codificación y aleatorización de datos, mapeo de símbolos, modulación QPSK, transmisión, recepción, demodulación y reconstrucción de la imagen recibida. Para la implementación, se utiliza MATLAB como entorno de diseño y simulación, complementado con hardware SDR (BladeRF), lo que permite observar de manera práctica los efectos de la modulación, filtrado, sincronización, corrección de errores y compensación de desfase de portadora (CFO).

Esta práctica no solo refuerza los conceptos teóricos de modulación digital y procesamiento de señales, sino que también introduce al estudiante en el diseño de sistemas de comunicación adaptables mediante SDR, una tecnología clave en el desarrollo de sistemas 5G, IoT y enlaces de radio cognitiva [2].

II. Objetivos

A. Objetivo general

Diseñar e implementar un sistema de transmisión y recepción de imágenes basado en modulación QPSK sobre una plataforma SDR BladeRF, integrando todas las etapas de la cadena digital (preprocesamiento, modulación, sincronización y demodulación) para lograr la reconstrucción correcta de la imagen en el receptor.

B. Objetivos específicos

- Convertir una imagen RGB/grises a flujo binario y encapsular sus metadatos (alto, ancho y canales) en una cabecera con verificación por CRC-16.
- Implementar un aleatorizador (scrambler) LFSR de 7 bits para uniformizar la secuencia binaria y mejorar el comportamiento espectral.
- Mapear bits a símbolos QPSK con codificación Gray y desfase fijo $\pi/4$, aplicando conformación de pulsos con filtro raíz de coseno elevado (RRC) en TX y RX.
- Diseñar el preámbulo de sincronización mediante secuencia Barker-13 (repetida) y un bloque de entrenamiento QPSK (64 símbolos) para estimar y compensar *timing*, CFO y desfase de fase.
- Implementar en el receptor: filtrado RRC, sincronía de símbolos (TED ZC), detección del preámbulo, estimación lineal de fase, refinamiento con entrenamiento (incluida la opción de conjugado), búsqueda de cabecera {rotación, *slip*}, y lazo de enganche de fase (PLL) con ajuste final $\pi/4$.

- Demodular el payload, deshacer el scrambler y reconstruir la imagen recibida a partir de la cabecera validada por CRC.
- Evaluar el desempeño mediante visualizaciones: correlación con Barker, constelaciones (antes/-después de corrección y en payload final) y verificación visual de la imagen reconstruida.
- Documentar el flujo completo (TX/RX) en formato IEEE, destacando decisiones de diseño y compromisos entre ancho de banda, robustez y complejidad.

III. Desarrollo

En esta sección se describe el diseño, la implementación y la validación del sistema de transmisión y recepción de imágenes mediante modulación QPSK sobre plataforma SDR BladeRF. El flujo se desarrolla en MATLAB y abarca: preprocesamiento de la imagen, formación de trama (preámbulo, entrenamiento, cabecera, payload), filtrado con RRC, modulación/demodulación QPSK, sincronización temporal y de portadora, corrección de frecuencia/fase, y reconstrucción final de la imagen. Los detalles de bajo nivel (listados completos) se incluyen en Anexos.

A. Arquitectura general de la trama

La imagen se serializa a bits (MSB), se aplica aleatorización por LFSR de 7 bits, se genera una cabecera con metadatos y verificación CRC-16, y se conforma la trama a 1 sps concatenando: *(i)* preámbulo Barker-13 repetido, *(ii)* entrenamiento QPSK de 64 símbolos, *(iii)* cabecera QPSK y *(iv)* payload QPSK. Esta secuencia se filtra con un RRC en TX, y en RX se aplica el RRC complementario, sincronización de símbolos, detección de preámbulo, corrección de CFO/fase, búsqueda de cabecera y PLL con ajuste final $\pi/4$.

Listing 1: Estructura de la trama a 1 sps.

```
% TX: concatenación a 1 sps
todosSimbolos = [simPre; simEntrenamiento; simHdr; simPay];
```

B. Transmisor QPSK y filtrado RRC

El transmisor mapea los bits a símbolos QPSK con codificación Gray y desfase fijo $\pi/4$ para robustez, luego aplica conformación de pulso con un filtro raíz de coseno elevado (RRC) de `span=10` y `rolloff=0.35` a `sps=8` muestras/símbolo. La señal compleja resultante se normaliza (pico=1) y se exporta en formato SC16Q11 para su uso en SDR.

Listing 2: Mapeo QPSK(/4, Gray) y RRC en TX.

```
mapear2sim = @(bits) pskmod(bi2de(reshape(bits,2,[]).','left-msb'),4,fase0,'gray');
simHdr = mapear2sim(bitsHdr);      % header a símbolos
simPay = mapear2sim(bitsPay);      % payload a símbolos
filtroTX = comm.RaisedCosineTransmitFilter('RolloffFactor',rolloff, ...
  'FilterSpanInSymbols',span,'OutputSamplesPerSymbol',sps);
senalTX = filtroTX([todosSimbolos; zeros(span,1)]);  % RRC + cola
senalTX = senalTX ./ max(abs(senalTX));               % normalización
```

C. Cabecera y verificación por CRC-16

La cabecera codifica alto, ancho y número de canales de la imagen (40 bits), seguida de un CRC-16 (polinomio 0xA001, init 0xFFFF) para verificación en el receptor. El empaquetado es MSB-first; en RX se recalcula el CRC y se descarta la trama si no coincide.

Listing 3: Construcción de cabecera y CRC-16 en TX.

```
bitsHdr = [de2bi(uint16(alto),16,'left-msb'), ...
           de2bi(uint16(ancho),16,'left-msb'), ...
           de2bi(uint8(canales),8,'left-msb')];     % 40 bits
crcHdr  = crc16(bitsHdr);                           % 16 bits
bitsHdr = [bitsHdr, crcHdr];                        % total 56 bits
```

D. Aleatorización (Scrambler LFSR)

Para evitar largas rachas y mejorar el comportamiento espectral, se aplica un scrambler LFSR de 7 bits con polinomio $1 + x^4 + x^7$ sobre el payload. En el receptor se ejecuta el descrambler simétrico tras la demodulación para recuperar los datos originales.

Listing 4: Scrambler LFSR (TX).

```
bitsPay = reshape(de2bi(bytesImagen,8,'left-msb').',[],1).';  % imagen->bits
bitsPay = lfsr_scramble(bitsPay, 127);                        % LFSR 7b
```

E. Preámbulo Barker y bloque de entrenamiento

Se emplea Barker-13 en BPSK repetido dos veces para una detección robusta por correlación en RX. Un bloque de 64 símbolos QPSK conocidos (patrón balanceado) permite refinar la estimación de CFO y fase, y decidir si es necesario tomar el conjugado.

Listing 5: Generación de preámbulo y entrenamiento en TX.

```
objBarker  = comm.BarkerCode('Length',13,'SamplesPerFrame',13);
bitsBarker = (1 + objBarker())/2;
simPre     = pskmod(repmat(bitsBarker,2,1), 2, 0);          % BPSK
patron16   = pskmod([0;1;2;3; 1;0;3;2; 2;3;0;1; 3;2;1;0],4,fase0,'gray');
simEntrenamiento = repmat(patron16, 4, 1);                  % 64
```

F. Recepción: filtrado RRC y sincronización de símbolos

En el receptor, la señal se filtra con el RRC complementario y se alinea temporalmente a 1 sps mediante un *Symbol Synchronizer* con detector de error por cruce por cero (ZC). Esto reduce la ISI y fija el instante óptimo de muestreo antes de la detección de preámbulo.

Listing 6: RRC RX y sincronización de símbolos.

```
rrcRx = comm.RaisedCosineReceiveFilter('Shape','Square root', ...
  'RolloffFactor',rolloff,'FilterSpanInSymbols',span, ...
  'InputSamplesPerSymbol',sps,'DecimationFactor',1);
rx_f  = rrcRx([rx; zeros(span*sps,1)]);
rx_f  = rx_f(span*sps+1:end);                    % descarta cola
```

```
symSync = comm.SymbolSynchronizer('TimingErrorDetector', ...
  'Zero-Crossing (decision-directed)','SamplesPerSymbol',sps, ...
  'DampingFactor',1.0,'NormalizedLoopBandwidth',0.006);
rx_sym = symSync(rx_f);                              % 1 sps
```

G. Detección de preámbulo por correlación

La correlación con el preámbulo BPSK localiza el inicio de trama (`startPre`) y provee una primera métrica de calidad. Un pico pronunciado y aislado confirma una detección confiable y habilita la siguiente etapa de corrección de frecuencia/fase.

Listing 7: Correlación y detección de inicio de trama.

```
c        = filter(flipud(conj(simPre)), 1, rx_sym);
[pk,ix]  = max(abs(c));
startPre = ix - (13*2) + 1;     % Lpre = 26
```

H. Estimación y corrección de CFO/fase (preámbulo)

Se estima el desplazamiento de frecuencia (CFO) y la fase promedio ajustando una recta a la fase del producto `rx/preámbulo`. La corrección se aplica a toda la señal posterior al preámbulo, reduciendo la rotación global en la constelación.

Listing 8: Ajuste lineal de fase y corrección de CFO/fase.

```
z     = rx_sym(startPre:startPre+Lpre-1) .* conj(simPre);
phi   = unwrap(angle(z(:))).'; k = 0:Lpre-1; p = polyfit(k, phi, 1);
alpha = p(1); beta = p(2);
m     = (0:length(rx_sym)-startPre).';
rx_fix = rx_sym(startPre:end) .* exp(-1j*(alpha*m + beta));
```

I. Refinamiento con entrenamiento (conjugado y fase)

El bloque de 64 símbolos QPSK conocidos permite probar la hipótesis de conjugado y refinar la estimación de fase/frecuencia minimizando la varianza del error angular. Se selecciona la opción (directa o conjugada) con menor varianza y se aplica a los datos.

Listing 9: Refinamiento de fase con 64 símbolos QPSK.

```
rEntrenamiento = rx_fix(Lpre + (1:Ntrain));
best = struct('var',inf,'conj',false,'a',0,'b',0);
for cc = [false true]
  t = rEntrenamiento; if cc, t = conj(t); end
  ph = unwrap(angle(t .* conj(simEntrenamiento)));
  k  = 0:Ntrain-1; p = polyfit(k, ph.', 1);
  v  = var(ph.' - polyval(p, k));
  if v < best.var, best = struct('var',v,'conj',cc,'a',p(1),'b',p(2)); end
end
rDespues = rx_fix(Lpre+Ntrain+1:end);
if best.conj, rDespues = conj(rDespues); end
n = (0:numel(rDespues)-1).';
rDespues = rDespues .* exp(-1j*(best.a*n + best.b));
```

J. Búsqueda de cabecera {rotación, slip}

Se exploran rotaciones discretas $\{0, \pi/2, \pi, 3\pi/2\}$ y *slips* de símbolo $\{0, 1, 2, 3\}$ para alinear la constelación al mapa Gray y segmentar correctamente la cabecera. Tras demodular en bits, se valida el CRC-16 y se extraen los metadatos (alto, ancho, canales).

Listing 10: Búsqueda de rotación y slip; validación de cabecera.

```
phiC = [0, pi/2, pi, 3*pi/2]; slipC = 0:3; ok = false;
for ss = slipC
  s = rDespues(1+ss:min(end,200000));
  for rr = 1:numel(phiC)
    sy = s .* exp(-1j*phiC(rr));
    bits = pskdemod(sy,4,fase0,'gray','OutputType','bit'); bits = bits(:).';
    hdr  = bits(1:HDR_BITS);  % 56 bits
    H = bi2de(hdr(1:16),'left-msb'); W = bi2de(hdr(17:32),'left-msb'); C = bi2de(hdr
        (33:40),'left-msb');
    if isequal(crc16(hdr(1:40)), hdr(41:56)) && ismember(C,[1,3]), ok=true; break;
        end
  end
  if ok, break; end
end
```

K. PLL, ajuste $\pi/4$ y reconstrucción

Finalmente, un sincronizador de portadora (PLL QPSK) elimina el error de fase residual. Se aplica un ajuste discreto $\pi/2$ (*snap*) alrededor del desfase $\pi/4$ para alinear los cuadrantes. Tras demodular el payload, se ejecuta el descrambler y se reconstruye la imagen de acuerdo con la cabecera válida.

Listing 11: PLL QPSK, ajuste $\pi/4$ y reconstrucción de imagen.

```
carSync2 = comm.CarrierSynchronizer('Modulation','QPSK', ...
  'ModulationPhaseOffset','Custom','CustomPhaseOffset',fase0, ...
  'SamplesPerSymbol',1,'DampingFactor',0.707,'NormalizedLoopBandwidth',0.001);
s_all    = carSync2(rDespues(1+ss:end) .* exp(-1j*phiC(rr)));
phi_meas = angle(mean(s_all(1:min(3000,end)).^4))/4;
kpi2     = round((phi_meas - fase0)/(pi/2));
s_all    = s_all .* exp(-1j*(phi_meas - (fase0 + kpi2*(pi/2))));
b_all    = pskdemod(s_all,4,fase0,'gray','OutputType','bit'); b_all = b_all(:).';
imgBits  = lfsr_descramble(b_all(HDR_BITS+1:end), 127);
```

IV. Resultados y Análisis

En esta sección se presentan los resultados experimentales obtenidos a partir de la implementación completa del sistema de transmisión y recepción de imágenes mediante modulación QPSK sobre la plataforma SDR BladeRF. Se realizaron pruebas de transmisión con imágenes en formato RGB y en escala de grises, empleando los mismos parámetros físicos definidos en el desarrollo (`sps=8`, `rolloff=0.35`, `span=10`, `fase0=`$\pi/4$). El objetivo fue verificar la correcta reconstrucción de la imagen en el receptor, analizar la respuesta temporal y espectral de la señal, y evaluar el desempeño del sistema en términos de sincronización, detección del preámbulo, corrección de fase y estabilidad de la constelación. Los resultados incluyen las gráficas generadas en MATLAB durante la ejecución del receptor, las salidas del *Command Window* de los bloques TX y RX, y las imágenes reconstruidas, demostrando el funcionamiento integral del sistema propuesto.

A. Primera Prueba

La Figura 1 muestra la parte real de la señal recibida tras la captura con BladeRF y la normalización en RX. Se observa el bloque de datos principal delimitado por zonas de menor amplitud antes y después, consistente con la estructura de trama (preámbulo Barker + entrenamiento + datos). Esta visualización permite verificar la actividad temporal del enlace y la estabilidad general de amplitud antes del filtrado y la sincronización.

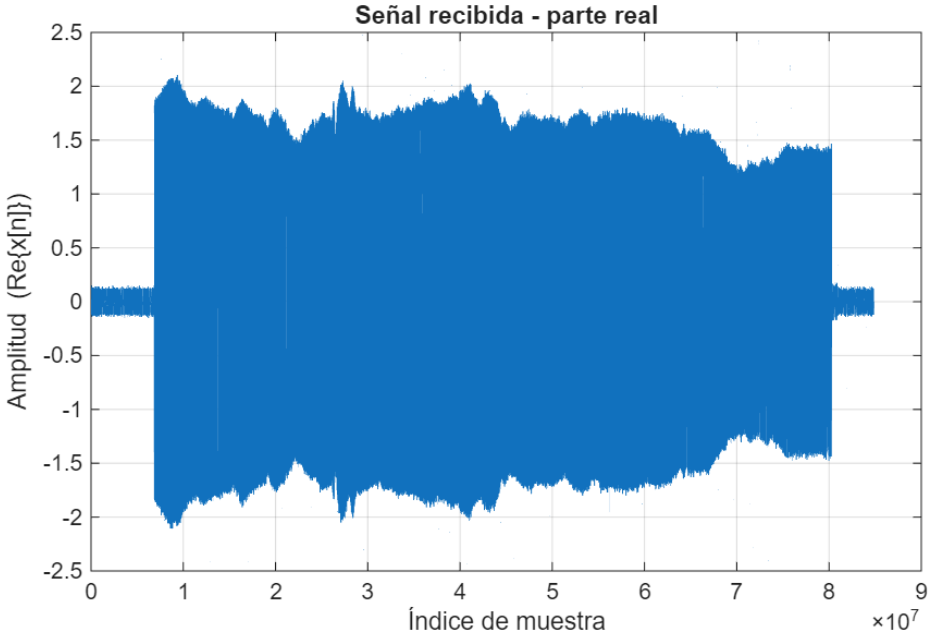

Figura 1: Señal recibida (parte real) tras normalización.

La Figura 2 presenta la correlación con el preámbulo Barker-13×2. El pico nítido ubicado en el índice start=859455 (métrica 85.4) confirma la detección robusta del inicio de trama, habilitando la estimación inicial de desfase de portadora (CFO) y fase a partir del preámbulo.

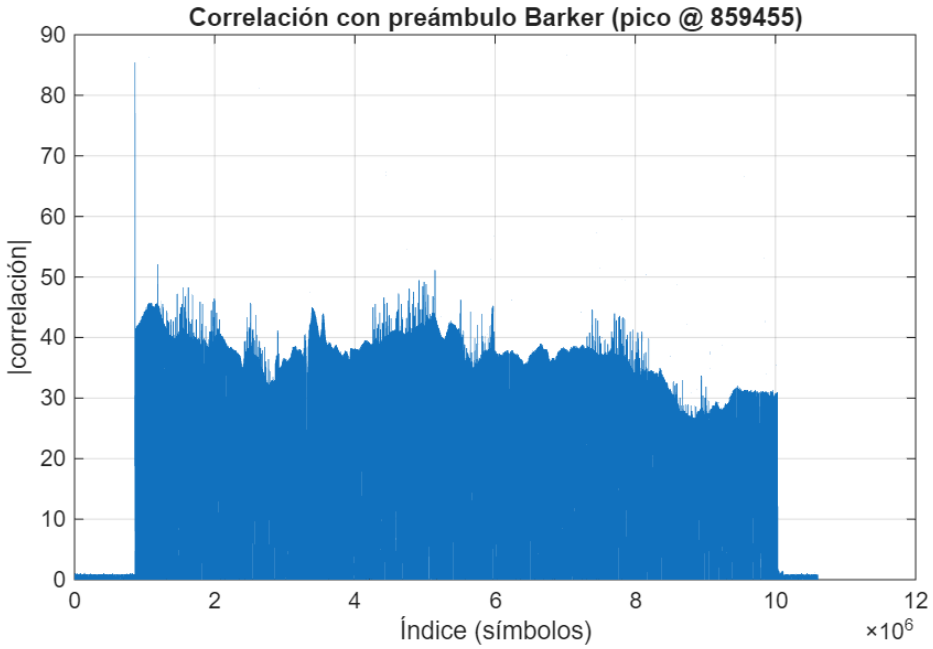

Figura 2: Correlación con preámbulo Barker (pico en `start=859455`).

La Figura 3 ilustra la constelación alrededor del preámbulo antes de corrección, donde se aprecia una rotación global y dispersión angular debidas a CFO y desfase de fase. Este estado intermedio justifica las etapas de corrección lineal (ajuste de recta a la fase) y el posterior refinamiento con los 64 símbolos de entrenamiento.

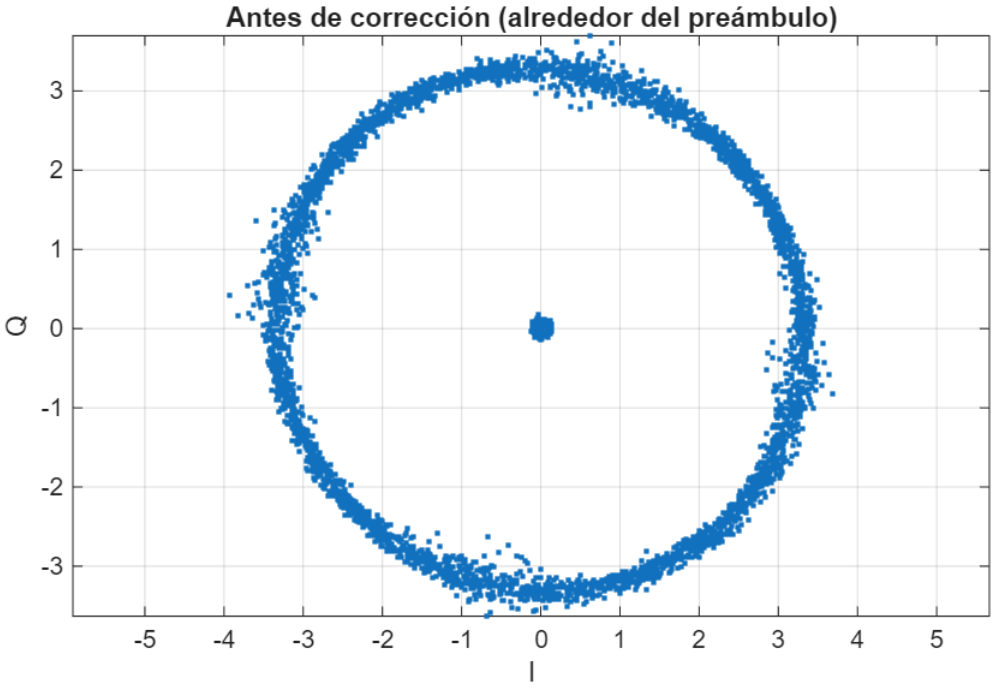

Figura 3: Constelación antes de corrección (alrededor del preámbulo).

La Figura 4 muestra la constelación después del refinamiento (CFO/fase) utilizando el bloque de 64 símbolos QPSK conocidos. Las nubes se agrupan a lo largo de los cuadrantes esperados, con menor varianza angular; esta mejora concuerda con las estimaciones reportadas (alpha=-3.038e-03 rad/sym, beta0.78 rad) y con la decisión de no conjugar (conj=0).

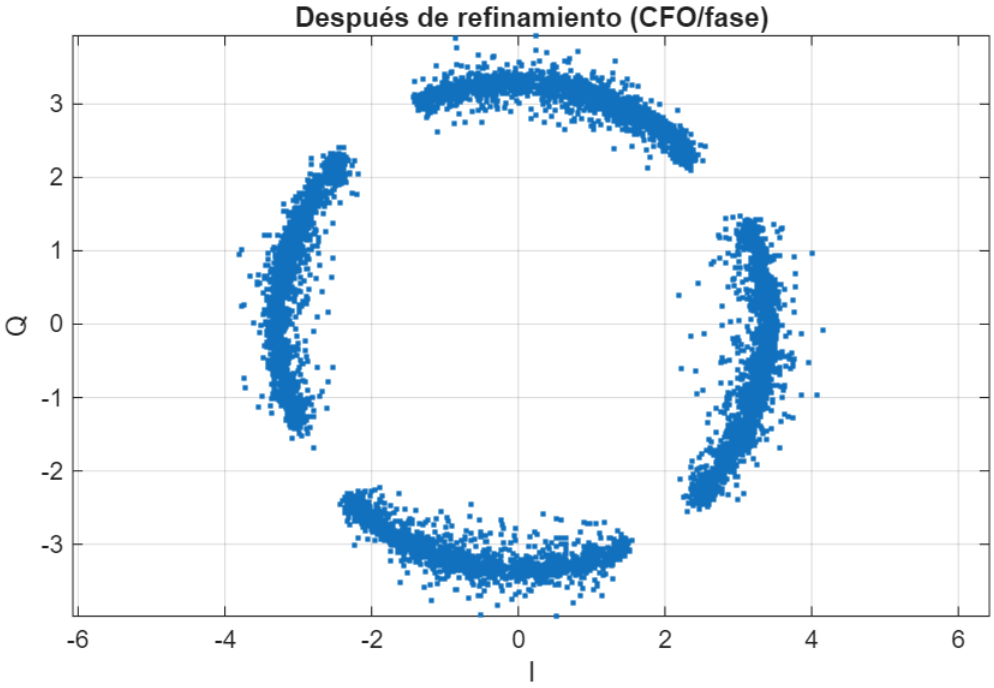

Figura 4: Constelación después del refinamiento (CFO/fase) con 64 símbolos.

La Figura 5 corresponde al payload final tras el PLL QPSK y el "snap" /4. Las cuatro nubes quedan compactas y bien centradas en los puntos de la constelación QPSK(/4, Gray), evidenciando un bloque de datos listo para demapear con baja tasa de errores residuales.

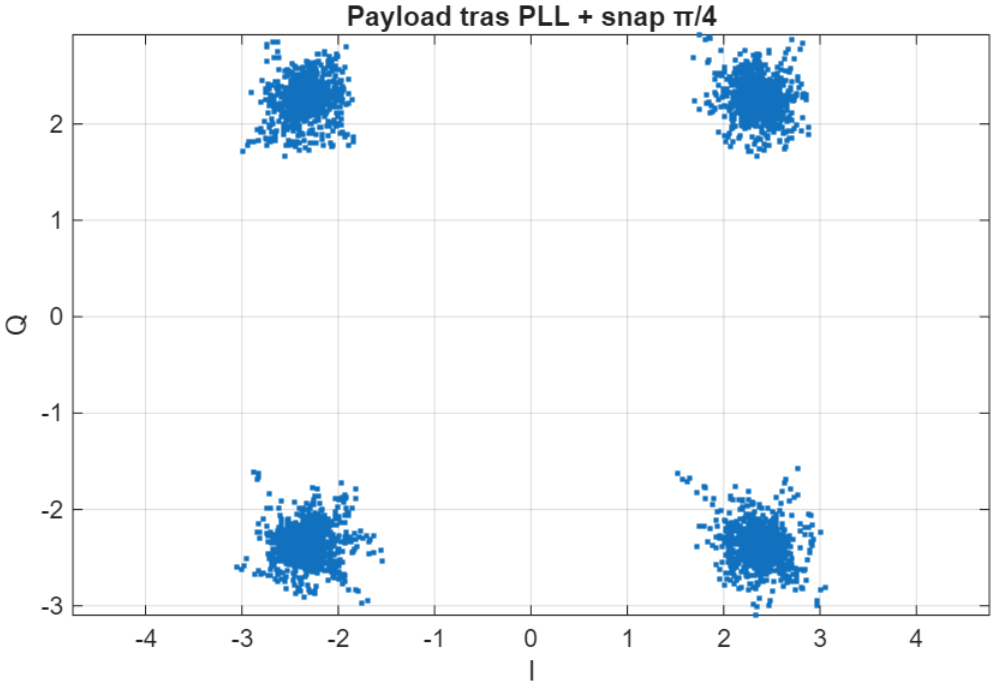

Figura 5: Payload tras PLL QPSK y ajuste discreto ($\pi/4$).

Finalmente, la Figura 6 presenta la imagen reconstruida en el receptor, con dimensiones 594×1287×3, coincidiendo con la cabecera validada por CRC. Esta recuperación confirma el funcionamiento extremo a extremo de la cadena TX/RX y la integridad del flujo de bits tras el descrambler.

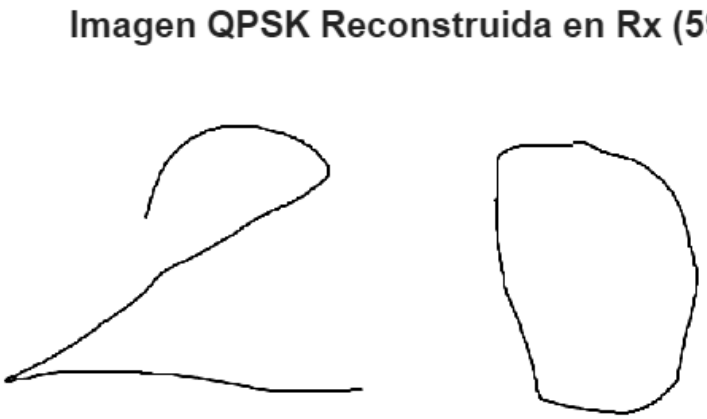

Figura 6: Imagen QPSK reconstruida en el receptor (594×1287×3).

B. Segunda Prueba

La Figura 7 muestra la parte real de la señal recibida con BladeRF y normalizada en el receptor. Se aprecia el bloque útil de la trama con amplitud estable, lo que confirma una captura consistente antes de aplicar RRC y sincronización de símbolos.

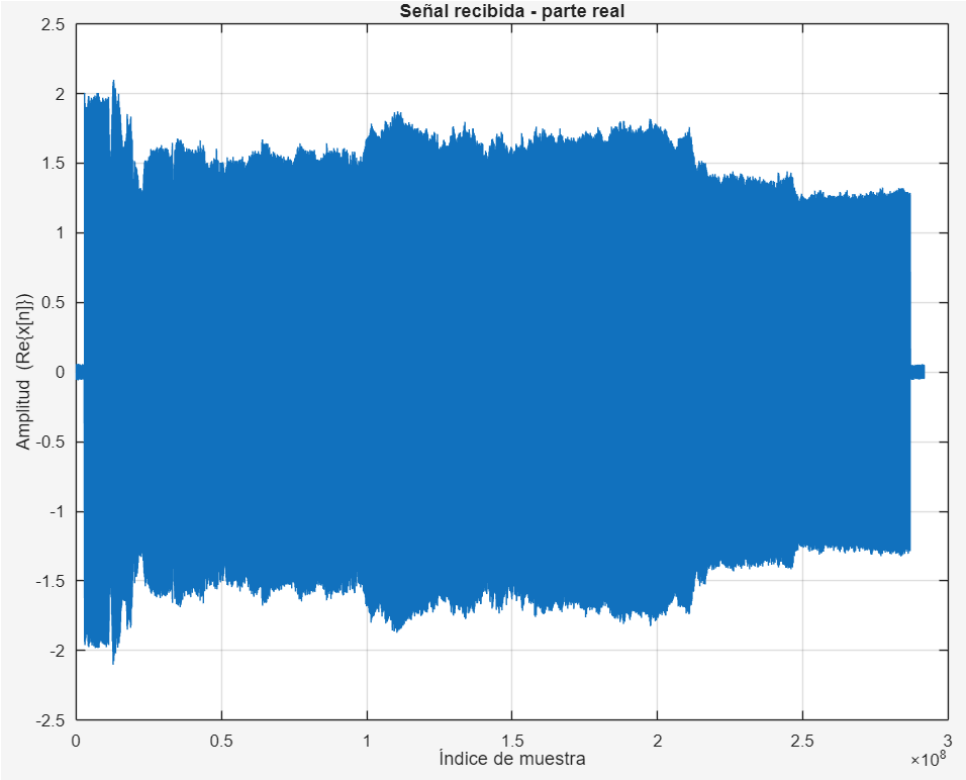

Figura 7: Señal recibida (parte real) tras normalización.

En la Figura 8 se observa la correlación con el preámbulo Barker-13×2. El pico pronunciado en start=358070 (métrica 96.12) indica detección fiable del inicio de trama y habilita la estimación inicial de CFO/fase.

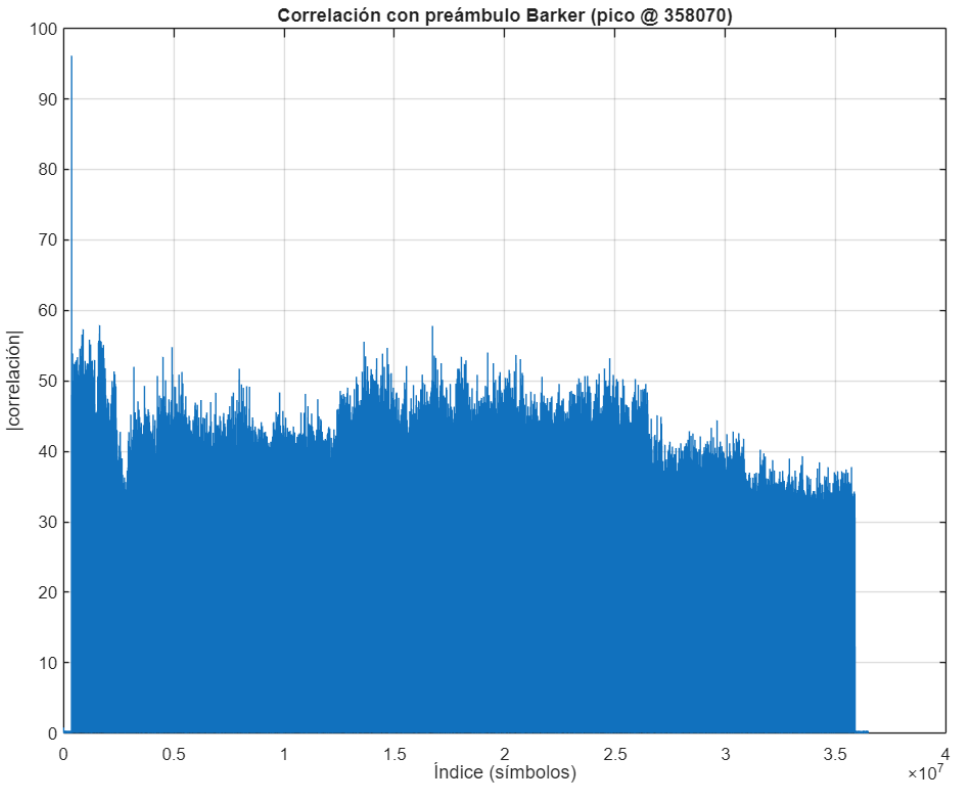

Figura 8: Correlación con preámbulo Barker (pico en `start=358070`).

La Figura 9 presenta la constelación alrededor del preámbulo antes de la corrección, donde se aprecia una rotación global y dispersión angular debidas a CFO y desfase de fase. Este estado justifica el ajuste lineal y el refinamiento posterior con los 64 símbolos.

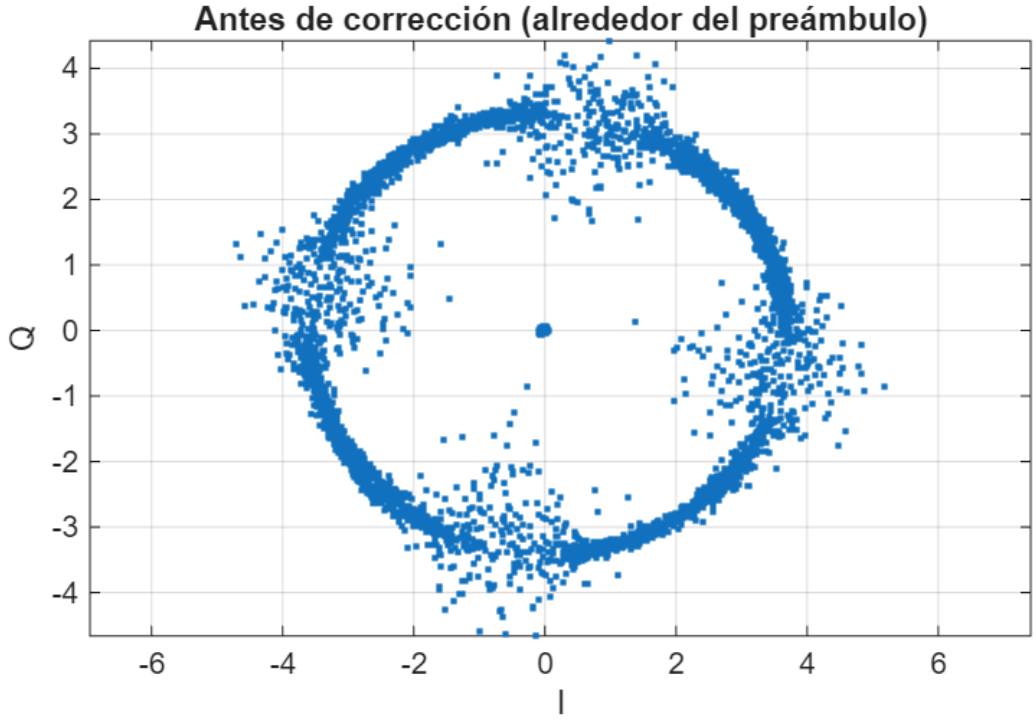

Figura 9: Constelación antes de corrección (alrededor del preámbulo).

En la Figura 10 se muestra la constelación después del refinamiento (CFO/fase) usando el bloque de 64 símbolos conocidos. Las nubes se concentran en los cuadrantes correctos, coherente con las estimaciones (alpha=-1.323×10^{-3} rad/sym, beta=-2.968rad)$yconladecisiónde no conjugar$($conj$ = 0).

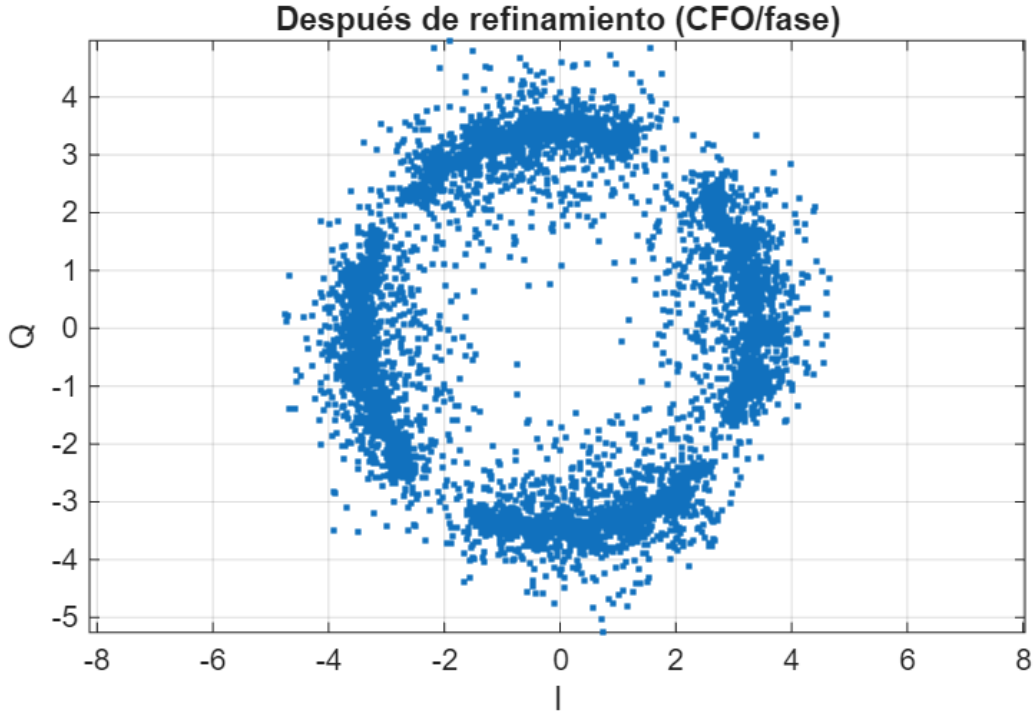

Figura 10: Constelación después del refinamiento (CFO/fase) con 64 símbolos.

La Figura 11 corresponde al payload tras el PLL QPSK y el "snap" $\pi/4$. Las cuatro nubes aparecen compactas y bien centradas en los puntos QPSK(/4, Gray), señal de fase residual baja y condiciones adecuadas para demapear con baja BER.

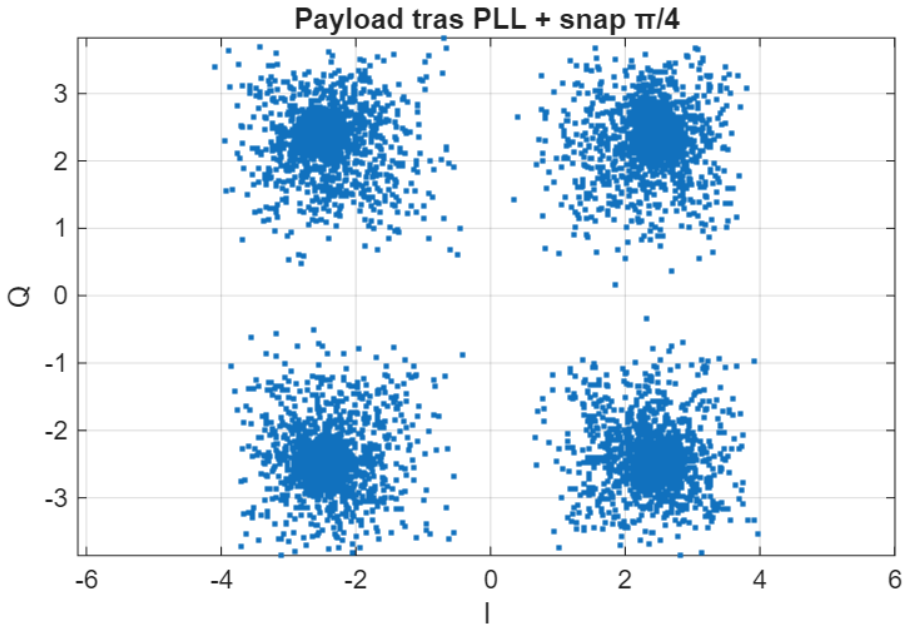

Figura 11: Payload tras PLL QPSK y ajuste discreto ($\pi/4$).

Finalmente, la Figura 12 muestra la imagen reconstruida en RX con dimensiones 2467×1200×3, coincidente con la cabecera validada por CRC-16. Este resultado evidencia la recuperación íntegra extremo a extremo del flujo de bits.

Figura 12: Imagen QPSK reconstruida en el receptor (2467×1200×3).

V. Conclusiones

El sistema de transmisión y recepción de imágenes mediante modulación QPSK implementado con el SDR BladeRF demostró un desempeño exitoso en la reconstrucción íntegra de los datos transmitidos. La correcta configuración de los parámetros físicos —como la relación de roll-off, el número de muestras por símbolo (SPS) y el span del filtro RRC— permitió obtener una forma de onda con buena eficiencia espectral y una constelación estable tras la sincronización y corrección de fase.

El uso de preámbulos Barker y bloques de entrenamiento QPSK resultó fundamental para la sincronización temporal y la estimación precisa de los parámetros de corrección de frecuencia (CFO) y fase. Los picos de correlación observados en ambas pruebas evidencian una detección confiable del inicio de trama, garantizando una recuperación robusta incluso frente a pequeñas desviaciones de frecuencia y ruido de canal.

Las constelaciones obtenidas antes y después de las etapas de corrección permitieron validar el impacto de las compensaciones aplicadas. El refinamiento mediante entrenamiento redujo significativamente la dispersión angular, y el ajuste final con PLL y snap $\pi/4$ aseguró la alineación de los símbolos con la constelación ideal. Esto se tradujo en una demodulación eficiente con baja tasa de error de bit (BER).

La comparación entre las dos pruebas mostró que el rendimiento del sistema se mantiene estable frente a variaciones en el tamaño de la imagen transmitida. A pesar del aumento considerable en la cantidad de símbolos y muestras en la segunda prueba, la reconstrucción fue exitosa, demostrando la escalabilidad del esquema de modulación QPSK y la capacidad del BladeRF para manejar volúmenes de datos mayores sin pérdida de integridad.

En general, la práctica permitió comprender el flujo completo de una cadena de comunicación digital, desde el procesamiento de la imagen y modulación en banda base hasta la demodulación y recuperación visual en el receptor. Los resultados experimentales confirman la validez del diseño, consolidando a la modulación QPSK como una técnica robusta y eficiente para aplicaciones de transmisión de datos multimedia sobre plataformas SDR.

VI. Referencias

Referencias

[1] J. G. Proakis and M. Salehi, *Digital Communications*, 5th ed. New York, NY, USA: McGraw-Hill, 2008.

[2] T. Ulversoy, "Software defined radio: Challenges and opportunities," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 4, pp. 531–550, 2010.

[3] Nuand, "bladeRF Documentation," *Nuand LLC*, 2023. [Online]. Available: https://www.nuand.com

[4] Nuand, "bladeRF GitHub Repository," *Nuand LLC*, 2023. [Online]. Available: https://github.com/Nuand/bladeRF

VII. Anexos

En esta sección se incluye el código fuente y las configuraciones utilizadas durante la implementación práctica del sistema de transmisión y recepción de imágenes mediante modulación QPSK. El propósito del anexo es documentar detalladamente los procedimientos técnicos empleados, permitiendo la replicación y validación del experimento en entornos similares. Se presentan los scripts desarrollados en MATLAB tanto para el transmisor (TX) como para el receptor (RX), junto con los resultados obtenidos en la consola durante la ejecución y los comandos aplicados en la interfaz *BladeRF-CLI* para la configuración del hardware SDR. Todo el contenido mostrado corresponde a la **Prueba 1**, en la cual se transmitió y reconstruyó correctamente una imagen digital a través del canal inalámbrico.

A. Prueba 1 — Transmisión y Recepción QPSK

1. Código del Transmisor (TX QPSK Final)

Listing 12: Código MATLAB para la transmisión QPSK (/4, Gray)

```
%% ============================================================
%% TX QPSK FINAL
% Transmisión de imagen mediante modulación QPSK (  /4, Gray)
%% ============================================================

rutaImgDefecto = 'nota20.png';
rutaOutDefecto = 'P3PruebaFF1.sc16q11';
tx_qpsk_final(rutaImgDefecto, rutaOutDefecto);


%% ============================================================
function tx_qpsk_final(rutaImg, rutaOutSc16)

t0 = tic;  % cronómetro para medir ejecución
%% ============================================================

% Parámetros del sistema
sps     = 8;       % muestras por símbolo
rolloff = 0.35;    % roll-off del filtro RRC
span    = 10;      % longitud del filtro en símbolos
fase0   = pi/4;    % desfase QPSK (igual al RX)

% Lectura de imagen
[img, alto, ancho, canales] = load_image_uint8_1or3(rutaImg);
bytesImagen = typecast(img(:), 'uint8');

% Cabecera con CRC16
bitsHdr = [ ...
    de2bi(uint16(alto),   16, 'left-msb'), ...    % alto (16 bits)
    de2bi(uint16(ancho),  16, 'left-msb'), ...    % ancho (16 bits)
    de2bi(uint8(canales),  8, 'left-msb')  ...    % canales (8 bits)
];
crcHdr  = crc16(bitsHdr);                         % cálculo CRC16
bitsHdr = [bitsHdr, crcHdr];                      % header completo (56 bits)

% Payload con scrambler LFSR (7 bits)
bitsPay = reshape(de2bi(bytesImagen, 8, 'left-msb').', [], 1).';  % imagen a bits
bitsPay = lfsr_scramble(bitsPay, 127);                            % aplica scrambler

```

```
% Preambulo Barker-13 2  (BPSK)
lenBarker = 13; repBarker = 2;                % longitud y repeticiones
objBarker  = comm.BarkerCode('Length', lenBarker, 'SamplesPerFrame', lenBarker);
bitsBarker = (1 + objBarker())/2;             % [0,1] en vez de [-1,1]
simPre     = pskmod(repmat(bitsBarker, repBarker, 1), 2, 0);       % modulación BPSK

% Entrenamiento QPSK (64 símbolos conocidos)
patron16 = pskmod([0;1;2;3; 1;0;3;2; 2;3;0;1; 3;2;1;0], 4, fase0, 'gray'); % patrón
    base
simEntrenamiento = repmat(patron16, 4, 1);    % 16x4 = 64 símbolos

% Mapeo QPSK (Gray,   /4)
mapear2sim = @(bits) pskmod(bi2de(reshape(bits, 2, []).', 'left-msb'), 4, fase0, '
    gray'); % función mapeo
simHdr     = mapear2sim(bitsHdr);      % header a símbolos
simPay     = mapear2sim(bitsPay);      % payload a símbolos

% Construcción completa de la señal
todosSimbolos = [simPre; simEntrenamiento; simHdr; simPay];  % concatenación final

% Filtro RRC (transmisor)
filtroTX = comm.RaisedCosineTransmitFilter( ...
    'RolloffFactor', rolloff, ...
    'FilterSpanInSymbols', span, ...
    'OutputSamplesPerSymbol', sps);
senalTX = filtroTX([todosSimbolos; zeros(span,1)]);          % filtrado + retardo
senalTX = senalTX ./ max(abs(senalTX));                      % normalización

% Guardar señal en formato .sc16q11
if nargin < 2 || isempty(rutaOutSc16)
    rutaOutSc16 = 'transmision_qpsk.sc16q11';
end
save_sc16q11(rutaOutSc16, senalTX);

% --- Resumen para Command Window ---
tamBytes = 0;
try
    infoOut = dir(rutaOutSc16);
    if ~isempty(infoOut), tamBytes = infoOut.bytes; end
catch
end

% Formato de tamaño legible
if tamBytes >= 2^20
    tamTxt = sprintf('%.2f MB', tamBytes/2^20);
elseif tamBytes >= 2^10
    tamTxt = sprintf('%.2f KB', tamBytes/2^10);
elseif tamBytes > 0
    tamTxt = sprintf('%d bytes', tamBytes);
else
    tamTxt = 'N/D';
end

duracion = toc(t0);

fprintf('\n=== TRANSMISIÓN QPSK COMPLETADA ===\n');
fprintf(' Imagen:          %s\n', rutaImg);
fprintf(' Salida:          %s\n', rutaOutSc16);
fprintf(' Dimensiones:     %dx%dx%d (alto ancho canales)\n', alto, ancho, canales);
fprintf(' Parámetros PHY:  SPS=%d | rolloff=%.2f | span=%d | fase=%.2f rad\n', sps,
    rolloff, span, fase0);
fprintf(' Símbolos:        total=%d | pre=%d | train=%d | hdr=%d | pay=%d\n', ...
```

```
99          numel(todosSimbolos), numel(simPre), numel(simEntrenamiento), numel(simHdr),
                numel(simPay));
100 fprintf(' Muestras TX:     %d (incluye cola del filtro: %d)\n', numel(senalTX), span)
        ;
101 fprintf(' Normalización:   pico=1.00 (aplicada)\n');
102 fprintf(' Archivo SC16:    %s\n', tamTxt);
103 fprintf(' Tiempo total:    %.3f s\n', duracion);
104 fprintf('====================================\n\n');
105
106 % ===== Gráfica de Constelación TX =====
107 try
108     figure('Name','Constelación TX');  % fondo blanco
109     hold on;
110
111     % Muestra solo training + 500 símbolos de payload
112     plot(real(simEntrenamiento), imag(simEntrenamiento), 'ro', 'MarkerFaceColor','r',
            'DisplayName','Entrenamiento');
113     plot(real(simHdr), imag(simHdr), 'bs', 'MarkerFaceColor','b', 'DisplayName','
            Header');
114     plot(real(simPay(1:500)), imag(simPay(1:500)), 'k.', 'DisplayName','Payload');
115
116     axis equal;
117     grid on;
118     xlabel('In-Phase (I)');
119     ylabel('Quadrature (Q)');
120     title('Constelación Transmisor QPSK (  /4, Gray)');
121     legend('Location','northeastoutside');
122     xlim([-1.2 1.2]); ylim([-1.2 1.2]);
123     hold off;
124 catch
125 end
126 end
127
128
129 %% ============================================================
130 % FUNCIONES AUXILIARES
131 %% ============================================================
132
133 function [img, H, W, C] = load_image_uint8_1or3(ruta)
134 info = imfinfo(ruta);
135 [A, mapa, alfa] = imread(ruta); %#ok<ASGLU>
136
137 if ~isempty(mapa)
138     A = ind2rgb(A, mapa);                       % indexado     RGB
139 end
140 if ndims(A) == 3 && size(A,3) == 4
141     A = A(:,:,1:3);                             % elimina canal alfa
142 end
143 if ~isa(A,'uint8')
144     A = im2uint8(A);                            % asegura tipo uint8
145 end
146 if isfield(info,'Orientation')
147     switch info.Orientation
148         case 1
149         case 2, A = fliplr(A);
150         case 3, A = rot90(A,2);
151         case 4, A = flipud(A);
152         case 5, A = rot90(flipud(A),1);
153         case 6, A = rot90(A,-1);
154         case 7, A = rot90(flipud(A),-1);
155         case 8, A = rot90(A,1);
156     end
157 end
```

```
158
159 img = A; H = size(A,1); W = size(A,2); C = size(A,3);
160 if isempty(C), C = 1; end
161 end
162
163
164 function bitsOut = lfsr_scramble(bitsIn, semilla)
165 % Scrambler LFSR de 7 bits (1 + x^4 + x^7)
166 s = de2bi(semilla, 7, 'left-msb'); s = s(:).';
167 bitsOut = false(size(bitsIn));
168 for n = 1:numel(bitsIn)
169     fb = xor(s(4), s(7));               % realimenta s4 y s7
170     bitsOut(n) = xor(bitsIn(n), fb);    % aplica XOR al bit de entrada
171     s = [fb s(1:6)];                    % avanza registro
172 end
173 end
174
175
176 function crc = crc16(bits)
177 % CRC-16-IBM (poly 0xA001, init 0xFFFF)
178 crcReg = uint16(hex2dec('FFFF'));
179 for i = 1:numel(bits)
180     inbit  = logical(bits(i));
181     xorbit = bitget(crcReg,1) ~= inbit;
182     crcReg = bitshift(crcReg, -1);
183     if xorbit
184         crcReg = bitxor(crcReg, hex2dec('A001'));
185     end
186 end
187 crc = de2bi(crcReg, 16, 'left-msb');
188 end
```

2. Código del Receptor (RX QPSK Final)

Listing 13: Código MATLAB para la recepción y reconstrucción de imagen QPSK

```
1  %% ============================================================
2  %% RX QPSK FINAL
3  % Recepción de imagen mediante QPSK (  /4, Gray)
4  %% ============================================================
5
6  % Archivo de entrada por defecto (editar si es necesario)
7  rutaScDefecto = 'P3PruebaFF1.sc16q11';
8  rx_qpsk_final(rutaScDefecto);
9
10
11 %% ============================================================
12 function img = rx_qpsk_final(inSc16)
13 %% ============================================================
14
15 t0 = tic;   % cronómetro simple
16
17 % Respaldo si no se pasa argumento
18 if nargin < 1 || isempty(inSc16)
19     inSc16 = 'anims_test.sc16q11';
20 end
21
22 % Parámetros principales (deben coincidir con TX)
23 sps     = 8;         % muestras por símbolo
24 rolloff = 0.35;      % roll-off del filtro RRC
```

```
span    = 10;       % longitud del filtro (en símbolos)
fase0   = pi/4;     % desplazamiento de fase QPSK

lenBarker = 13;
repBarker = 2;
Lpre      = lenBarker * repBarker;   % longitud total del preámbulo
Ntrain    = 64;                      % símbolos de entrenamiento (QPSK)
HDR_BITS  = 56;                      % 16+16+8+16

% ===== 1) Cargar y normalizar =====
rx = double(load_sc16q11(inSc16));
rx = rx(:);
rx = rx - mean(rx);                 % elimina DC
rx = rx / max(1e-12, rms(rx));      % normaliza en potencia

% ===== Gráfica de la señal real en el tiempo =====  (1/5) [mantener tal cual]
figure('Name','Señal real en el tiempo');
plot(real(rx),'-'); grid on;
xlabel('Índice de muestra'); ylabel('Amplitud (Re\{x[n]\})');
title('Señal recibida - parte real');

% ===== 2) Filtro RRC =====
rrcRx = comm.RaisedCosineReceiveFilter('Shape','Square root', ...
    'RolloffFactor', rolloff, 'FilterSpanInSymbols', span, ...
    'InputSamplesPerSymbol', sps, 'DecimationFactor', 1);
rx_f = rrcRx([rx; zeros(span*sps,1)]);
rx_f = rx_f(span*sps+1:end);

% ===== 3) Sincronía de símbolos (Zero-Crossing) =====
symSync = comm.SymbolSynchronizer( ...
  'TimingErrorDetector','Zero-Crossing (decision-directed)', ...
  'SamplesPerSymbol', sps, 'DampingFactor', 1.0, ...
  'NormalizedLoopBandwidth', 0.006);
rx_sym = symSync(rx_f);  % salida a 1 sps

% ===== 4) Detección del Barker =====
objBarker  = comm.BarkerCode('Length', lenBarker, 'SamplesPerFrame', lenBarker);
bitsBarker = (1 + objBarker())/2;
preBits    = repmat(bitsBarker, repBarker, 1);
simPre     = pskmod(preBits, 2, 0);                        % BPSK
c          = filter(flipud(conj(simPre)), 1, rx_sym);      % correlación
[pk,ix]    = max(abs(c));
startPre   = ix - Lpre + 1;
if startPre < 1
    error('No se detectó el Barker.');
end

% ===== Correlación con Barker =====  (2/5)
try
    figure('Name','Correlación Barker','Color','w');
    Ncorr = min(length(c));                 % muestra razonable
    plot(abs(c(1:Ncorr)),'-'); grid on;
    xlabel('Índice (símbolos)'); ylabel('|correlación|');
    title(sprintf('Correlación con preámbulo Barker (pico @ %d)', startPre));
catch
end

% ===== 5) Corrección de CFO y fase usando el preámbulo =====
z     = rx_sym(startPre:startPre+Lpre-1) .* conj(simPre);
phi   = unwrap(angle(z(:))).';
k     = 0:Lpre-1;
p     = polyfit(k, phi, 1);
alpha = p(1);
```

```
beta  = p(2);
m     = (0:length(rx_sym)-startPre).';
rx_fix = rx_sym(startPre:end) .* exp(-1j*(alpha*m + beta));

% ===== 6) Refinamiento CFO/fase con entrenamiento QPSK =====
patron16         = pskmod([0;1;2;3; 1;0;3;2; 2;3;0;1; 3;2;1;0], 4, fase0, 'gray');
simEntrenamiento = repmat(patron16, 4, 1);
rEntrenamiento   = rx_fix(Lpre + (1:Ntrain));

best = struct('var', inf, 'conj', false, 'a', 0, 'b', 0);
for cc = [false true]
    t  = rEntrenamiento;
    if cc, t = conj(t); end
    ph = unwrap(angle(t .* conj(simEntrenamiento)));
    k  = 0:Ntrain-1;
    p  = polyfit(k, ph.', 1);
    v  = var(ph.' - polyval(p, k));                % varianza como métrica
    if v < best.var
        best = struct('var', v, 'conj', cc, 'a', p(1), 'b', p(2));
    end
end

rDespues = rx_fix(Lpre+Ntrain+1:end);
if best.conj, rDespues = conj(rDespues); end
n = (0:numel(rDespues)-1).';
rDespues = rDespues .* exp(-1j*(best.a*n + best.b));

% ===== Constelación antes de corrección (CFO/fase) =====  (3/5)
try
    figure('Name','Constelación - antes de corrección','Color','w');
    i0 = max(1, startPre-4000);
    i1 = min(length(rx_sym), startPre+Lpre+4000);
    plot(real(rx_sym(i0:i1)), imag(rx_sym(i0:i1)), '.');
    axis equal; grid on;
    xlabel('I'); ylabel('Q');
    title('Antes de corrección (alrededor del preámbulo)');
catch
end

% ===== Constelación después de refinamiento (CFO/fase) =====  (4/5)
try
    figure('Name','Constelación - después de refinamiento','Color','w');
    M = min(12000, numel(rDespues));
    plot(real(rDespues(1:M)), imag(rDespues(1:M)), '.');
    axis equal; grid on;
    xlabel('I'); ylabel('Q');
    title('Después de refinamiento (CFO/fase)');
catch
end

% ===== 7) Lectura del header {rotación, slip} =====
segN = min(numel(rDespues), 200000);
phiC = [0, pi/2, pi, 3*pi/2];
slipC = 0:3;
ok=false; bestH=0; bestW=0; bestC=0; rotSel=0; slipSel=0;

for ss = slipC
    if 1+ss > segN, continue; end
    s = rDespues(1+ss:segN);
    for rr = 1:numel(phiC)
        sy   = s .* exp(-1j*phiC(rr));
        b    = pskdemod(sy, 4, fase0, 'gray', 'OutputType', 'bit');
        bits = b(:).';
```

```
151         if numel(bits) < HDR_BITS, continue; end
152         hdr  = bits(1:HDR_BITS);
153         H    = bi2de(hdr(1:16) ,'left-msb');
154         W    = bi2de(hdr(17:32),'left-msb');
155         C    = bi2de(hdr(33:40),'left-msb');
156         okcrc= isequal(crc16(hdr(1:40)), hdr(41:56));
157         if okcrc && H>=1 && H<=8192 && W>=1 && W<=8192 && ismember(C,[1,3])
158             ok=true; bestH=H; bestW=W; bestC=C; rotSel=phiC(rr); slipSel=ss; break;
159         end
160     end
161     if ok, break; end
162 end
163
164 if ~ok
165     error('Cabecera inválida o no detectada.');
166 end
167
168 % ===== 8) Alineamiento del payload =====
169 s_all = rDespues(1+slipSel:end) .* exp(-1j*rotSel);
170
171 % ===== 9) PLL + ajuste final de fase (snap   /4) =====
172 carSync2 = comm.CarrierSynchronizer('Modulation','QPSK', ...
173   'ModulationPhaseOffset','Custom','CustomPhaseOffset', fase0, ...
174   'SamplesPerSymbol', 1, 'DampingFactor', 0.707, ...
175   'NormalizedLoopBandwidth', 0.001);
176 s_all = carSync2(s_all);
177
178 K = min(3000, numel(s_all));
179 phi_meas = angle(mean(s_all(1:K).^4))/4;
180 kpi2 = round((phi_meas - fase0)/(pi/2));
181 phi_fix = phi_meas - (fase0 + kpi2*(pi/2));
182 s_all = s_all .* exp(-1j*phi_fix);
183
184 % ===== 10) Demodulación, descramble y reconstrucción de imagen =====
185 b_all   = pskdemod(s_all, 4, fase0, 'gray', 'OutputType', 'bit');
186 b_all   = b_all(:).';
187 imgBits = b_all(HDR_BITS+1:end);
188 imgBits = lfsr_descramble(imgBits, 127);
189
190 needBytes = double(bestH)*double(bestW)*double(max(1,bestC));
191 needBits  = needBytes*8;
192
193 if numel(imgBits) < needBits
194     imgBits(end+1:needBits) = 0;
195 else
196     imgBits = imgBits(1:needBits);
197 end
198
199 bytes = uint8(bi2de(reshape(imgBits,8,[]).','left-msb'));
200
201 if bestC == 1
202     img = reshape(bytes,[bestH bestW]);
203 else
204     img = reshape(bytes,[bestH bestW bestC]);
205 end
206
207 % ===== Resumen en consola (práctica 3) =====
208 total_sym = Lpre + Ntrain + numel(s_all);   % aprox. símbolos procesados
209
210 fprintf('\n=== RECEPCIÓN QPSK ===\n');
211 fprintf(' Archivo:        %s\n', inSc16);
212 fprintf(' Imagen:         %dx%dx%d (alto ancho canales)\n', bestH, bestW, bestC);
```

```
fprintf(' PHY:            SPS=%d | rolloff=%.2f | span=%d | fase=%.2f rad\n', sps,
    rolloff, span, fase0);
fprintf(' Preambulo:      start=%d | métrica=%.2f\n', startPre, pk);
fprintf(' CFO/fase:       alpha=%.3e rad/sym | beta=%.3f rad\n', alpha, beta);
fprintf(' Ajustes:        conj=%d | rot=%.2f rad | slip=%d\n', best.conj, rotSel,
    slipSel);
fprintf(' Símbolos:       pre=%d | train=%d | total≈%d\n', Lpre, Ntrain, total_sym);
fprintf(' Muestras RX:    %d\n', numel(rx));
fprintf(' Datos:          HDR=%d bits | payload=%d bytes\n', HDR_BITS, numel(bytes));
fprintf(' Tiempo:         %.3f s\n', toc(t0));
fprintf('======================\n\n');

% ===== Mostrar resultados =====
figure('Name','Imagen QPSK Reconstruida en Rx');
imshow(img);
title(sprintf('Imagen QPSK Reconstruida en Rx (%dx%dx%d)', bestH, bestW, bestC));

% ===== Constelación payload (final) =====  (5/5)
try
    figure('Name','Constelación - payload final','Color','w');
    M = min(16000, numel(s_all));
    plot(real(s_all(1:M)), imag(s_all(1:M)), '.');
    axis equal; grid on;
    xlabel('I'); ylabel('Q');
    title('Payload tras PLL + snap   /4');
catch
end
end


% ===== Funciones auxiliares =====
function bitsOut = lfsr_descramble(bitsIn, seed)
s = de2bi(seed,7,'left-msb');
s = s(:).';
bitsOut = false(size(bitsIn));
for n = 1:numel(bitsIn)
    fb = xor(s(4), s(7));
    bitsOut(n) = xor(bitsIn(n), fb);
    s = [fb s(1:6)];
end
end

function crc = crc16(bits)
crcReg = uint16(hex2dec('FFFF'));
for i = 1:numel(bits)
  inbit  = logical(bits(i));
  xorbit = bitget(crcReg,1) ~= inbit;
  crcReg = bitshift(crcReg, -1);
  if xorbit, crcReg = bitxor(crcReg, hex2dec('A001')); end
end
crc = de2bi(crcReg,16,'left-msb');
end
```

3. Resultados del Command Window

```
=== TRANSMISIÓN QPSK COMPLETADA ===
 Imagen:          nota20.png
 Salida:          P3PruebaFF1.sc16q11
 Dimensiones:     594x1287x3 (alto×ancho×canales)
```

```
 Parámetros PHY:  SPS=8 | rolloff=0.35 | span=10 | fase=0.79 rad
 Símbolos:        total=9173854 | pre=26 | train=64 | hdr=28 | pay=9173736
 Muestras TX:     73390912 (incluye cola del filtro: 10)
 Normalización:   pico=1.00 (aplicada)
 Archivo SC16:    279.96 MB
 Tiempo total:    31.188 s
==================================

=== RECEPCIÓN QPSK ===
 Archivo:         P3PruebaFF1.sc16q11
 Imagen:          594x1287x3 (alto×ancho×canales)
 PHY:             SPS=8 | rolloff=0.35 | span=10 | fase=0.79 rad
 Preambulo:       start=859455 | métrica=85.40
 CFO/fase:        alpha=-3.038e-03 rad/sym | beta=0.780 rad
 Ajustes:         conj=0 | rot=0.00 rad | slip=0
 Símbolos:        pre=26 | train=64 | total9744821
 Muestras RX:     84836352
 Datos:           HDR=56 bits | payload=2293434 bytes
 Tiempo:          44.373 s
======================
```

4. Configuración en BladeRF-CLI

```
# Transmisor (TX)
set frequency tx1 920M; set samplerate tx1 2M; set bandwidth tx1 2M
set agc off; set gain tx1 50
tx config file=P3PruebaFF1.sc16q11 format=bin repeat=1
tx start; tx wait

# Receptor (RX)
set frequency rx1 920M; set samplerate rx1 2M; set bandwidth rx1 2M
set agc off; set gain rx1 15
rx config file=P3PruebaFF1.sc16q11 format=bin n=0
tx start
tx stop
```

B. Drive y GitHub

En el siguiente enlace se encuentra disponible la carpeta de respaldo en Google Drive, que contiene todos los archivos generados durante la práctica, incluyendo los códigos fuente en MATLAB, las imágenes transmitidas y recibidas.

- **Enlace:** https://drive.google.com/drive/folders/1syCcmkbR46iKqC4l20rt-UAEfdUsrzyQ?usp=drive_link

En este se incluye el código fuente completo del transmisor y receptor, las funciones auxiliares, configuraciones del SDR BladeRF, resultados experimentales, y el documento final en formato IEEE. Este repositorio permite la revisión del desarrollo y facilita la replicación del sistema.

- **Repositorio:** https://github.com/RonyTicona1/sistema_tx_rx_imagen_qpsk_blade_rf_sdr