Trabajo Fin de Grado
en Ingeniería Aeroespacial

Implementación de un Sistema de Radio Definida por Software (SDR) basado en ADALM Pluto SDR para la Transmisión y Recepción de Modulaciones Digitales

Autor: Alejandro Madero Vílchez

Tutor: Juan Antonio Becerra González

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2025

Trabajo Fin de Grado
en Ingeniería Aeroespacial

Implementación de un Sistema de Radio Definida por Software (SDR) basado en ADALM Pluto SDR para la Transmisión y Recepción de Modulaciones Digitales

Autor:

Alejandro Madero Vílchez

Tutor:

Juan Antonio Becerra González

Profesor Titular

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2025

Trabajo Fin de Grado: Implementación de un Sistema de Radio Definida por Software (SDR) basado en ADALM Pluto SDR para la Transmisión y Recepción de Modulaciones Digitales

Autor: Alejandro Madero Vílchez
Tutor: Juan Antonio Becerra González

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Gracias a toda la gente tan maravillosa que he conocido en el camino. En especial a Patri, Aarón, Andrea, Vicky, Antonio, Clara, Vania y a todos los que habéis estado ahí, antes o después. Me llevo una etapa muy bonita a vuestro lado, y sin duda, no habría sido tan amena ni especial sin vuestra compañía.

Gracias Marta y Lucía, por ser mis pilares, las amistades de siempre, las que siempre están y las que siempre estarán, aunque la distancia no esté a nuestro favor.

Gracias infinitas a mi tutor, Juan Antonio. Has hecho que este proyecto parezca pan comido. Gracias por tu paciencia, por saber ver siempre el lado positivo incluso en medio del caos, y por enseñar con tanta claridad y cercanía. Gracias, de corazón, por enseñar como enseñas y por ser como eres.

Como siempre me decís vosotros, mamá y papá: "Con paciencia se sacan las cosas. Al final, ya verás cómo todo se consigue". Gracias a toda mi familia por estar siempre apoyándome en todo lo que he hecho, por celebrar conmigo cada logro, por aguantarme en mis momentos más difíciles, por estar siempre ahí cuando os necesito, y por enseñarme que, por mucho que crezcamos, nunca dejamos de aprender.

A todos los que habéis formado parte de este camino. Y a ti también, lector, gracias por hacerlo inolvidable.

Alejandro Madero Vílchez
Estudiante del Grado en Ingeniería Aeroespacial

Sevilla, 2025

Resumen

Este trabajo de fin de grado aborda la implementación de un sistema de radio definida por software (SDR) utilizando el dispositivo ADALM Pluto SDR. El objetivo principal es explorar la transmisión y recepción de modulaciones digitales, sentando las bases para futuras aplicaciones en la detección, reconocimiento e identificación de drones.

El documento comienza presentando los conceptos fundamentales de la radio definida por software y el hardware ADALM Pluto SDR, detallando sus capacidades y arquitectura. Se describen los pasos para la configuración del entorno de desarrollo, incluyendo la instalación del software necesario y las librerías de programación. Se implementan, además, ejemplos de modulaciones como PAM, PSK y QAM, explicando el proceso de generación de la señal en banda base, la modulación digital y la transmisión a través del ADALM Pluto SDR. Paralelamente, se abordan los sistemas de recepción, incluyendo la captura de la señal recibida, la demodulación y el procesamiento para recuperar la información original.

Se presentan y analizan los resultados de los experimentos realizados, comparando las señales transmitidas y recibidas en términos de diagramas de constelación, espectros de frecuencia y otras métricas relevantes para evaluar el rendimiento del sistema implementado. Se discuten los desafíos encontrados y las soluciones aplicadas durante el proceso de implementación.

Finalmente, se plantean diversas líneas futuras para la continuación de este trabajo, destacando la aplicación de la plataforma SDR y los conocimientos adquiridos a la detección, reconocimiento e identificación de drones.

En resumen, este proyecto demuestra la viabilidad de utilizar una plataforma SDR de bajo coste como el ADALM Pluto SDR para implementar sistemas de comunicación digital.

Abstract

This project addresses the implementation of a software-defined radio (SDR) system using the ADALM Pluto SDR device. The main objective is to explore the transmission and reception of digital modulations, laying the groundwork for future applications in drone detection, recognition, and identification.

The document begins by presenting the fundamental concepts of software-defined radio and the ADALM Pluto SDR hardware, detailing its capabilities and architecture. It describes the steps involved in setting up the development environment, including the installation of the required software and programming libraries. Examples of modulations such as PAM, PSK, and QAM are implemented, with explanations of the baseband signal generation process, digital modulation, and transmission through the ADALM Pluto SDR. In parallel, reception systems are addressed, including signal capture, demodulation, and processing to recover the original information.

The results of the experiments conducted are presented and analyzed, comparing the transmitted and received signals in terms of constellation diagrams, frequency spectra, and other relevant metrics to evaluate the performance of the implemented system. The challenges encountered and the solutions applied during the implementation process are also discussed.

Finally, several future lines of work are proposed, highlighting the application of the SDR platform and the knowledge acquired to the detection, recognition, and identification of drones.

In summary, this project demonstrates the feasibility of using a low-cost SDR platform such as the ADALM Pluto SDR to implement digital communication systems.

Índice Abreviado

Índice

Glosario

ADC	(Convertidor Analógico-Digital)
AD936x	(Transceptor RF de Analog Devices)
AGC	(Control Automático de Ganancia)
AM	(Modulación de Amplitud)
API	(Interfaz de Programación de Aplicaciones)
AP	(Punto de Acceso)
ARM Cortex-A9	(Tipo de procesador)
AWGN	(Ruido Blanco Gaussiano Aditivo)
BER	(Tasa de Error de Bit)
BEREC	(Organismo de Reguladores Europeos de las Comunicaciones Electrónicas)
C	(Lenguaje de programación)
C++	(Lenguaje de programación)
C#	(Lenguaje de programación)
CE	(Comisión Europea)
CNAF	(Cuadro Nacional de Atribución de Frecuencias)
DAC	(Convertidor Digital-Analógico)
DDC	(Conversión Digital Descendente)
DFT	(Transformada Discreta de Fourier)
DRI	(Detección, Reconocimiento e Identificación)
DSB	(Doble Banda Lateral)
DSP	(Procesamiento Digital de Señales)
DUC	(Conversión Digital Ascendente)
EHF	(Frecuencia Extremadamente Alta)
ENISA	(Agencia de la Unión Europea para la Ciberseguridad)
ESP8266	(Microcontrolador con WiFi)
ETD	(Orden Ministerial Española)
EVM	(Magnitud del Vector de Error)
FCC	(Comisión Federal de Comunicaciones de EEUU)
FFT	(Transformada Rápida de Fourier)
FI	(Frecuencia Intermedia)
FM	(Modulación de Frecuencia)
FPGA	(Field-Programmable Gate Array)
GHz	(Gigahertzio)
HF	(Alta Frecuencia)
Hz	(Hertzio)
I/Q	(En Fase / En Cuadratura)
IP	(Protocolo de Internet)
ISI	(Interferencia Intersimbólica)
kHz	(Kilohertzio)

LF	(Baja Frecuencia)
LGT	(Ley General de Telecomunicaciones)
LO	(Oscilador Local)
LTE	(Long-Term Evolution - estándar de comunicación móvil)
MF	(Frecuencia Media)
MHz	(Megahertzio)
MSE	(Error Cuadrático Medio)
NMSE	(Error Cuadrático Medio Normalizado)
PAM	(Modulación por Amplitud de Pulso)
PCM	(Modulación por Código de Pulsos)
PSK	(Modulación por Desplazamiento de Fase)
PSD	(Densidad Espectral de Potencia)
QAM	(Modulación por Amplitud en Cuadratura)
RF	(Radiofrecuencia)
RFSoC	(Radio Frequency System-on-Chip)
RX	(Recepción)
SDR	(Radio Definida por Software)
SHF	(Frecuencia Súper Alta)
SMA	(Tipo de conector)
SNR	(Relación Señal-Ruido)
SoC	(System-on-Chip)
SSB	(Banda Lateral Única)
TFG	(Trabajo Fin de Grado)
TV	(Televisión)
TX	(Transmisión)
UE	(Unión Europea)
UHF	(Frecuencia Ultra Alta)
UIT	(Unión Internacional de Telecomunicaciones)
URI	(Uniform Resource Identifier)
USB	(Universal Serial Bus)
VHF	(Frecuencia Muy Alta)
VLF	(Frecuencia Muy Baja)
WELCH	(Método de Welch para estimación espectral)
WiFi	(Wireless Fidelity)
Zynq UltraScale+ RF-SoC	(Plataforma SoC de Xilinx)
2G, 3G, 4G, 5G	(Generaciones de tecnología móvil)

1 Introducción

Esta sección presenta los fundamentos y la motivación detrás del desarrollo de un sistema de radio definida por software (del inglés, *Software Defined Radio*, SDR) para la Detección, Reconocimiento e Identificación (DRI) de señales en drones, así como los objetivos que guían esta investigación.

1.1 Contexto y motivación

En el proceso de tecnología, muchas industrias se hacen cargo de los drones, desde la agricultura hasta la seguridad. Este avance subraya la necesidad de tener sistemas de control y comunicación más complejos.

Los sistemas basados en todos los sistemas analíticos y de vigilancia tradicionales no pueden abordar las diversas modulaciones y frecuencias utilizadas en drones eléctricos. Es por ello que podemos ver que el uso de radio definida por software es una de las soluciones posibles para mitigar estos defectos, y que su adaptabilidad a la dinámica eficiente proporciona una variedad de esquemas de frecuencia y modulación. El uso de SDR en los sistemas de reconocimiento y la detección en drones permite una mayor flexibilidad y precisión en el análisis de señales variadas.

Por lo tanto, el desarrollo de sistemas SDR centrados en la detección, detección e identificación de drones no solo responderá a los desafíos actuales, sino que también proporcionará participación en el futuro de la tecnología de drones en aplicaciones clave en áreas, como es el ámbito de la seguridad.

1.2 Objetivos planteados

El objetivo principal de este trabajo es analizar y comprender algunas de las distintas modulaciones empleadas en la transmisión de datos. Para ello, se estudiarán los aspectos técnicos que caracterizan estas modulaciones, así como su comportamiento y eficiencia usando el dispositivo ADALM Pluto SDR.

Además, se realizará una comprobación experimental de la captación de la señal WiFi por parte del dispositivo, generada por una placa ESP8266 12-F. Esto permitirá evaluar la capacidad del Pluto para detectar cambios en el inicio y apagado de la señal WiFi.

El trabajo pretende, en conjunto, ofrecer una visión clara y aplicada de las tecnologías de modulación, hablando finalmente de una de las implementaciones aeronáuticas.

1.3 Material empleado

Para llevar a cabo el desarrollo de este Trabajo de Fin de Grado (TFG), se han utilizado varios materiales y dispositivos electrónicos. A continuación, se exponen los componentes empleados:

- **ADALM Pluto SDR (2 unidades)**: Se ha utilizado dos unidades de los dispositivos ADALM Pluto SDR, los cuales son utilizados para la transmisión y recepción de señales. Cada dispositivo se conecta al ordenador a través de un cable micro USB a USB, permitiendo el control y la configuración mediante software. Los dispositivos incluyen antenas para la recepción y transmisión de señales. A su vez, se empleó un cable especial para cortocircuitar las conexiones de transmisión (TX) y recepción (RX) de los dispositivos ADALM Pluto SDR. Este cable se utilizó para cortocircuitar un mismo dispositivo, y para entrelazar TX y RX de dos diferentes.

- **ESP8266 12-F**: Este módulo WiFi ha sido utilizado para la conmutación y control de las conexiones de red WiFi. El ESP8266 12-F es un microcontrolador con capacidad WiFi integrado, el cual se conecta a un ordenador mediante un cable micro USB a USB, facilitando la programación y la conexión con otros dispositivos a través de redes inalámbricas con ayuda de la interfaz Arduino.
- **Cable micro USB a USB**: Se utilizaron cables micro USB a USB para establecer la conexión entre los dispositivos mencionados (ADALM Pluto SDR y ESP8266) y el ordenador.

1.4 Estructura

El presente trabajo se organiza en seis capítulos, estructurados de forma que facilite la comprensión progresiva de los conceptos y resultados alcanzados:

- **Capítulo 1:** Se introducen los conceptos básicos de este trabajo, su relevancia en el contexto actual y la motivación del proyecto. También se plantean los objetivos generales del trabajo.
- **Capítulo 2:** Se abordan los fundamentos teóricos necesarios para entender las modulaciones digitales más comunes, incluyendo sus propiedades espectrales y características clave, así como el marco legal relacionado con la transmisión de señales, tanto a nivel europeo como a nivel nacional.
- **Capítulo 3:** Se presenta el dispositivo SDR utilizado y sus principales características. Se describen también los entornos de programación empleados.
- **Capítulo 4:** Se documentan los distintos experimentos de transmisión y recepción realizados con el dispositivo, aplicando las modulaciones PAM, PSK y QAM.
- **Capítulo 5:** Se realiza un análisis detallado de los resultados experimentales, comparando el comportamiento observado con el esperado teóricamente. Se destacan las conclusiones obtenidas en cuanto al rendimiento del sistema y las posibles líneas de mejora futura.
- **Capítulo 6:** Se destacan las conclusiones obtenidas en cuanto al rendimiento del sistema a alto nivel y las posibles líneas de mejora y trabajo futuro.

2 Fundamentos teóricos de comunicaciones digitales

Las señales son funciones que representan información y varían en el tiempo u otras variables. Su análisis matemático permite describir propiedades como periodicidad, energía, frecuencia y simetría. Estos fundamentos son esenciales para el estudio y diseño de sistemas de comunicación, procesamiento digital y control, utilizando herramientas como el análisis de Fourier, muestreo y transformadas.

2.1 Procesado de señales

2.1.1 Señales exponenciales complejas y señales senoidales

Atendiendo a [1], una señal exponencial compleja en tiempo continuo es de la forma:

$$x(t) = Ce^{at} \quad , \tag{2.1}$$

donde C y a son, en general, números complejos. Una clase importante de exponenciales complejas se obtiene cuando a es puramente imaginario:

$$a = j\omega_0 \Rightarrow x(t) = Ce^{j\omega_0 t} \quad . \tag{2.2}$$

Esta señal es periódica. El periodo fundamental T_0 se obtiene como:

$$T_0 = \frac{2\pi}{|\omega_0|} \quad . \tag{2.3}$$

Una señal relacionada es la senoidal:

$$x(t) = A\cos(\omega_0 t + \phi) \quad . \tag{2.4}$$

También es común escribir $\omega_0 = 2\pi f_0$, donde f_0 tiene unidades de hertz (Hz). La señal senoidal es periódica con el mismo periodo T_0 que la exponencial compleja.

2.1.2 Señal portadora y modulación senoidal

En muchos sistemas de comunicación se emplea la modulación senoidal de amplitud, donde una señal $c(t)$ (portadora) se modula en amplitud por una señal $x(t)$ (información o mensaje):

$$y(t) = x(t) \cdot c(t) \quad . \tag{2.5}$$

La señal $x(t)$ se llama señal moduladora y $c(t)$ es la **señal portadora**. La modulación tiene como objetivo generar una señal cuya banda de frecuencias se adapte al canal de transmisión. Por ejemplo:

- Señal de voz: 200 Hz – 4 kHz
- Enlace satelital: 500 MHz – 40 GHz

Así, la portadora permite desplazar el espectro de $x(t)$ a una frecuencia central alta como f_c (frecuencia de la portadora), logrando una señal adecuada para transmisión.

2.1.3 Representación general de señales moduladas

Cualquier señal senoidal puede expresarse como combinación de exponenciales complejas:

$$\cos(\omega_0 t + \phi) = \frac{1}{2}\left(e^{j(\omega_0 t+\phi)} + e^{-j(\omega_0 t+\phi)}\right) \quad . \tag{2.6}$$

Esta relación es útil para el análisis espectral de señales moduladas, permitiendo usar transformadas de Fourier para caracterizar desplazamientos espectrales. Sin embargo, el tema de la modulación será tratado más en profundidad en la *Sección 2.3*.

2.1.4 Parámetros fundamentales de una señal

Periodo de muestreo (sampling period)

El proceso de muestreo por tren de impulsos, según [4], consiste en multiplicar una señal continua $x(t)$ por una función de impulsos periódica $p(t)$. Esta se define como:

$$p(t) = \sum_{n=-\infty}^{+\infty} \delta(t - nT) \quad , \tag{2.7}$$

donde T es llamado el **periodo de muestreo** y representa el tiempo entre dos deltas de Dirac consecutivas.

Frecuencia de muestreo (sampling frequency)

La frecuencia fundamental del tren de impulsos $p(t)$ es:

$$\omega_s = \frac{2\pi}{T} \quad . \tag{2.8}$$

Se conoce como la **frecuencia de muestreo** y se expresa en radianes por segundo.
En Hz (ciclos por segundo):

$$f_s = \frac{1}{T} \quad . \tag{2.9}$$

Periodo de bit (bit period)

En un sistema binario, cada bit ocupa un cierto tiempo llamado **periodo de bit**:

$$T_b = \frac{1}{R_b} \quad , \tag{2.10}$$

donde R_b es la **tasa binaria** o **bit rate** (bits por segundo). Cada muestra $x[n]$ se codifica como una secuencia de bits, y cada bit se transmite en un intervalo de tiempo.

Tiempo de símbolo (symbol time)

Cuando un símbolo representa más de un bit, su duración es mayor. El **tiempo de símbolo** se define como:

$$T_s = \frac{1}{f_s} \quad , \tag{2.11}$$

donde f_s es la frecuencia de símbolos (número de símbolos por segundo). Está asociado directamente al tiempo de transmisión de una unidad codificada.

Frecuencia de portadora o frecuencia central

En la modulación con una señal senoidal o exponencial compleja:

$$c(t) = \cos(\omega_c t + \phi) \quad \text{ó} \quad c(t) = e^{j\omega_c t} \quad , \tag{2.12}$$

donde ω_c es la **frecuencia angular de la portadora**. La frecuencia en Hz es:

$$f_c = \frac{\omega_c}{2\pi} \quad . \tag{2.13}$$

Esta es conocida como la **frecuencia de portadora** o **frecuencia central**, pues es el centro del espectro de la señal modulada. En frecuencia angular:

- ω_c: frecuencia de la portadora en rad/s.
- f_c: frecuencia de la portadora en Hz.

La modulación por una portadora senoidal desplaza el espectro de la señal original $x(t)$ a una banda centrada en ω_c.

2.1.5 Muestreo de señales. Teorema de Nyquist

Teorema de muestreo. Frecuencia de Nyquist

Según [3], sea $x(t)$ una señal limitada en banda con $X(j\omega) = 0$ para $|\omega| > \omega_M$. Entonces $x(t)$ está determinada de manera única por sus muestras $x(nT)$, $n = 0, \pm 1, \pm 2, \ldots$ si

$$\omega_s > 2\omega_M \quad \text{, tal que} \tag{2.14}$$

$$\omega_s = 2\pi f_s \qquad \text{y} \qquad f_s = \frac{1}{T} \quad . \tag{2.15}$$

Dadas estas muestras, podemos reconstruir $x(t)$ generando un tren de impulsos periódico en el que impulsos sucesivos tienen amplitudes iguales a los valores sucesivos de las muestras. Este tren de impulsos se procesa luego mediante un filtro de paso bajo ideal con ganancia T y frecuencia de corte mayor que ω_M y menor que $\omega_s - \omega_M$. La señal de salida resultante será exactamente igual a $x(t)$.

La frecuencia $2\omega_M$, que bajo el teorema de muestreo debe ser excedida por la frecuencia de muestreo, es comúnmente llamada la **tasa de Nyquist**. La frecuencia ω_M correspondiente a la mitad de la tasa de Nyquist a menudo se denomina **frecuencia de Nyquist**.

Consideraciones prácticas

Los filtros ideales generalmente no se utilizan en la práctica por diversas razones. En cualquier aplicación práctica, el filtro pasa bajas ideal se reemplaza por un filtro no ideal $H(j\omega)$ que aproxima la característica de frecuencia deseada con la precisión suficiente para el problema en cuestión (es decir, $H(j\omega) = 1$ para $|\omega| < \omega_M$ y $H(j\omega) = 0$ para $|\omega| > \omega_s - \omega_M$). Obviamente, cualquier aproximación de este tipo en la etapa de filtrado paso bajo conducirá a alguna discrepancia entre $x(t)$ y $x_r(t)$, o equivalentemente, entre $X(j\omega)$ y $X_r(j\omega)$. La elección particular del filtro no ideal se dicta entonces por el nivel aceptable de distorsión para la aplicación bajo consideración.

Para mayor conveniencia y para enfatizar principios básicos como el teorema de muestreo, asumiremos regularmente la disponibilidad y uso de filtros ideales, con el entendimiento de que en la práctica tal filtro debe ser reemplazado por uno no ideal diseñado para aproximar con suficiente precisión las características ideales para el problema en cuestión.

2.1.6 Sincronización de señales. Correlación

Correlación de señales en tiempo discreto

Sea $x[n]$ y $y[n]$ dos señales de tiempo discreto reales. Las funciones de **autocorrelación** $\phi_{xx}[n]$ y $\phi_{yy}[n]$ están definidas como:

$$\phi_{xx}[n] = \sum_{m=-\infty}^{+\infty} x[m+n]x[m] \quad , \tag{2.16}$$

$$\phi_{yy}[n] = \sum_{m=-\infty}^{+\infty} y[m+n]y[m] \quad . \tag{2.17}$$

Las **funciones de correlación cruzada** son:

$$\phi_{xy}[n] = \sum_{m=-\infty}^{+\infty} x[m+n]y[m] \quad , \tag{2.18}$$

$$\phi_{yx}[n] = \sum_{m=-\infty}^{+\infty} y[m+n]x[m] \quad . \tag{2.19}$$

Propiedades:

- $\phi_{xx}[n]$ y $\phi_{yy}[n]$ son funciones pares.
- $\phi_{xy}[n] = \phi_{yx}[-n]$

Correlación de señales en tiempo continuo

Dadas dos señales reales de tiempo continuo $x(t)$ y $y(t)$, la **función de correlación cruzada** está definida como:

$$\phi_{xy}(t) = \int_{-\infty}^{+\infty} x(t+\tau)y(\tau)\,d\tau \quad . \tag{2.20}$$

La **función de autocorrelación** se obtiene tomando $y(t) = x(t)$:

$$\phi_{xx}(t) = \int_{-\infty}^{+\infty} x(t+\tau)x(\tau)\,d\tau \quad . \tag{2.21}$$

El filtro con respuesta al impulso igual a la señal $x(t)$ invertida en el tiempo es llamado **filtro adaptado**. Produce una salida máxima cuando la señal de entrada coincide con aquella para la cual fue diseñado:

- Si una señal $x(t)$ de duración finita es entrada a un sistema LTI con $h(t) = x(T-t)$, entonces la salida es la autocorrelación desplazada: $\phi_{xx}(T-t)$.
- Este filtro maximiza la salida $y(T)$ bajo la restricción de energía del filtro.

Correlación en el dominio de la frecuencia

Sean $x(t)$ e $y(t)$ dos señales reales. Sea $\phi_{xy}(t)$ su correlación cruzada. Definimos las transformadas de Fourier como:

$$\Phi_{xy}(j\omega) = \mathscr{F}\left\{\phi_{xy}(t)\right\} \quad . \tag{2.22}$$

Entonces:

- $\Phi_{xy}(j\omega) = X(j\omega)Y^*(j\omega)$
- $\Phi_{xx}(j\omega)$ es **real y no negativa** para toda ω.

Si $x(t)$ es entrada a un sistema LTI con respuesta al impulso $h(t)$ y respuesta en frecuencia $H(j\omega)$, y la salida es $y(t)$, entonces:

$$\Phi_{xy}(j\omega) = \Phi_{xx}(j\omega)H^*(j\omega) \quad , \tag{2.23}$$

$$\Phi_{yy}(j\omega) = |H(j\omega)|^2\Phi_{xx}(j\omega) \quad . \tag{2.24}$$

En la modulación y demodulación AM, la relación de fase entre los osciladores del modulador y del demodulador es crítica. Si las fases están desincronizadas, la salida se ve afectada:

- Si la diferencia de fase es $\theta_c - \phi_c = \frac{\pi}{2}$, entonces la salida es cero.
- Para salida máxima, las fases deben estar sincronizadas: $\theta_c = \phi_c$.

Demodulación asincrónica: se usa cuando no se puede garantizar la sincronización. Se requiere que $x(t)$ sea positiva y lenta comparada con w_c (frecuencia de la portadora), para permitir la recuperación a través de un detector de envolvente.

Aplicaciones en radar y comunicación binaria

- En radar, se transmite un pulso $p(t)$. La señal reflejada $x(t) = ap(t - t_0)$ tiene su **máxima correlación cruzada** con $p(t)$ en $t = t_0$, lo que permite estimar el retardo y por tanto la distancia.
- En comunicación binaria, diferentes bits son transmitidos como distintas señales, y filtros adaptados permiten detectar cuál fue recibida comparando salidas máximas de correlación.

2.2 Estimación de la densidad espectral de potencia

El procesamiento de espectros de señales es una de técnicas claves en cuanto a utilidad en el análisis de señales y sistemas, que proporciona información importante sobre la frecuencia, el rendimiento y las características del sistema. Este enfoque presenta el método como un instrumento clave para la transformación de la Transformada Rápida de Fourier (del inglés, *Fast Fourier Transform* FFT) y la estimación espectral. Cada uno tiene sus propias características y ventajas especiales.

En relación a [6], la FFT es una implementación eficiente de los algoritmos de transformación discreta (DFT) de Fourier, lo que le permite analizar las frecuencias presentes en señales digitales. Este procedimiento revolucionó el procesamiento de señales en tiempo real al reducir el tiempo requerido. El método de Welch, por otro lado, es un método estadístico, que permite estimar con mayor precisión el espectro de potencia de la señal, especialmente cuando esta es ruidosa o sujeta a variación. Este enfoque se basa en el dividir la señal en segmentos, el uso de ventanas para cada uno y para reducir los errores de estimación y mejorar la resolución espectral.

En ambos sentidos, FFT y los algoritmos de Welch son fundamentalmente importantes para la interpretación y el análisis de señales, proporcionando medios precisos para la investigación del comportamiento de frecuencia de señales en varios contextos.

2.2.1 Transformada de Fourier (FFT). Relación de la FFT con la DFT

Aunque se han desarrollado muchos algoritmos diferentes de FFT, el algoritmo **radix-2** es especialmente eficiente para realizar DFTs cuando el número de puntos es una potencia de dos, es decir, $N = 2^k$, donde k es un entero positivo.

La DFT directa está dada por:

$$X(m) = \sum_{n=0}^{N-1} x(n) \cdot e^{-j\frac{2\pi}{N}nm} \quad . \tag{2.25}$$

Esto requiere N^2 multiplicaciones complejas. Por ejemplo, para $N = 8$, se necesitan 64 multiplicaciones. En cambio, la FFT radix-2 reduce esto a:

$$\frac{N}{2}\log_2 N \quad \text{(para } N = 8\text{, solo 12 multiplicaciones).} \tag{2.26}$$

Desarrollo algebraico del algoritmo FFT Radix-2

Partimos de la DFT:

$$X(m) = \sum_{n=0}^{N-1} x(n) \cdot W_N^{nm}, \quad W_N = e^{-j\frac{2\pi}{N}} \quad . \tag{2.27}$$

Dividimos la secuencia de entrada en índices pares e impares:

$$X(m) = \sum_{n=0}^{N/2-1} x(2n) W_N^{2nm} + \sum_{n=0}^{N/2-1} x(2n+1) W_N^{(2n+1)m} \quad . \tag{2.28}$$

Factorizando:

$$X(m) = \sum_{n=0}^{N/2-1} x(2n) W_{N/2}^{nm} + W_N^m \sum_{n=0}^{N/2-1} x(2n+1) W_{N/2}^{nm} \quad . \tag{2.29}$$

Definimos:

$$E(m) = \sum_{n=0}^{N/2-1} x(2n) W_{N/2}^{nm}, \qquad O(m) = \sum_{n=0}^{N/2-1} x(2n+1) W_{N/2}^{nm} \ . \tag{2.30}$$

Entonces:

$$X(m) = E(m) + W_N^m O(m) \ , \tag{2.31}$$

$$X(m+N/2) = E(m) - W_N^m O(m) \ . \tag{2.32}$$

Este proceso puede aplicarse recursivamente hasta que todas las DFT sean de 2 puntos.
El bloque fundamental de la FFT radix-2 es la mariposa de 2 puntos:

$$X_0 = x_0 + x_1, \qquad X_1 = x_0 - x_1 \ . \tag{2.33}$$

Estas mariposas se repiten en cada etapa del algoritmo. La reorganización de los datos mediante **bit-reversal** es necesaria para que los datos estén en el orden correcto para el procesamiento.

Ventajas computacionales

La FFT radix-2 reduce la complejidad de $O(N^2)$ a $O(N \log_2 N)$, lo que es una mejora notoria, especialmente cuando N es grande.
Ejemplo:

$$\text{DFT directa: } N = 1024 \Rightarrow 1{,}048{,}576 \text{ multiplicaciones.} \tag{2.34}$$

$$\text{FFT radix-2: } \frac{1024}{2} \cdot \log_2(1024) = 512 \cdot 10 = 5120 \text{ multiplicaciones.} \tag{2.35}$$

Interpretación de los resultados de la FFT

La salida de la FFT es una secuencia compleja $X(m)$. La frecuencia absoluta del bin m es:

$$f_m = \frac{m \cdot f_s}{N} \ . \tag{2.36}$$

Para entrada real:

- Solo $m = 0$ a $N/2$ son únicos (simetría).

Para entrada compleja:

- Todos los N bins son significativos.

Magnitud del espectro

Dado:

$$X(m) = X_{\text{real}}(m) + jX_{\text{imag}}(m) \ , \tag{2.37}$$

la magnitud es:

$$|X(m)| = \sqrt{X_{\text{real}}^2(m) + X_{\text{imag}}^2(m)} \ . \tag{2.38}$$

Para obtener la amplitud real:

$$A(m) = \begin{cases} \frac{2}{N}|X(m)| & \text{si entrada real} \\ \frac{1}{N}|X(m)| & \text{si entrada compleja} \end{cases} \tag{2.39}$$

Espectro de potencia

La potencia se define como:

$$X_{PS}(m) = |X(m)|^2 \quad . \tag{2.40}$$

Potencia en decibelios:

$$X_{dB}(m) = 10 \cdot \log_{10}[X_{PS}(m)] \quad . \tag{2.41}$$

Potencia normalizada en dB:

$$X_{dB}(m) = 10 \cdot \log_{10}\left(\frac{X_{PS}(m)}{\text{máx}[X_{PS}(m)]}\right) \quad \text{ó} \quad 20 \cdot \log_{10}\left(\frac{|X(m)|}{|X(m)|_{\max}}\right) \quad . \tag{2.42}$$

Fase del espectro

La fase se calcula con:

$$X_{\phi}(m) = \tan^{-1}\left(\frac{X_{\text{imag}}(m)}{X_{\text{real}}(m)}\right) \quad . \tag{2.43}$$

Advertencia: Si $X_{\text{real}}(m) = 0$, se debe manejar la división para evitar errores: - Si $X_{\text{imag}} > 0 \rightarrow \phi = +90^\circ$ - Si $X_{\text{imag}} < 0 \rightarrow \phi = -90^\circ$ - Si $X_{\text{imag}} = 0 \rightarrow \phi = 0^\circ$

La fase solo es confiable cuando $|X(m)|$ está por encima del ruido.

Conclusión

La FFT radix-2:

- Reduce significativamente el esfuerzo computacional.
- Se basa en dividir y conquistar: reutiliza subresultados.
- Es idéntica a la DFT en resultado, pero más rápida.
- Es esencial para análisis espectral, diseño de filtros y procesamiento en tiempo real.

2.2.2 Método de Welch

El método de Welch es una refinación del método de Bartlett con dos mejoras principales. Primero, permite la superposición de segmentos de datos. Segundo, cada segmento es modulado con una ventana antes de calcular el periodograma.

Para describir matemáticamente este método, consideremos:

$$y_j(t) = y((j-1)K+t), \quad t = 1,\ldots,M, \quad j = 1,\ldots,S \quad , \tag{2.44}$$

donde $y_j(t)$ representa el j-ésimo segmento de datos y $(j-1)K$ es el punto inicial de dicho segmento. Si $K = M$, los segmentos son contiguos y equivalentes a los del método de Bartlett, resultando en $S = L = N/M$. Sin embargo, se recomienda usar $K = M/2$, lo que implica una superposición del 50 % entre segmentos y un número mayor de submuestras $S \approx 2M/N$.

El periodograma con ventana correspondiente a $y_j(t)$ se calcula como:

$$\hat{\phi}_j(\omega) = \frac{1}{MP}\left|\sum_{t=1}^{M} v(t)y_j(t)e^{-i\omega t}\right|^2 \quad , \tag{2.45}$$

donde P representa la "potencia" de la ventana $v(t)$:

$$P = \frac{1}{M}\sum_{t=1}^{M}|v(t)|^2 \quad . \tag{2.46}$$

La estimación de la densidad espectral de potencia (PSD) se obtiene como el promedio de los periodogramas calculados:

$$\hat{\phi}_W(\omega) = \frac{1}{S}\sum_{j=1}^{S}\hat{\phi}_j(\omega) \quad . \tag{2.47}$$

Las razones para estas modificaciones del método de Bartlett son simples. Al permitir solapamiento en los segmentos y promediar un mayor número de periodogramas en (2.47), se busca reducir la varianza de la

PSD estimada. Además, el uso de ventanas ayuda a reducir las correlaciones entre las submuestras sucesivas, aunque estén solapadas, proporcionando una mejor reducción de la varianza mediante la promediación en (2.47).

La relación entre el estimador de Welch y el de Blackman-Tukey se obtiene a partir de (2.45) y (2.47):

$$\hat{\phi}_W(\omega) = \frac{1}{S}\sum_{j=1}^{S}\frac{1}{MP}\sum_{t=1}^{M}\sum_{k=1}^{M} v(t)v^*(k)y_j(t)y_j^*(k)e^{-i\omega(t-k)} \quad . \tag{2.48}$$

Para valores grandes de N y cuando $K = M/2$ o menor, S es lo suficientemente grande para que el promedio,

$$\frac{1}{S}\sum_{j=1}^{S} y_j(t)y_j^*(k) \quad , \tag{2.49}$$

sea una buena aproximación de la covarianza $r(t-k)$. Asumiendo que la suma no depende de t y k, sino solo de su diferencia:

$$\tilde{r}(t,k) = \frac{1}{S}\sum_{j=1}^{S} y_j(t)y_j^*(k) \approx \tilde{r}(t-k) \quad . \tag{2.50}$$

Sustituyendo en (2.48), obtenemos:

$$\hat{\phi}_W(\omega) \approx \frac{1}{MP}\sum_{t=1}^{M}\sum_{k=1}^{M} v(t)v^*(k)\tilde{r}(t-k)e^{-i\omega(t-k)} \tag{2.51}$$

$$= \frac{1}{MP}\sum_{t=1}^{M}\sum_{\tau=-M}^{M} v(t)v^*(t-\tau)\tilde{r}(\tau)e^{-i\omega\tau} \tag{2.52}$$

$$= \sum_{\tau=-(M-1)}^{M-1}\left[\frac{1}{MP}\sum_{t=1}^{M} v(t)v^*(t-\tau)\right]\tilde{r}(\tau)e^{-i\omega\tau} \tag{2.53}$$

Definiendo la función de peso:

$$w(\tau) = \frac{1}{MP}\sum_{t=1}^{M} v(t)v^*(t-\tau) \quad , \tag{2.54}$$

bajo la convención de que $v(k) = 0$ para $k < 1$ y $k > M$, podemos escribir finalmente:

$$\hat{\phi}_W(\omega) \approx \sum_{\tau=-(M-1)}^{M-1} w(\tau)\tilde{r}(\tau)e^{-i\omega\tau} \quad . \tag{2.55}$$

Este resultado es comparable con la forma del estimador de Blackman-Tukey. En resumen, el estimador de Welch aproxima un estimador tipo Blackman-Tukey de la covarianza. Si bien la interpretación teórica del estimador de Blackman-Tukey es más sólida, la implementación del método de Welch mediante FFT lo convierte en una de las opciones más utilizadas en la estimación de densidad espectral de potencia (del inglés, *Power Spectral Density*, PSD).

2.3 Modulaciones digitales

Las modulaciones digitales son una herramienta clave en las comunicaciones actuales, ya que permiten transmitir información a través de señales digitales a través de diferentes medios de transmisión.

En esta sección se van a analizar tres modulaciones concretas, como son la modulación por amplitud de pulsos (del inglés, *Pulse Amplitude Modulation*, PAM), la modulación por desfase de portadora (del ingles, *Phase-Shift Keying*, PSK) y la modulación por amplitud en cuadratura (del inglés, *Quadrature Amplitude Modulation*, QAM), atendiendo a su modelo matématico, a las características de cada una de ellas y a su constelación.

2.3.1 Modulación por amplitud de pulso (PAM)

De acuerdo a [5], la modulación por amplitud de pulsos (PAM) es una técnica fundamental en las comunicaciones digitales que permite transformar una señal analógica en una secuencia de pulsos cuya amplitud refleja la información original. Este método es de gran importancia porque representa el primer paso en la conversión de señales analógicas a digitales dentro del proceso de modulación por código de pulsos (del inglés, *Pulse Code Modulation*, PCM). En ciertos casos, la señal PAM puede ser utilizada directamente sin necesidad de conversión posterior.

La modulación PAM es ampliamente utilizada para la transmisión de información en sistemas de comunicaciones debido a su simplicidad y facilidad de implementación. Esta técnica de modulación se basa en la variación de la amplitud de los pulsos para representar los datos, lo que la convierte en una de las técnicas más básicas y eficientes en sistemas de comunicación.

En la PAM digital, las formas de onda de señal pueden representarse como:

$$s_m(t) = A_m p(t), \quad 1 \leq m \leq M \ , \tag{2.56}$$

donde $p(t)$ es un pulso de duración T, y $\{A_m\}$ representa el conjunto de $M = 2^k$ posibles amplitudes correspondientes a los bloques de símbolos de k bits.

Usualmente, las amplitudes A_m toman valores discretos como:

$$A_m = 2m - 1 - M, \quad m = 1, 2, \ldots, M \ , \tag{2.57}$$

es decir, las amplitudes son $\pm 1, \pm 3, \pm 5, \ldots, \pm(M-1)$. La forma del pulso $p(t)$ afecta el espectro de la señal transmitida.

La energía de la señal $s_m(t)$ es:

$$E_m = \int_{-\infty}^{\infty} A_m^2 p^2(t)\, dt = A_m^2 E_p \ , \tag{2.58}$$

donde E_p es la energía del pulso $p(t)$.

Codificación Gray y distancia euclidiana

Una posible asignación de amplitudes de señal para PAM puede hacerse de varias formas. La asignación preferida es aquella donde los símbolos adyacentes difieren en un solo bit binario. Esta asignación se denomina **codificación Gray**. Es importante para la demodulación de la señal, ya que los errores más probables causados por ruido implican la selección errónea de un símbolo adyacente al transmitido, lo cual produce solo un error de bit en la secuencia de k bits.

La distancia euclidiana entre cualquier par de puntos de señal es:

$$d_{mn} = |A_m - A_n| \sqrt{E_p} \ , \tag{2.59}$$

y para el caso de PAM paso banda:

$$d_{mn} = |A_m - A_n| \sqrt{\frac{E_g}{2}} \ . \tag{2.60}$$

Para señales adyacentes, $|A_m - A_n| = 2$, y la distancia mínima de la constelación se expresa como:

$$d_{\text{mín}} = 2\sqrt{E_p} = \sqrt{2E_g} \ . \tag{2.61}$$

Podemos expresar la distancia mínima de un sistema PAM-M en función de su energía promedio por bit, E_{bavg}, obteniendo:

$$d_{\text{mín}} = \sqrt{\frac{12 \log_2 M}{M^2 - 1} E_{bavg}} \ . \tag{2.62}$$

Señal PAM SSB

La señal PAM con portadora es de doble banda lateral (DSB) y requiere el doble de ancho de banda que su equivalente en baja frecuencia. Alternativamente, podemos usar PAM de banda lateral única (SSB), cuya representación es:

$$s_m(t) = \Re\left\{A_m[g(t) \pm j\hat{g}(t)] e^{j2\pi f_c t}\right\}, \quad m = 1, 2, \ldots, M \ , \tag{2.63}$$

donde $\hat{g}(t)$ es la transformada de Hilbert de $g(t)$. La señal SSB requiere solo la mitad del ancho de banda que la señal DSB.

En el caso especial de $M = 2$, o señales binarias, las formas de onda de PAM cumplen que:

$$s_1(t) = -s_2(t) \quad .$$

Estas señales tienen la misma energía y un coeficiente de correlación cruzada de -1. Dichas señales se denominan antipodales. Este caso particular se conoce como **modulación binaria antipodal**.

Muestreo y ancho de banda

De acuerdo con el teorema de muestreo, es posible reconstruir una señal analógica a partir de muestras tomadas a intervalos regulares, siempre que la frecuencia de muestreo f_s sea al menos el doble de la máxima frecuencia de la señal original, es decir, $f_s \geq 2B$, donde B representa la máxima frecuencia de la señal analógica y $2B$ es conocido como la tasa de Nyquist.

Debido a que la modulación PAM utiliza pulsos en lugar de muestras instantáneas, su ancho de banda es mayor que el de la señal analógica original. Sin embargo, los pulsos ofrecen ventajas en la implementación digital.

Tipos de señal PAM

Existen dos formas principales de señales PAM:

- **Muestreo Natural (gating)**: Se obtiene cuando la señal analógica se multiplica por una onda rectangular de pulsos. El muestreo natural se expresa matemáticamente como:

$$w_s(t) = w(t)s(t) \quad , \tag{2.64}$$

  donde $w(t)$ es la señal analógica original y $s(t)$ es una onda rectangular que actúa como interruptor. La onda rectangular $s(t)$ se define como:

$$s(t) = \sum_{k=-\infty}^{\infty} \Pi\left(\frac{t - kT_s}{\tau}\right) \quad , \tag{2.65}$$

  donde $\Pi(t)$ representa una función rectangular, T_s es el período de muestreo, y τ es la duración del pulso.

- **Muestreo Instantáneo**: Se obtiene tomando muestras discretas de la señal en instantes específicos.

Espectro de la señal PAM. Constelación

El espectro de una señal PAM muestreada naturalmente se expresa como:

$$W_s(f) = d \sum_{n=-\infty}^{\infty} \left(\frac{\sin(\pi nd)}{\pi nd}\right) W(f - nf_s) \quad , \tag{2.66}$$

donde $d = \frac{\tau}{T_s}$ es el ciclo de trabajo de la onda rectangular. Esta expresión indica que el espectro de la señal PAM consiste en copias del espectro original $W(f)$ desplazadas en múltiplos de la frecuencia de muestreo f_s y moduladas por una función sinc.

En la *Figura 2.1*, se puede observar la representación de la constelación de una 8-PAM. Esto lleva al hecho de que cada señal en un carácter está indicada por un símbolo que contiene una de las ocho amplitudes posibles. Estos valores suelen ser los mismos y están destinados a maximizar la eficiencia y minimizar el rendimiento promedio.

Las constelaciones PAM son 1D, ya que solo son visibles en el eje de amplitud (o eje) cuando la representación se usa a niveles complejos. Esta es una modulación eficiente de ancho de banda, pero es relativamente sensible al ruido. Al aumentar el número de niveles de amplitud, la distancia entre ellos disminuye para una energía promedio de símbolo fija.

2.3.2 Modulación por desplazamiento de fase (PSK)

En la modulación digital por fase, las M formas de onda de señal se representan como:

$$s_m(t) = \Re\left\{g(t)e^{j\left[\frac{2\pi(m-1)}{M}\right]}e^{j2\pi f_c t}\right\} \quad , \tag{2.67}$$

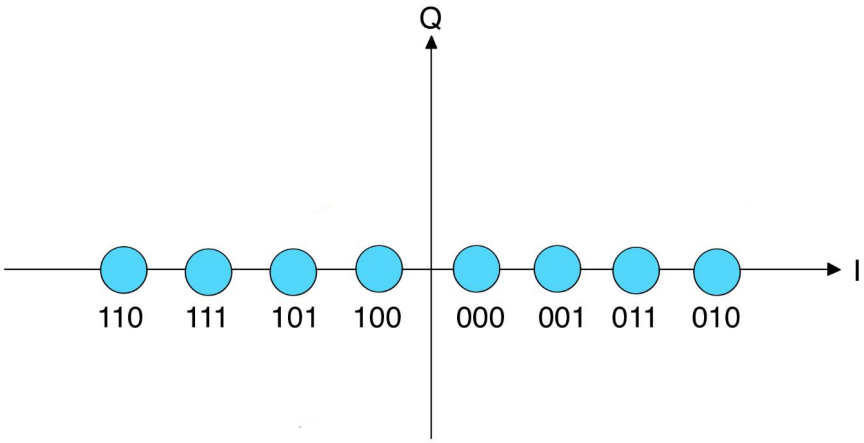

Figura 2.1 Esquema de constelación 8-PAM.

lo cual se puede expandir como:

$$s_m(t) = g(t)\cos\left(\frac{2\pi(m-1)}{M}\right)\cos(2\pi f_c t) - g(t)\sin\left(\frac{2\pi(m-1)}{M}\right)\sin(2\pi f_c t) \quad . \tag{2.68}$$

Aquí, $g(t)$ es la forma del pulso de la señal, y $\phi_m = \frac{2\pi(m-1)}{M}$ representa las posibles fases de la portadora que codifican la información. Esta modulación se conoce comúnmente como *Phase-Shift Keying* (PSK).

Todas las formas de onda tienen igual energía. Se deduce que:

$$E = E_{avg} = \frac{1}{2}E_g \quad , \tag{2.69}$$

donde E_g es la energía del pulso $g(t)$.

Las funciones base ortogonales para la expansión de $s_m(t)$ son:

$$\phi_1(t) = \sqrt{2/E_g}\cdot g(t)\cos(2\pi f_c t) \quad , \tag{2.70}$$

$$\phi_2(t) = -\sqrt{2/E_g}\cdot g(t)\sin(2\pi f_c t) \quad . \tag{2.71}$$

Por tanto, la expansión de $s_m(t)$ en el espacio de señales queda como:

$$s_m(t) = \sqrt{E_g/2}\cos\left(\frac{2\pi(m-1)}{M}\right)\phi_1(t) + \sqrt{E_g/2}\sin\left(\frac{2\pi(m-1)}{M}\right)\phi_2(t) \quad . \tag{2.72}$$

Esto da lugar a representaciones vectoriales bidimensionales:

$$\mathbf{s}_m = \left[\sqrt{\frac{E_g}{2}}\cos\left(\frac{2\pi(m-1)}{M}\right), \sqrt{\frac{E_g}{2}}\sin\left(\frac{2\pi(m-1)}{M}\right)\right] \quad . \tag{2.73}$$

Distancia mínima euclidiana

La distancia euclidiana entre dos puntos de señal es:

$$d_{mn} = \sqrt{2E_g\left[1-\cos\left(\frac{2\pi(m-n)}{M}\right)\right]} = 2\sqrt{E_g}\sin\left(\frac{\pi(m-n)}{M}\right) \quad . \tag{2.74}$$

Para símbolos adyacentes ($|m-n|=1$), la distancia mínima es:

$$d_{\text{mín}} = 2\sqrt{E_g}\sin\left(\frac{\pi}{M}\right) \quad . \tag{2.75}$$

Al sustituir E_g en función de la energía promedio por bit E_b:

$$d_{\text{mín}} = 2\sqrt{E_b \cdot \log_2 M \cdot \sin^2\left(\frac{\pi}{M}\right)} \quad . \tag{2.76}$$

Para valores grandes de M, se aproxima como:

$$d_{\text{mín}} \approx 2\sqrt{\frac{\pi^2}{M^2}\log_2 M \cdot E_b} \quad . \tag{2.77}$$

Una variante de PSK, llamada $\pi/4$–QPSK, introduce un desplazamiento adicional de fase de $\pi/4$ en cada intervalo de símbolo, lo que facilita la sincronización de símbolos.

Constelación 8-PSK

La modulación 8-PSK consiste en una modulación de fase. El objetivo es codificar 3 bits por símbolo requeridos (como la 8-PAM), pero distribuyéndolos en el plano complejo, es decir, variando tanto la amplitud como la fase de la señal (véase la *Figura 2.2*).

En las constelaciones típicas 8-PSK, los puntos no están alineados con el eje, sino que se colocan en varios radios y ángulos. Por ejemplo, puede tener dos niveles de amplitud y cuatro fases diferentes o combinaciones diferentes que dan lugar a 8 combinaciones únicas.

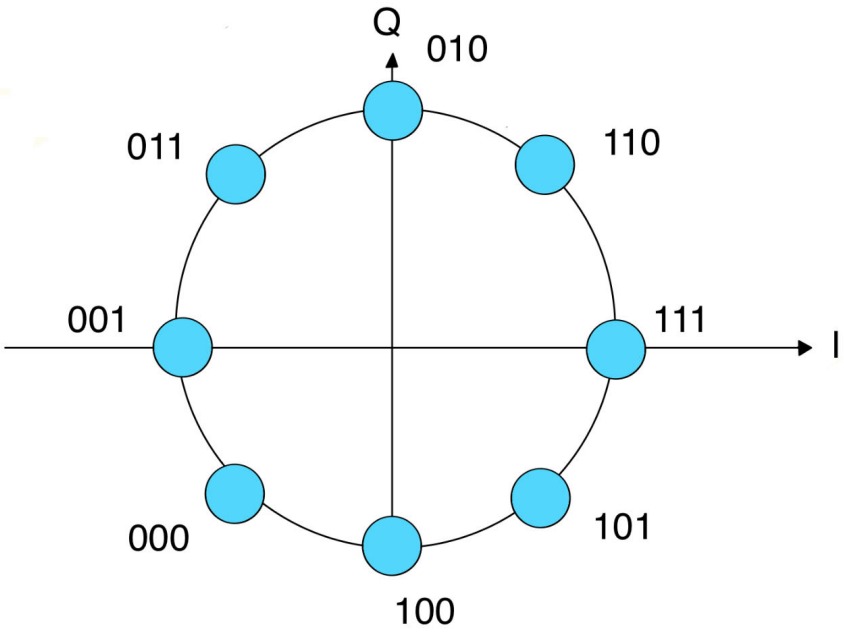

Figura 2.2 Esquema de constelación 8-PSK.

Este tipo de modulación es más robusto considerando un tipo de ruido particular, lo que permite un uso más eficiente del espacio de constelaciones.

2.3.3 Modulación por amplitud en cuadratura (QAM)

La eficiencia espectral de la modulación PAM/SSB también puede lograrse al imponer simultáneamente dos símbolos de k bits de la secuencia de información en dos portadoras en cuadratura, $\cos(2\pi f_c t)$ y $\sin(2\pi f_c t)$. Esta técnica de modulación se llama *Quadrature Amplitude Modulation* (QAM).

Las formas de onda de la señal se pueden expresar como:

$$s_m(t) = \Re\left\{(A_{mi} + jA_{mq})g(t)e^{j2\pi f_c t}\right\} \quad , \tag{2.78}$$

lo que se traduce en:

$$s_m(t) = A_{mi}g(t)\cos(2\pi f_c t) - A_{mq}g(t)\sin(2\pi f_c t) \quad . \tag{2.79}$$

Aquí, A_{mi} y A_{mq} son las amplitudes de señal correspondientes a las componentes en fase y en cuadratura, respectivamente, y $g(t)$ es el pulso de forma.

Alternativamente, las formas de onda de QAM también pueden representarse como:

$$s_m(t) = r_m \cos(2\pi f_c t + \phi_m) \quad , \tag{2.80}$$

donde:

$$r_m = \sqrt{A_{mi}^2 + A_{mq}^2} \quad ,$$
$$\phi_m = \tan^{-1}\left(\frac{A_{mq}}{A_{mi}}\right) \quad .$$

Usando las funciones ortonormales $\phi_1(t)$ y $\phi_2(t)$, las señales QAM se pueden representar como:

$$s_m(t) = A_{mi}\sqrt{\frac{E_g}{2}}\,\phi_1(t) + A_{mq}\sqrt{\frac{E_g}{2}}\,\phi_2(t) \quad . \tag{2.81}$$

Por tanto, la representación vectorial es:

$$\mathbf{s}_m = \left(A_{mi}\sqrt{\frac{E_g}{2}}, A_{mq}\sqrt{\frac{E_g}{2}}\right) \quad . \tag{2.82}$$

La energía de la señal es:

$$E_m = \frac{E_g}{2}\left(A_{mi}^2 + A_{mq}^2\right) \quad . \tag{2.83}$$

Distancia euclidiana

La distancia euclidiana entre dos vectores de señal es:

$$d_{mn} = \sqrt{\frac{E_g}{2}\left[(A_{mi} - A_{ni})^2 + (A_{mq} - A_{nq})^2 \quad .\right]} \tag{2.84}$$

Para constelaciones rectangulares, la distancia mínima es:

$$d_{\text{mín}} = \sqrt{2E_g} \quad . \tag{2.85}$$

La energía promedio transmitida es:

$$E_{\text{avg}} = \frac{M-1}{3}E_g \quad . \tag{2.86}$$

De aquí, se obtiene:

$$E_{b,\text{avg}} = \frac{M-1}{3\log_2 M}E_g \quad . \tag{2.87}$$

Entonces, la distancia mínima en términos de E_b es:

$$d_{\text{mín}} = \sqrt{\frac{6\log_2 M}{M-1}E_{b,\text{avg}}} \quad . \tag{2.88}$$

Constelación 16-QAM

En la *Figura 2.3* se muestra la constelación de una modulación 16-QAM. La modulación 16-QAM es una técnica bidimensional en la que los componentes en fase (I) y en cuadratura (Q) de la señal pueden variar de forma simultánea para codificar información. En este caso, se utilizan 16 combinaciones posibles, lo que permite transmitir 4 bits por símbolo, ya que $2^4 = 16$.

Las constelaciones 16-QAM se representan comúnmente como una cuadrícula de 4×4 puntos distribuidos en el plano I-Q. Cada punto representa un símbolo distinto con una combinación específica de amplitudes en los ejes I y Q. Esta disposición permite aprovechar de forma más eficiente el espacio disponible en la

constelación, en comparación con esquemas de modulación que solo varían la amplitud (como PAM) o solo la fase (como PSK).

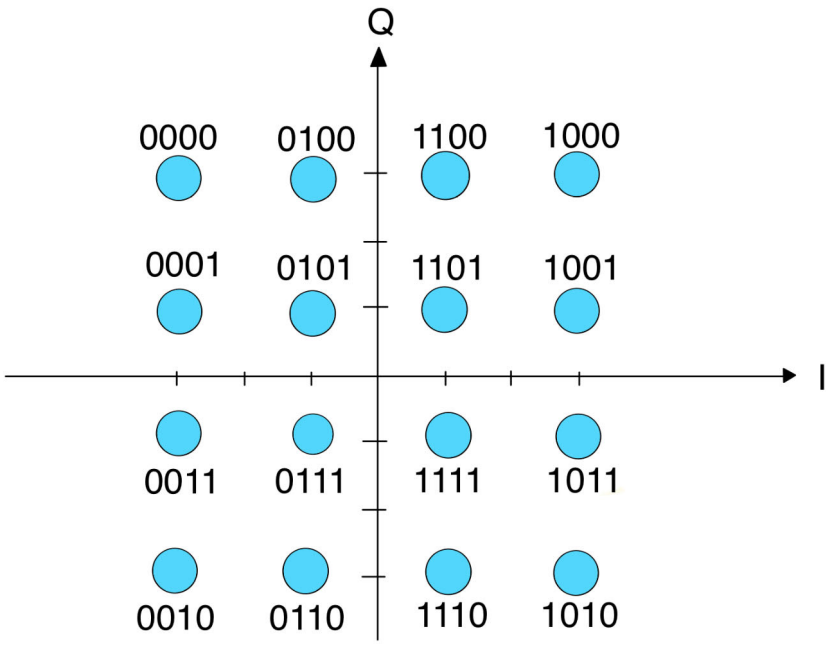

Figura 2.3 Esquema de constelación 16-QAM.

La 16-QAM ofrece un buen equilibrio entre eficiencia espectral y complejidad de implementación. Sin embargo, al incrementar la densidad de puntos en la constelación (es decir, más bits por segundo en el mismo ancho de banda), los puntos se encuentran más próximos entre sí. Esto hace que la modulación sea más sensible al ruido y, por tanto, se requiera una relación señal-ruido (SNR) mayor para distinguir correctamente los símbolos y minimizar la probabilidad de error.

Comparación y observaciones

Las modulaciones PAM, PSK y QAM pueden expresarse como:

$$s_m(t) = \Re\left[A_m g(t) e^{j2\pi f_c t}\right] \quad , \tag{2.89}$$

donde:

- Para PAM: $A_m \in \mathbb{R}$
- Para PSK: $A_m = e^{j2\pi(m-1)/M}$
- Para QAM: $A_m = A_{mi} + jA_{mq}$

Esto demuestra que todas pertenecen a la misma familia de esquemas de modulación, siendo PAM y PSK casos particulares de QAM. En QAM, tanto la amplitud como la fase transportan información, mientras que en PAM y PSK solo una de ellas lo hace.

2.3.4 Demodulación de señales digitales.

La demodulación es el proceso mediante el cual una señal recibida se convierte nuevamente en información útil después de haber sido transmitida a través de un canal. En un sistema de comunicación digital, este proceso implica extraer los símbolos digitales transmitidos a partir de una forma de onda modulada que ha sido afectada por el canal de transmisión.

La detección óptima en estos sistemas requiere filtros adaptados a $\phi_1(t)$ y $\phi_2(t)$. Dado que tanto la señal recibida $r(t)$ como las funciones base son señales paso banda de alta frecuencia, el proceso de filtrado, si se implementa en software, requiere altas tasas de muestreo.

Para aliviar este requerimiento, primero podemos demodular la señal recibida para obtener su señal equivalente de baja frecuencia y luego realizar la detección sobre esta señal.

Es importante notar que el proceso de demodulación es un proceso invertible. Por lo tanto, el detector óptimo diseñado para la señal demodulada funciona tan bien como el detector óptimo diseñado para la señal paso banda. El beneficio de la implementación demodulador-detector es que en esta estructura el procesamiento de señal requerido para la detección se realiza sobre la señal demodulada de baja frecuencia, reduciendo así la complejidad del receptor.

2.3.5 Demoduladores PAM, PSK, QAM

Para señales PAM, se requiere un correlador único, y el detector es un detector de amplitud. Se incluye un control automático de ganancia (del inglés, *Automatic Gain Control*, AGC) al principio del demodulador para eliminar variaciones de ganancia del canal.

Para señales QAM, el demodulador es similar al de PSK, ya que ambos generan muestras en fase y en cuadratura para el detector. En el caso de QAM, el detector calcula la distancia euclidiana entre el punto recibido con ruido y los M posibles puntos transmitidos, y selecciona el más cercano.

2.3.6 Tasa de error de bit (BER) y criterio de mínimo error cuadrático medio (MSE)

En sistemas de comunicación digital, la tasa de error de bit (BER) es una métrica fundamental para evaluar el rendimiento. Para un ecualizador lineal, la salida estimada se expresa como:

$$\hat{a}_k = \sum_{n=-N}^{N} w_n r_{k-n} \quad , \tag{2.90}$$

donde r_k es la secuencia recibida, w_n son los coeficientes del ecualizador, y $\hat{a}_k$ es la estimación del símbolo transmitido.

A partir de (**??**) se define la tasa de error de bit (BER) como:

$$\text{BER} = \frac{1}{N_b} \sum_{k=1}^{N_b} \mathbb{I}\left(\hat{b}_k \neq b_k\right) \tag{2.91}$$

donde N_b es el número total de bits transmitidos, a_k es el bit original, $\hat{a}_k$ es el bit estimado, y $\mathbb{I}$ es la función indicadora que vale 1 cuando su argumento es verdadero, y 0 en caso contrario.

Criterio de mínimo error cuadrático medio (MSE)

El criterio MSE busca minimizar el valor esperado del error cuadrático entre el símbolo transmitido y su estimación:

$$\text{MSE} = \mathbb{E}\left[|a_k - \hat{a}_k|^2\right] \quad . \tag{2.92}$$

El conjunto óptimo de coeficientes $\mathbf{w}_{opt}$ se obtiene resolviendo:

$$\mathbf{w}_{opt} = \mathbf{R}^{-1}\mathbf{p} \quad , \tag{2.93}$$

donde $\mathbf{R}$ es la matriz de autocorrelación de r_k, y $\mathbf{p}$ es el vector de correlación cruzada entre r_k y a_k.

Relación con la BER

Para un canal con interferencia intersimbólica (ISI) y ruido blanco gaussiano aditivo (AWGN), la BER puede aproximarse como:

$$P_b \approx Q\left(\sqrt{\frac{d_{\text{eff}}^2}{2N_0}}\right) \quad , \tag{2.94}$$

donde d_{eff} es la distancia efectiva entre símbolos después de la ecualización, y N_0 es la densidad espectral de potencia del ruido.

En un ecualizador óptimo bajo el criterio MSE, la señal útil se reduce debido a la distorsión residual, lo que puede afectar negativamente la BER. La compensación ideal entre complejidad y rendimiento se alcanza mediante ecualizadores como:

- Ecualizador de decisión retroalimentada

- Ecualizador fraccionalmente espaciado

Exceso de MSE debido a estimaciones ruidosas

Cuando se usan algoritmos adaptativos como LMS, el MSE mínimo no se alcanza exactamente debido a la naturaleza ruidosa de los gradientes. Esto introduce un exceso de MSE, definido como:

$$\text{Exceso de MSE} = \text{MSE}_{\text{final}} - \text{MSE}_{\text{mínima}} \quad . \tag{2.95}$$

Este exceso depende de la tasa de aprendizaje, longitud del ecualizador y características estadísticas del canal y del ruido.

2.3.7 Magnitud del vector de error (EVM) y su relación con MSE

En la evaluación moderna de sistemas de modulación digital, especialmente en estándares como LTE y 5G, una métrica comúnmente utilizada es la magnitud del vector de error (Error Vector Magnitude, EVM).

El EVM mide la distancia entre el símbolo recibido (posiblemente distorsionado) y el símbolo ideal esperado en el espacio de la constelación. Se define como:

$$\text{EVM}_{\text{rms}} = \sqrt{\frac{\sum_{k=1}^{N} |a_k - \hat{a}_k|^2}{\sum_{k=1}^{N} |a_k|^2}} \quad . \tag{2.96}$$

El numerador de la expresión de EVM es proporcional al MSE:

$$\text{MSE} = \mathbb{E}\left[|a_k - \hat{a}_k|^2\right] \quad . \tag{2.97}$$

Por tanto, podemos expresar la EVM en términos de MSE como:

$$\text{EVM}_{\text{rms}} = \sqrt{\frac{\text{MSE}}{\mathbb{E}[|a_k|^2]}} \quad . \tag{2.98}$$

Interpretación práctica

El EVM expresa la degradación general del sistema en términos de interferencia, distorsión no lineal, ruido y errores de sincronización. Es por ello que, a menor EVM, mejor rendimiento del sistema. Los valores típicos de EVM aceptables dependen de la orden de modulación. Por ejemplo:

- QPSK: EVM $\leq 17.5\,\%$
- 16-QAM: EVM $\leq 12.5\,\%$
- 64-QAM: EVM $\leq 8\,\%$

Aunque la BER es una medida directa del rendimiento en términos de errores de bit, EVM ofrece una métrica más inmediata y continua del estado del sistema. Puede correlacionarse con la BER mediante análisis estadístico en canales Gaussianos.

2.4 Marco legal de transmisión europeo. Bandas de frecuencia.

El marco legal para la transmisión en Europa está regulado por la Unión Europea a través de directrices y regulaciones que promueven la competencia, la interoperabilidad, el acceso a la infraestructura de comunicaciones y el acceso para un mercado digital único y sostenible.

2.4.1 Regulaciones europeas con respecto al uso del espectro radioeléctrico

La Comisión Europea, según [18], establece una serie de normativas y directivas para garantizar que el espectro radioeléctrico se gestione de manera eficiente en toda la Unión Europea. Una de las principales regulaciones en este ámbito es la **Decisión 676/2002/CE** del Parlamento Europeo y del Consejo, que define el marco para la gestión del espectro radioeléctrico en Europa. Estas regulaciones exigen la armonización de las políticas de radiofrecuencia entre los Estados miembros y fomentan la implementación de nuevas tecnologías y servicios de telecomunicaciones.

Además, la **Directiva 2002/21/CE** (Directiva Marco de Telecomunicaciones) regula los aspectos fundamentales de las telecomunicaciones electrónicas en Europa, incluyendo la gestión del espectro, la competencia y

la protección de los usuarios. Estas directivas establecen los principios clave para garantizar que los servicios de telecomunicaciones sean accesibles, seguros y competitivos para todos los ciudadanos de la UE.

2.4.2 Uso responsable y sostenible del espectro

Dentro del marco legal, se toma en cuenta la naturaleza limitada del espectro radioeléctrico, por lo que las regulaciones europeas promueven su uso responsable y sostenible. La asignación de frecuencias se realiza de manera que se maximicen los beneficios del espectro, minimizando las interferencias entre diferentes servicios y garantizando la calidad del servicio para los usuarios finales.

El **Reglamento (UE) 2017/900** del Parlamento Europeo y del Consejo establece un sistema para asignar eficientemente las frecuencias para nuevas tecnologías, como las redes 5G. Este reglamento asegura que los recursos espectrales estén disponibles para el desarrollo de nuevas aplicaciones de telecomunicaciones y promueve la innovación en la región.

2.4.3 Supervisión y cumplimiento

Los Estados miembros de la UE son responsables de implementar estas regulaciones a nivel nacional, bajo la supervisión de la **Agencia de la Unión Europea para la Ciberseguridad** (ENISA) y la **Red Europea de Reguladores de Comunicaciones Electrónicas** (BEREC). Estas instituciones aseguran que las normativas sean cumplidas correctamente a nivel nacional y regional, promoviendo la transparencia y la competitividad en el mercado europeo de telecomunicaciones.

2.4.4 Bandas de frecuencia

El espectro eléctrico de una aplicación particular se divide en diferentes bandas de frecuencia. Estas bandas de frecuencia están reguladas y asignadas por organizaciones internacionales como la Comisión Federal de Comunicaciones de los Estados Unidos (FCC), la Unión Internacional de Telecomunicaciones (UIT) y la Agencia Nacional de Comunicaciones de Brasil (Anatel).

A continuación, explicaremos las principales bandas de frecuencia, su uso y las aplicaciones asociadas a cada una de ellas.

1. Banda de muy baja frecuencia (VLF) de 3 kHz a 30 kHz

Aplicaciones principales:

- **Comunicación por radio a largo alcance:** Esta banda se utiliza principalmente para la comunicación a largas distancias, como la comunicación entre estaciones de radio de bajo nivel y submarinos.
- **Navegación y geolocalización:** Se utiliza en los sistemas de navegación marítima y submarina.

Aplicaciones específicas:

- **Submarinos:** Las comunicaciones VLF son extremadamente importantes para la comunicación en submarinos durante operaciones militares, ya que las ondas VLF pueden penetrar grandes profundidades bajo el agua.
- **Sistemas de ubicación:** Se utiliza para determinar la geolocalización en algunas tecnologías de navegación especializadas.

2. Banda de baja frecuencia (LF) de 30 kHz a 300 kHz

Aplicaciones principales:

- **Radio de onda larga:** Las señales LF pueden cubrir distancias largas debido a la curvatura de la tierra, por lo que se utilizan para transmitir señales de radio, especialmente en rutas muy largas.
- **Navegación marítima:** Se utiliza en sistemas de navegación de baliza debido a su amplio alcance.

Aplicaciones específicas:

- **Radiodifusión:** Se utiliza para la cobertura de radio de onda larga, especialmente en transmisiones que cubren grandes distancias.
- **Sistemas de navegación:** Se emplea en la navegación marítima y aeronáutica para garantizar la precisión de las rutas y ubicaciones.

3. Banda de frecuencia media (MF) de 300 kHz a 3 MHz

Aplicaciones principales:

- **Radiodifusión (AM):** La modulación de amplitud es la técnica utilizada con mayor frecuencia en esta banda. Es común en radios comerciales y servicios públicos.
- **Comunicación de corto alcance:** Se utiliza para comunicaciones locales, como las utilizadas en estaciones de servicio.

Aplicaciones específicas:

- **Radiodifusión AM:** Esta banda se utiliza principalmente para transmitir estaciones de radio AM, que pueden cubrir distancias de hasta miles de kilómetros dependiendo de las condiciones de propagación.
- **Sistemas de comunicación marítima y aeronáutica:** Las bandas MF también se utilizan para la comunicación entre aviones, barcos y estaciones terrestres.

4. Banda de alta frecuencia (HF) de 3 MHz a 30 MHz

Aplicaciones principales:

- **Comunicación de radio de onda corta:** Utilizada para comunicaciones de radio internacionales, especialmente en sistemas de radio de emergencia.
- **Radio internacional:** Las emisoras internacionales utilizan frecuencias de esta banda para llegar al público global.

Aplicaciones específicas:

- **Radio amateur:** Los operadores de radio aficionado o estaciones privadas utilizan esta banda para comunicaciones a largas distancias.
- **Radiodifusión internacional:** Algunas estaciones internacionales, como el servicio público de radio, utilizan esta banda para llegar a varios países, particularmente en áreas fuera del alcance de las bandas de onda media.
- **Comunicación de emergencia:** En situaciones de emergencia, la frecuencia HF se utiliza para establecer comunicación cuando otras redes no funcionan.

5. Banda de frecuencia muy alta (VHF) de 30 MHz a 300 MHz

Aplicaciones principales:

- **TV y radio FM:** Las transmisiones de radio FM y televisión utilizan frecuencias en esta banda.
- **Comunicación aeronáutica y marítima:** Se utiliza para la comunicación entre aviones y estaciones terrestres, así como para la comunicación marina.

Aplicaciones específicas:

- **Radio FM:** Las emisoras de radio FM utilizan esta banda para ofrecer transmisiones de audio comerciales y públicas, destacándose por su alta calidad de sonido debido a la modulación de frecuencia.
- **Televisión:** Muchas transmisiones de televisión analógica y digital se realizan en frecuencias VHF.
- **Comunicación de seguridad:** Los dispositivos inalámbricos VHF se utilizan frecuentemente en sistemas de comunicación de emergencia, como en la policía y los bomberos.
- **Comunicación aeronáutica y marítima:** Esta frecuencia es clave para la comunicación entre aeronaves y estaciones terrestres en la aviación y la navegación marítima.

6. Banda de ultra alta frecuencia (UHF) de 300 MHz a 3 GHz

Aplicaciones principales:

- **Comunicaciones móviles y satelitales:** Las frecuencias UHF son esenciales para las telecomunicaciones móviles (3G, 4G, 5G) y comunicaciones por satélite.
- **Televisión digital y radio:** La transmisión de televisión digital y algunos servicios de radio utilizan frecuencias en esta banda.

Aplicaciones específicas:

- **Servicios de teléfonos móviles:** Esta banda se utiliza en las generaciones de teléfonos móviles (2G, 3G, 4G) y se usará en 5G.
- **Comunicación por satélite:** UHF se utiliza para las comunicaciones satelitales y los sistemas de radar.
- **Redes Wi-Fi y Bluetooth:** Las frecuencias altas dentro de UHF se utilizan en redes locales como Wi-Fi y Bluetooth.

7. Banda de frecuencia súper alta (SHF) de 3 GHz a 30 GHz

Aplicaciones principales:

- **Comunicación por satélite:** Esta banda se utiliza en la comunicación por satélite, especialmente en sistemas geosincrónicos y no geosincrónicos.
- **Radar y comunicaciones militares:** Los sistemas de radar civiles y militares utilizan frecuencias SHF para reconocer objetos de largo alcance, como aviones y satélites.

Aplicaciones específicas:

- **Comunicación satelital de banda ancha:** Se utiliza para servicios de transmisión de datos de alta velocidad, como Internet satelital.
- **Radar de control de tráfico aéreo:** Se emplea en sistemas de radar en aeropuertos y para defensa.
- **Redes de telecomunicaciones de alta velocidad:** Las tecnologías como 5G utilizan frecuencias SHF para ofrecer altas velocidades de transmisión de datos.

8. Banda de frecuencia extremadamente alta (EHF) de 30 GHz a 300 GHz

Aplicaciones principales:

- **Comunicación de microondas:** Se utiliza para permitir transmisiones de gran capacidad a través de cables físicos.
- **Investigación científica:** Se utiliza en experimentos astrofísicos y estudios atmosféricos debido a las propiedades únicas de las ondas electromagnéticas en esta banda.

Aplicaciones específicas:

- **Sistemas de comunicación de alta velocidad:** Se utiliza para transmisiones de datos a gran velocidad entre estaciones terrestres.
- **Investigación científica y radar extendido:** Se emplea en aplicaciones de radar para experimentos científicos avanzados, como telescopios y estudios de partículas.

Las frecuencias del espectro eléctrico son vastas y cubren muchos segmentos específicos con aplicaciones diferenciadas. Desde las comunicaciones de largo alcance en frecuencias bajas hasta las tecnologías de alta velocidad en las frecuencias más altas, cada banda tiene aplicaciones fundamentales que permiten una amplia variedad de comunicaciones eficientes y conectadas en todo el mundo. La regulación y gestión adecuada de este espectro es crucial para garantizar la eficiencia de las comunicaciones, la seguridad y el continuo desarrollo tecnológico. En la *Tabla 2.1* se ha recopilado toda la información sobre las bandas de frecuencia de transmisión desarrollada en este capítulo.

Tabla 2.1 Bandas de frecuencia y sus aplicaciones principales y específicas.

Banda (Nombre y Rango)	Aplicaciones principales	Aplicaciones específicas	Frecuencia central aprox.
VLF (3–30 kHz)	Comunicación por radio a largo alcance; navegación submarina y marítima.	Comunicación con submarinos; sistemas de geolocalización bajo el agua.	16.5 kHz
LF (30–300 kHz)	Radio de onda larga; navegación de baliza.	Radiodifusión de larga distancia; navegación marítima/aeronáutica.	165 kHz
MF (300 kHz–3 MHz)	Radiodifusión AM; comunicación de corto alcance.	Radio AM; comunicación marítima y aeronáutica.	1.65 MHz
HF (3–30 MHz)	Radio de onda corta; radio internacional.	Radioaficionados; radiodifusión global; comunicación de emergencia.	16.5 MHz
VHF (30–300 MHz)	Radio FM; televisión; comunicación aeronáutica y marítima.	Radio FM comercial; TV analógica/digital; comunicación de seguridad.	165 MHz
UHF (300 MHz–3 GHz)	Telefonía móvil; TV digital; comunicaciones satelitales.	3G–5G; radar; Wi-Fi/Bluetooth; servicios móviles.	1.65 GHz
SHF (3–30 GHz)	Comunicaciones satelitales; radar civil y militar.	Internet satelital; radar de control aéreo; 5G de alta velocidad.	16.5 GHz
EHF (30–300 GHz)	Microondas; investigación científica.	Comunicaciones de datos de alta velocidad; astrofísica y radar extendido.	165 GHz

2.4.5 Cuadro nacional de atribución de frecuencias (CNAF)

Como se describe en [19] y en [20], el CNAF se publicó por vez primera en 1990. Debido al contenido regulador y marcadamente técnico de su información en cuanto a la utilización del espectro radioeléctrico se refiere, es un documento que requiere constantes actualizaciones derivadas generalmente de las actividades de los organismos internacionales con competencias regulatorias a los que España pertenece.

Actualizaciones nacionales

Además, el CNAF puede actualizarse por exigencias nacionales derivadas de:

- Disposiciones regulatorias relativas al uso del espectro
- Modificaciones de la Ley General de Telecomunicaciones (LGT)
- Necesidades de la industria española y del sector de las telecomunicaciones
- Nuevos sistemas y dispositivos que requieren operar en determinadas frecuencias

El CNAF transpone al ordenamiento legal español los cambios y modificaciones de atribución de bandas de frecuencia y servicios radioeléctricos derivados de:

- Conferencias Mundiales de Radiocomunicaciones de la UIT
- Decisiones de la Comisión Europea

Versión actual

La versión actual del CNAF fue aprobada por la **Orden ETD 1449/2021**, de 16 de diciembre, y publicada en el Boletín Oficial del Estado el 24 de diciembre de 2021. Esta versión deroga y sustituye la edición anterior de 2017 y sus modificaciones parciales de 2018 y 2020.

Las modalidades de uso se clasifican con los siguientes códigos:

- **C**: Uso común
- **E**: Uso especial
- **P**: Uso privativo
- **R**: Uso reservado al Estado
- **M**: Uso mixto (comprende usos P y R)

Sin embargo, se ha optado por recurrir a la versión de Estados Unidos, al no haber encontrado la española. Es por ello que, en la *Figura 2.4*, se puede observar en mayor profundidad las diferentes bandas de frecuencia y su atribución a los diferentes servicios a nivel nacional.

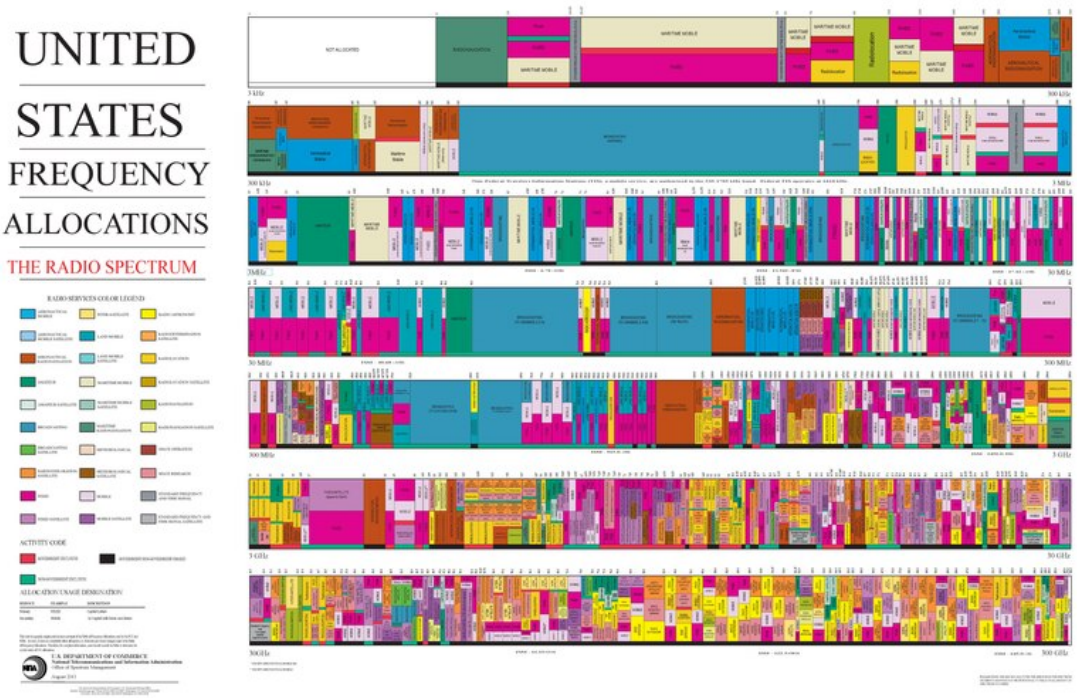

Figura 2.4 Atribución de frecuencias a nivel nacional [21].

3 Fundamentos de Software Defined Radio (SDR)

Las radios definidas por software o *Software Defined Radio* (SDR), son dispositivos que utilizan algoritmos de procesamiento digital para el tratamiento de señales de radiofrecuencia y el procesamiento de señal digital, *Digital Signal Processing* (DSP).

El receptor SDR ideal consiste en un hardware *front-end*, antenas y muestreadores, lo que le permite digitalizar una amplia banda de radiofrecuencia.

El *front-end* recibe la señal de radio desde la antena, la convierte a banda base, la digitaliza y envía muestras de la señal a través de una interfaz USB para extraer la información contenida en la señal.

A nivel conceptual, el SDR consta de una sección de radiofrecuencia (antena, amplificadores y filtros), convertidores Analógico-Digital (ADC) o Digital-Analógico (DAC), y un DSP y/o sistemas de computación conectados a un *Digital Front-End*. La *Figura 3.1* muestra un esquema general de una SDR como transmisor y receptor (transceptor).

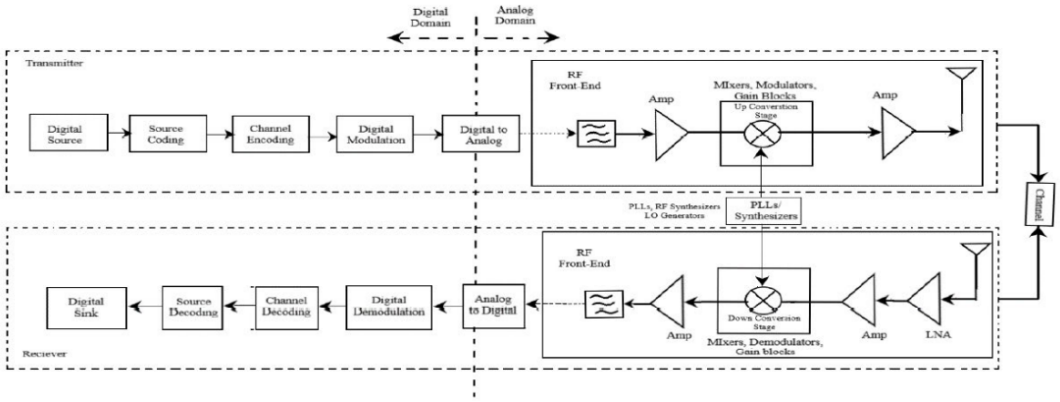

Figura 3.1 Sistema de un Transceptor SDR [8].

Historia y evolución

Atendiendo a [9], la primera generación de SDR apareció a mediados de los años 90. En estos dispositivos, la parte analógica de la arquitectura radioeléctrica convierte las señales de RF a FI (Frecuencia Intermedia) mediante un *Local Oscillator* (LO), y posteriormente un segundo LO convierte la señal de frecuencia intermedia a banda base. La señal en banda base es muestreada y digitalizada utilizando un ADC con frecuencia de muestreo de algunos kilohercios (véase la *Figura 3.2*).

La siguiente generación, surgida en los años 2000, comenzó el proceso de muestreo y digitalización directamente en frecuencias intermedias. La primera etapa del DSP utilizó un *Direct Digital Downconverter* (DDC) para trasladar la señal de frecuencia intermedia a banda base, y luego se aplicó un diezmado. La segunda fase de conversión se realiza en el dominio digital, a diferencia de la primera generación. En esta etapa, gracias al avance de los ADC/DAC, se implementaron más funcionalidades digitales, lo que incrementó

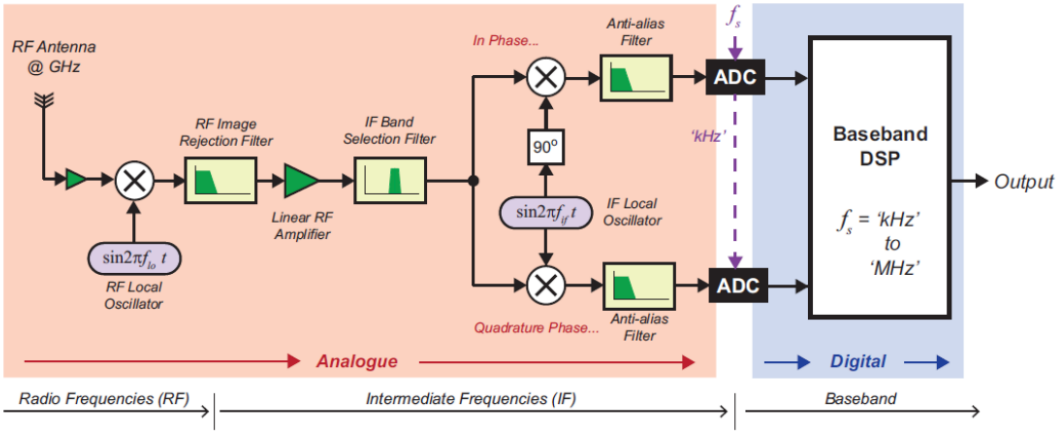

Figura 3.2 Diagrama de bloques de un SDR de primera generación [3].

la flexibilidad de las SDR. La *Figura 3.3* muestra un diagrama de bloques de esta segunda generación, en la que la parte digital adquiere una importancia mucho mayor.

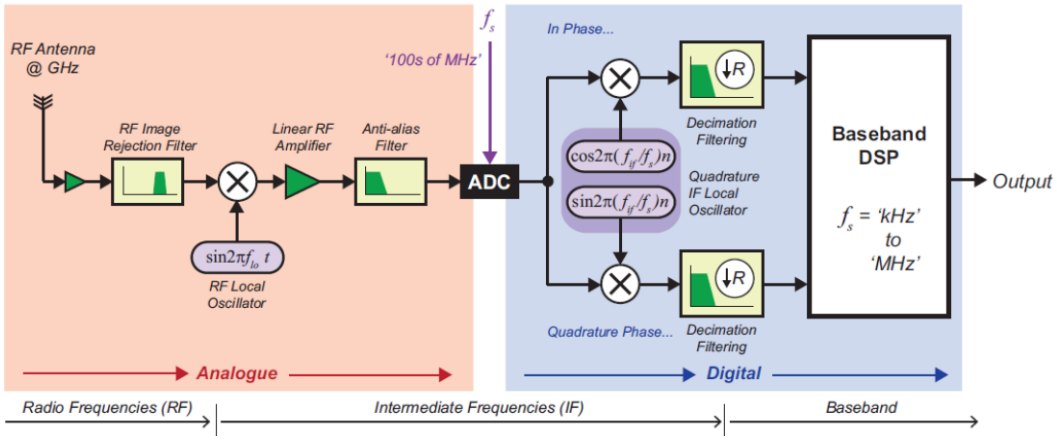

Figura 3.3 Diagrama de bloques de un SDR de segunda generación [3].

En la última generación de SDR, se ha intentado muestrear directamente las señales de RF después de pasar por un amplificador y un filtro anti-aliasing. Una vez digitalizada, la señal se traslada desde RF a banda base en una sola etapa mediante un único DDC, y posteriormente se aplica diezmado. La *Figura 3.4* ilustra cómo solo una pequeña parte del proceso permanece en el dominio analógico, siendo el procesamiento digital el componente fundamental.

Como se puede observar, la arquitectura de una SDR implica la implementación de la funcionalidad de *Baseband* DSP en MATLAB/Simulink o Python, lo que proporciona una gran flexibilidad en el desarrollo de la parte software. El uso de estas herramientas permite diseñar distintos sistemas capaces de recibir señales que utilizan diversos estándares de radio.

3.1 ADALM-Pluto SDR

Según [11] y [12], el ADALM-Pluto SDR (Software Defined Radio) es un dispositivo desarrollado por Analog Devices que permite transmitir y recibir señales de radio en un amplio rango de frecuencias, utilizando procesamiento digital en lugar de circuitos analógicos tradicionales. Gracias a su arquitectura flexible y programable, es ideal para la investigación, desarrollo y aprendizaje de sistemas de comunicación inalámbrica.

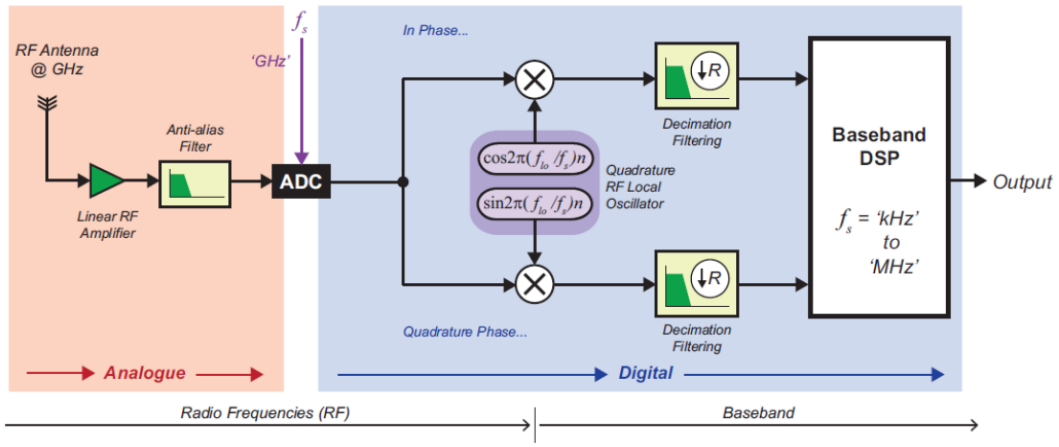

Figura 3.4 Diagrama de bloques de un SDR de tercera generación [3].

3.1.1 Características principales

El ADALM-Pluto SDR destaca por su versatilidad y portabilidad. A continuación, se describen sus características más relevantes:

1. **Diseño basado en el ADALM-Pluto:** Este modelo toma como base el diseño original de Analog Devices y, en muchos casos, incorpora mejoras tanto en hardware como en capacidad de procesamiento.
2. **Amplia gama de frecuencias de operación:** Puede operar en un espectro que abarca desde 70 MHz hasta 6 GHz. Algunos modelos modificados o versiones mejoradas permiten incluso extender este rango mediante ajustes de firmware.
3. **Soporte de transmisión y recepción simultánea (Full Duplex):** Esta funcionalidad permite emitir y captar señales al mismo tiempo, facilitando la implementación de sistemas de comunicación bidireccional en tiempo real.
4. **Conectividad por USB y red:** El dispositivo se conecta principalmente a través de USB, aunque ciertas versiones incluyen conectividad Ethernet, lo cual permite integrarlo en sistemas distribuidos o accesibles remotamente.
5. **Compatibilidad con software de análisis y diseño de señales:** Se integra fácilmente con entornos como GNU Radio, MATLAB y Cubic SDR, lo cual facilita el diseño, simulación y prueba de sistemas de procesamiento digital de señales (DSP).

Diferencias con el ADALM-Pluto original

Existen algunas diferencias clave entre el modelo básico de ADALM-Pluto y sus versiones extendidas o clones:

- **Rango de frecuencias:** Las versiones modificadas suelen ampliar la cobertura de frecuencia más allá de los límites estándar del modelo original.
- **Hardware mejorado:** Algunos modelos incluyen componentes con mejor desempeño, como amplificadores, filtros o módulos de alimentación más estables.

3.1.2 Módulo de aprendizaje

El ADALM-Pluto está diseñado también como una herramienta educativa, proporcionando una plataforma accesible para experimentar con señales de RF. Su módulo de aprendizaje permite transmitir y recibir señales analógicas dentro de un rango específico de frecuencias, facilitando el estudio práctico de conceptos de telecomunicaciones (veáse la *Figura 3.5*).

Figura 3.5 ADALM-Pluto SDR [12].

Características

Este módulo incluye una serie de especificaciones que lo hacen ideal para la formación académica y la investigación técnica:

- Dispositivo portátil y autónomo para la experimentación de señales RF.
- Cobertura de frecuencias de 325 MHz a 3,8 GHz.
- Convertidores ADC y DAC de 12 bits con capacidad de muestreo flexible.
- Conectores SMA hembra de 50 Ω para transmisión y recepción.
- Soporte para operación en dúplex medio o completo.
- Interoperabilidad con MATLAB y Simulink.
- Bloques integrables en GNU Radio para diseño por bloques.
- Biblioteca Libiio compatible con C, C++, C# y Python.
- Conexión mediante interfaz USB 2.0.
- Estructura ligera con carcasa plástica.
- Alimentación eléctrica por USB.
- Ancho de banda instantáneo de hasta 20 MHz para señales I/Q complejas.

3.1.3 Hardware

A continuación se describen los principales componentes de hardware que conforman el ADALM-Pluto SDR. Cada uno juega un papel fundamental en la operación del dispositivo:

1. **USB 2.0 PHY (Host):** Se encarga de la comunicación entre el dispositivo y la computadora anfitriona mediante el puerto USB.
2. **Micron DDR3L:** Memoria dinámica de bajo consumo que actúa como almacenamiento temporal para datos durante el procesamiento de señales digitales.
3. **Xilinx Zynq-7000:** Sistema en chip (SoC) que integra un procesador ARM Cortex-A9 y una FPGA, siendo el núcleo computacional del dispositivo.
4. **Micron QSPI Flash:** Memoria no volátil destinada al almacenamiento del firmware y configuraciones esenciales.
5. **Analog Devices AD936x:** Transceptor RF altamente integrado que realiza las funciones de conversión de frecuencia:
    - **Recepción (RX):** Transforma señales de radiofrecuencia a señales digitales de banda base.

- **Transmisión (TX):** Realiza la conversión inversa desde señales digitales de banda base hacia RF.

6. **Analog Devices Power:** Módulo de alimentación y regulación que distribuye energía de manera eficiente, reduciendo interferencias y asegurando el funcionamiento estable del sistema.

La *Figura 3.6* muestra todos los componentes de hardware descritos anteriormente que componen este SDR.

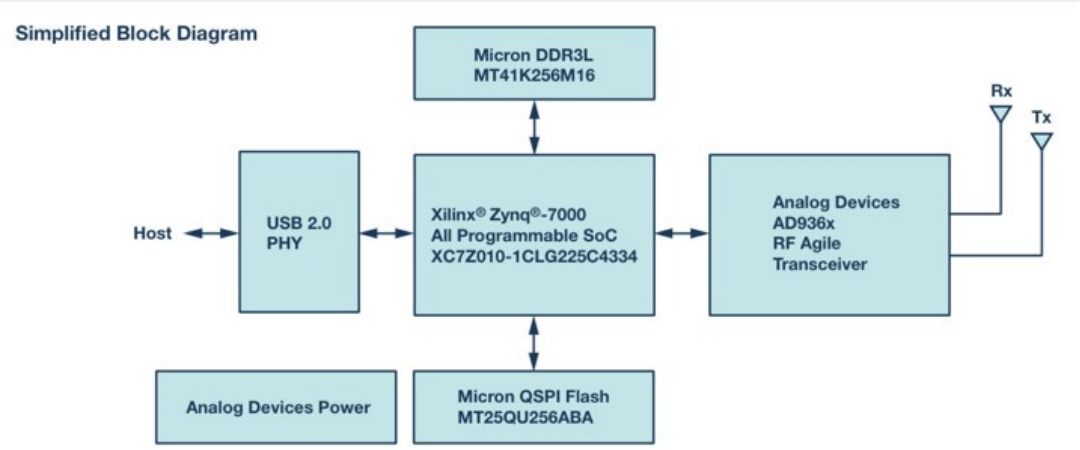

Figura 3.6 Diagrama de bloques del ADALM-Pluto SDR [13].

3.1.4 Lenguajes de programación

Una de las principales ventajas del ADALM-Pluto SDR es su capacidad para ser controlado y programado mediante diferentes lenguajes y entornos. Esto permite adaptarlo tanto a proyectos profesionales como a ejercicios académicos o personales.

1. MATLAB y Simulink

MATLAB, junto con Simulink, ofrece un paquete de soporte específico denominado `Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio`. Este paquete facilita la conexión, configuración y control del SDR directamente desde el entorno de MATLAB. Con él se pueden diseñar y simular sistemas de comunicación inalámbrica de manera gráfica o programática, ideal para pruebas rápidas y validaciones conceptuales.

2. GNU Radio

GNU Radio es un entorno de código abierto orientado al desarrollo de aplicaciones SDR mediante el uso de bloques visuales. El ADALM-Pluto puede integrarse fácilmente utilizando los bloques dedicados disponibles dentro del entorno. Esto permite definir flujos de señal complejos de manera modular, facilitando el diseño iterativo de transmisores, receptores y filtros digitales.

3. Python / C / C++

Gracias a la biblioteca `Libiio`, el ADALM-Pluto puede controlarse mediante scripts escritos en Python, así como en lenguajes compilados como C o C++. Esta librería proporciona una API[1] que facilita la interacción directa con el dispositivo, permitiendo configurar parámetros como frecuencia, ganancia, ancho de banda, y transmitir/recibir datos en tiempo real.

3.2 Ventajas de las SDR

Como se describe en [14], las principales ventajas de las radios definidas por software incluyen:

- **Flexibilidad:** Las funciones de radio pueden cambiarse o actualizarse mediante software, sin necesidad de modificar el hardware.

[1] Una API (Interfaz de Programación de Aplicaciones) actúa como un puente entre software y hardware, ocultando detalles técnicos de bajo nivel y ofreciendo una serie de funciones y estructuras que simplifican el desarrollo de aplicaciones que interactúan con el SDR.

- **Multiestándar:** Una misma plataforma puede utilizarse para múltiples protocolos de comunicación (Wi-Fi, LTE, Bluetooth, etc.).
- **Reducción de costos:** La reutilización de hardware para diferentes aplicaciones permite una disminución significativa del costo total del sistema.
- **Adaptabilidad:** Las SDR pueden ajustarse dinámicamente a cambios en el entorno, como la congestión del espectro o condiciones de canal adversas.
- **Portabilidad del diseño:** Una vez desarrollado un sistema SDR, puede migrarse fácilmente entre distintas plataformas de hardware.

3.3 Aplicaciones de las SDR

Las SDR son utilizadas en una amplia gama de aplicaciones:

- **Comunicaciones móviles:** Permiten el desarrollo y prueba de nuevos estándares como 5G.
- **Sistemas de radar:** Su flexibilidad permite la implementación de distintos tipos de radares sobre una misma plataforma.
- **Radioastronomía:** Facilitan la captura y análisis de señales débiles en rangos de frecuencia específicos.
- **Defensa y seguridad:** Las SDR permiten la escucha y transmisión en múltiples bandas, útiles para inteligencia de señales (SIGINT) y guerra electrónica (EW).
- **Radioaficionados y educación:** Ofrecen una plataforma accesible para aprender y experimentar con sistemas de comunicación.

3.3.1 Implementación de SDR en FPGAs

Las FPGAs son especialmente adecuadas para la implementación de SDR debido a:

- Su capacidad de paralelismo masivo, que permite procesar múltiples canales o etapas de señal en tiempo real.
- Su baja latencia, importante para aplicaciones como radar o comunicaciones críticas.
- La posibilidad de integrar bloques de procesamiento con interfaces de alta velocidad (por ejemplo, JESD204B para datos desde el ADC).

Una implementación típica en FPGA incluye módulos como:

- Conversión digital de frecuencia (DDC/DUC)
- Filtros FIR
- Moduladores/demoduladores digitales
- Controladores de ganancia automáticos (AGC)

3.4 Conclusión

El enfoque basado en SDR representa un cambio significativo en la forma en que se diseñan los sistemas de comunicación modernos. Gracias a plataformas como el ADALM-Pluto o el Zynq UltraScale+ RFSoC, es posible diseñar, probar y desplegar sistemas de radio de alto rendimiento en tiempos reducidos y con un alto grado de personalización.

4 Experimentos

Esta sección presenta experimentos para evaluar el proceso de transmisión y recepción de diferentes señales. Primero, se realizó una transmisión de sonido simple o tono para garantizar el funcionamiento adecuado del sistema de radio (SDR). Las pruebas más avanzadas se extendieron posteriormente mediante pruebas de señales moduladas, usando la modulación por amplitud de pulsos (PAM), la modulación por cambio de fase (PSK) y la modulación de amplitud de cuadratura (QAM).

Cada experimento consistió en la generación de señales paso banda, su transmisión por un transmisor SDR, su recepción y su posterior modulación. Se prestó especial atención a parámetros como las distorsiones introducidas durante la transmisión y el rendimiento del esquema de modulación en diversas condiciones. Los resultados obtenidos nos permiten comparar el comportamiento de las diferentes modulaciones, evaluar la eficiencia espectral y examinar los efectos de factores como EVM, NMSE y el BER en cada una de las modulaciones.

A modo de resumen, en la *Figura 4.1* se presentan recopilados todos los experimentos que se van a realizar con el Pluto SDR, con el fin de que, de una manera más visual, se puedan ver a simple vista todos ellos.

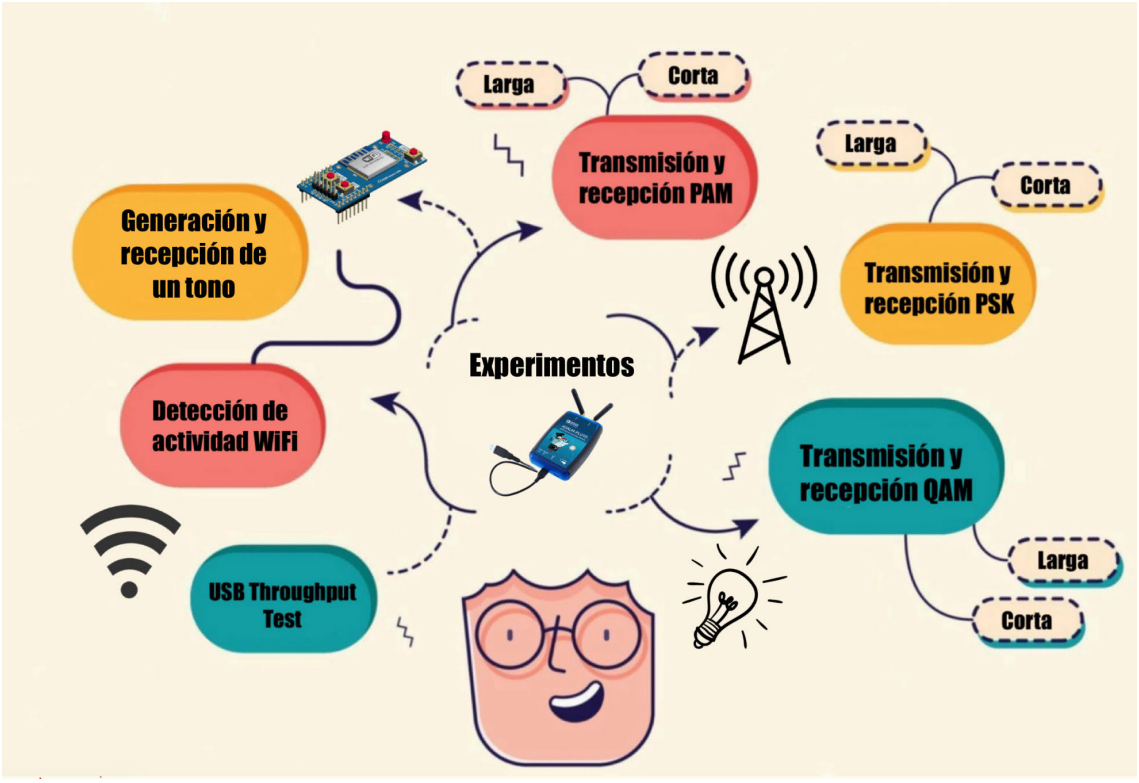

Figura 4.1 Recopilación de experimentos a describir.

4.1 Inicialización del ADALM PLuto SDR. Configuración de parámetros básicos

4.1.1 Inicialización y configuración

Se procede a analizar los parámetros de inicialización del dispositivo, que serán comunes en cada uno de los experimentos que se realizarán a continuación.

```
from adi import Pluto
sdr = Pluto(uri="ip:192.168.2.1")
```

Se importa la librería `Pluto` del paquete `adi`, que permite controlar el dispositivo PlutoSDR. La instancia `sdr` se inicializa con la URI de red correspondiente al dispositivo conectado (`192.168.2.1`). Esto establece la comunicación entre el Software (Python) y el Hardware (PlutoSDR). Otra posibilidad, es establecer dicha comunicación mediante la configuración del puerto USB.

4.1.2 Parámetros de transmisión y recepción

```
fs = int(20e6)            # Frecuencia de muestreo: 20 MHz
tx_freq = int(500e6)      # Frecuencia de transmisión: 500 MHz
rx_freq = int(500e6)      # Frecuencia de recepción: 500 MHz
tone_freq = 1e6           # Frecuencia del tono a transmitir: 1 MHz
num_samples = 4096        # Número total de muestras a transmitir/recibir
```

Estos parámetros definen el entorno de la transmisión, siendo los parámetros configurables de este experimento:

- `fs`: tasa de muestreo que determina cuántas muestras por segundo se procesan.
- `tx_freq` y `rx_freq`: frecuencias del oscilador local para transmisión y recepción, respectivamente.
- `tone_freq`: frecuencia de la señal senoidal que se va a generar y transmitir.
- `num_samples`: cantidad de datos generados para una sola transmisión o recepción.

4.1.3 Configuración del canal de transmisión

```
sdr.tx_lo = tx_freq
sdr.tx_rf_bandwidth = int(2e6)
sdr.tx_cyclic_buffer = True
sdr.tx_hardwaregain_chan0 = -10
sdr.sample_rate = fs
```

Aquí se configuran los parámetros del canal de transmisión:

- `tx_lo`: frecuencia central del canal de transmisión.
- `tx_rf_bandwidth`: ancho de banda del filtro de transmisión.
- `tx_cyclic_buffer`: permite enviar continuamente la misma señal de forma cíclica.
- `tx_hardwaregain_chan0`: ganancia aplicada al canal de transmisión.
- `sample_rate`: se asegura que tanto TX como RX compartan la misma tasa de muestreo.

4.2 Transmisión y recepción de un tono

4.2.1 Objetivo del experimento

Este primer experimento tiene como fin la generación de un tono o señal senoidal compleja (I/Q) a una frecuencia de 1 MHz, la cual se transmite de forma continua sobre una portadora de 500 MHz. De forma simultánea, el propio dispositivo está configurado para recibir datos en la misma frecuencia y analizar el

espectro de la señal recibida, cortocircuitando el transmisor y el receptor del dispositivo con un cable, a fin de evitar ruido externo y otras señales no deseadas.

En esta prueba inicial se pretende lograr una serie de objetivos dispuestos a continuación:

- La fidelidad de la cadena de transmisión, asegurando que la transmisión y la recepción sean prácticamente idénticas.
- El comportamiento espectral del sistema completo, a fin de que el espectro de la señal recibida esté centrado en la frecuencia deseada.

Dado que el tono de 1 MHz se modula sobre la portadora centrada en 500 MHz, se espera que el espectro de la señal recibida presente un pico centrado en 501 MHz. Este valor corresponde a la suma de la frecuencia de portadora y la frecuencia del tono.

4.2.2 Configuración del entorno y parámetros

Para la ejecución del experimento se establece una conexión directa con el PlutoSDR a través de su interfaz IP o USB. Los parámetros técnicos utilizados son los siguientes:

- **Frecuencia de transmisión**: 500 MHz
- **Frecuencia de recepción)**: 500 MHz
- **Frecuencia de muestreo**: 20 MHz
- **Frecuencia del tono**: 1 MHz
- **Número de muestras**: 4096
- **Ganancia de transmisión**: -50 dB
- **Ganancia de recepción**: 0 dB

Cabe destacar que se ha escogido una frecuencia de muestreo adecuada (20 MHz) para observar correctamente el espectro alrededor de la frecuencia portadora, incluyendo el tono de 1 MHz.

4.2.3 Generación y transmisión de la señal

La señal transmitida se genera digitalmente como una señal senoidal compleja (I/Q), mediante la ecuación:

$$\texttt{tx\_signal} = 2^{14} \cdot e^{j2\pi \cdot \texttt{tone\_freq} \cdot t} \tag{4.1}$$

Esta expresión genera una señal de frecuencia 1 MHz. Al ser una señal compleja se evita el fenómeno de *aliasing* y se permite el desplazamiento directo de frecuencia tras la modulación con la portadora.

La transmisión se realiza en modo cíclico (`cyclic_buffer = True`), lo que implica que el Pluto SDR transmite continuamente el mismo paquete de muestras. Este modo resulta especialmente útil para señales periódicas como la usada en esta prueba. Sin embargo, una vez acabada la transmisión, es recomendable destruir el *buffer* de transmisión para asegurar una correcta eliminación de residuos de transmisión.

```
t = np.arange(num_samples) / fs
tx_signal = np.exp(2j * np.pi * tone_freq * t)
tx_signal *= windows.hann(num_samples)
sdr.tx(tx_signal)
```

Se crea una señal senoidal compleja a 1 MHz:

- `t`: vector temporal.
- `tx_signal`: señal generada con exponencial compleja.
- Se aplica una ventana de Hann para reducir efectos espectrales no deseados.
- Finalmente, la señal es enviada a través del PlutoSDR con `sdr.tx()`.

4.2.4 Recepción y procesado de la señal

La cadena de recepción se configura con los mismos parámetros de frecuencia y tasa de muestreo que la transmisión, centrando la señal en 500 MHz, y capturándose así, 4096 muestras. Para asegurar que los datos sean válidos, se descartan las primeras capturas, lo que permite limpiar el *buffer* interno del dispositivo.

```
sdr.rx_lo = rx_freq
sdr.rx_rf_bandwidth = int(2e6)
sdr.rx_buffer_size = num_samples
sdr.rx_hardwaregain_chan0 = 10

rx_signal = sdr.rx()
```

Se configuran parámetros similares en recepción:

- `rx_lo`: frecuencia central del canal de recepción.
- `rx_rf_bandwidth`: ancho de banda del receptor.
- `rx_buffer_size`: tamaño del buffer que se llenará con muestras.
- `rx_hardwaregain_chan0`: ganancia aplicada al canal de recepción.
- `rx_signal = sdr.rx()`: se capturan las muestras recibidas.

Posteriormente, se aplica una Transformada Rápida de Fourier (FFT) a la señal recibida para analizar su contenido espectral, siguiendo la siguiente expresión:

$$X(f) = \texttt{fftshift}\left(\left|\texttt{fft}(\texttt{rx\_signal})\right|/N\right) \tag{4.2}$$

donde N es el número de muestras.

El eje de frecuencias se ajusta en MHz y se centra correctamente respecto a la frecuencia de muestreo y la frecuencia de recepción:

$$f = \texttt{fftshift}\left(\texttt{fftfreq}(N, d = 1/fs)\right) \cdot 10^{-6} + \left(\texttt{rx\_freq} \cdot 10^{-6}\right) \tag{4.3}$$

Este ajuste es crucial para una interpretación precisa del resultado en el dominio de la frecuencia.

4.3 Transmisión y recepción de señal PAM paso banda

Este segundo experimento tiene como objetivo la transmisión y recepción de una señal PAM (Pulse Amplitude Modulation) en paso banda utilizando el sistema PlutoSDR. A diferencia del experimento anterior basado en una señal senoidal, en este caso se transmite una secuencia digital modulada en amplitud mediante una forma de onda portadora. El objetivo principal es comprobar la integridad de la señal recibida y analizar su forma temporal y espectral, así como los posibles efectos del canal (ruido, desfase, atenuación, etc.).

4.3.1 Transmisión corta

Generación de la señal PAM paso banda

La señal transmitida se genera a partir de una cadena de bits aleatoria, de 394 bits concretamente, a la que se antepone un preámbulo conocido para facilitar la sincronización en recepción. Los bits se agrupan en símbolos para una modulación M-PAM (en este caso, M=8).

Para adecuar la señal a una transmisión paso banda, se utiliza un pulso conformador basado en un coseno a la frecuencia central f_0.

La señal resultante es una forma de onda en paso banda, que se transmite utilizando la interfaz de transmisión del PlutoSDR.

En este experimento se ha implementado un esquema de doble modulación, compuesto por dos etapas sucesivas:

1. **Modulación en amplitud de pulsos (PAM)**: La primera etapa consiste en una modulación digital en amplitud, en la que los datos binarios se agrupan en símbolos según el orden de modulación M. A cada

símbolo se le asigna un nivel de amplitud diferente, centrado en cero. Así, se obtiene una señal discreta de amplitud variable:

$$a_k = 2s_k - (M-1)$$

donde s_k es el valor decimal del símbolo k-ésimo. Esta señal representa la versión en banda base de la información digital.

2. **Modulación paso banda (coseno a frecuencia f_0)**: Dado que la señal PAM es de banda base y no puede ser radiada directamente a través de una antena, es necesario trasladar su espectro a una frecuencia portadora. Para ello, se multiplica la señal por una onda cosenoidal de frecuencia f_0:

$$s(t) = \sum_k a_k \cdot p(t-kT) \cdot \cos(2\pi f_0 t)$$

donde:

- a_k son los símbolos modulados en amplitud.
- $p(t)$ es la forma de pulso utilizada (por ejemplo, rectangular o coseno alzado).
- T es el período de símbolo.
- f_0 es la frecuencia de la portadora.

Esta operación genera una señal continua en el tiempo, centrada en la frecuencia f_0, que es apta para la transmisión mediante hardware de radiofrecuencia.

La modulación PAM, por sí sola, genera una señal de banda base no adecuada para ser emitida por un sistema SDR. Es por ello que una segunda modulación traslada el espectro a paso banda, permitiendo la transmisión efectiva por un canal físico. Esta estrategia permite mantener la simplicidad de una modulación digital lineal, a la vez que se garantiza la compatibilidad con sistemas reales de comunicación por radio.

Transmisión y recepción de la señal modulada

La señal generada se transmite a través del PlutoSDR, configurando adecuadamente la frecuencia de portadora, la tasa de muestreo y la ganancia.

Durante la recepción, se captura una muestra suficientemente larga para garantizar que contiene toda la señal transmitida, incluyendo el preámbulo.

Posteriormente, se normaliza la señal recibida y se realiza una correlación cruzada con la señal transmitida para detectar el desfase temporal y alinear ambas señales. En sistemas reales, factores como el retardo del canal, el *buffering* del dispositivo receptor o interferencias externas pueden provocar que la señal recibida esté desfasada respecto a la transmitida. Para resolver esto, se ha implementado un mecanismo de sincronización por correlación cruzada, que consiste en lo siguiente:

- **Diseño de un preámbulo conocido:** Antes de transmitir los datos, se genera un fragmento de señal determinista (por ejemplo, una secuencia alternante de amplitudes), cuya forma es conocida tanto por el transmisor como por el receptor.
- **Correlación cruzada en recepción:** Una vez capturada la señal recibida, se realiza una correlación cruzada entre esta y la señal de referencia (el preámbulo transmitido).
- **Alineamiento de la señal recibida:** Se localiza dónde se alcanza el máximo de la correlación, y se ajusta la señal recibida a partir de ese punto. Esto permite que los símbolos PAM estén correctamente situados en el tiempo para su posterior demodulación.

4.3.2 Transmisión larga

Generación de la señal PAM paso banda

En este experimento se ha realizado una transmisión de mayor envergadura, compuesta por un total de 10244 bits, agrupados en tandas de 394 bits cada una. A cada tanda, al igual que en la Subsección 4.3.1, se le antepone un preámbulo fijo que facilita la sincronización en recepción.

Al igual que en el experimento anterior, se emplea una modulación 8-PAM, y para adaptar la señal a una transmisión paso banda, se utiliza un pulso conformador basado en un coseno de frecuencia central f_0.

La señal resultante es una onda en paso banda continua, generada mediante un esquema de doble modulación, siguiendo las mismas directrices que en la Subsección 4.3.1.

Transmisión y recepción de la señal modulada

Se ha transmitido la señal por tandas, recibiendo y almacenando cada una de ellas. Cada señal recibida ha sido normalizada, sincronizada mediante correlación cruzada con el preámbulo conocido, y corregida en fase si era necesario.

Este proceso se ha repetido para las 26 tandas que componen la transmisión total, alineando y concatenando los fragmentos para obtener la señal completa recibida.

4.4 Transmisión y recepción de señal PSK paso banda

4.4.1 Transmisión corta

Generación de la señal PSK paso banda

La señal transmitida se vuelve a construir a partir de una secuencia aleatoria de 394 bits, precedida por un preámbulo. Estos bits se agrupan en símbolos según el esquema de modulación M-PSK, siendo en este experimento $M = 8$, es decir, modulación 8-PSK.

Cada símbolo se asocia a una fase distinta de una portadora, lo cual permite codificar tres bits por símbolo.

La señal modulada en fase se convierte en paso banda mediante la multiplicación por un coseno de frecuencia f_0, de forma análoga al caso de la PAM. El objetivo es desplazar el espectro de la señal a una frecuencia adecuada para su transmisión a través de radiofrecuencia.

Transmisión y recepción de la señal modulada

La señal generada se transmite mediante el PlutoSDR, configurando los parámetros clave: frecuencia central f_0, tasa de muestreo, ganancia de transmisión, y ancho de banda del canal.

Durante la recepción, se captura una ventana temporal amplia que asegure contener el preámbulo y la totalidad del mensaje. Luego, se lleva a cabo una normalización de amplitud y sincronización temporal usando correlación cruzada y sincronización de fase con el preámbulo transmitido.

Tras la sincronización, se realiza una demodulación coherente recuperando la fase de cada símbolo y asignando los bits correspondientes. Finalmente, se comparan los bits recuperados con los transmitidos para determinar el rendimiento del experimento.

4.4.2 Transmisión larga

Generación de la señal 8-PSK paso banda

En este experimento se ha realizado una transmisión de gran volumen al igual que en la Subsección 4.3.2 utilizando modulación 8-PSK. Se han transmitido un total de 10244 bits, divididos en tandas de 394 bits. A cada tanda se le antepone un preámbulo fijo de 32 bits, utilizado para facilitar la sincronización en recepción.

La señal se genera mediante un esquema de doble modulación:

1. **Modulación 8-PSK:** Los bits se agrupan en símbolos de 3 bits (dado que $M = 8$), y se mapean a puntos de una constelación 8-PSK ideal. Cada símbolo se representa como un número complejo de módulo constante, con fase determinada por:

$$s_k = \sqrt{E_b} \cdot e^{j\frac{2\pi m_k}{M}}, \quad m_k \in \{0, 1, ..., 7\}$$

   donde m_k es el valor decimal del símbolo k-ésimo, y $E_b = 8$ es la energía por bit.

2. **Modulación paso banda:** La señal base compleja se convierte en una señal real utilizando pulsos conformadores ortogonales $g_1(t)$ y $g_2(t)$, definidos mediante cosenos y senos de frecuencia f_0. Estos se ponderan con las partes real e imaginaria de cada símbolo:

$$X_n(t) = \sum_k \Re(s_k) \cdot g_1(t - kT) + \Im(s_k) \cdot g_2(t - kT)$$

   Posteriormente, se escala por $\sqrt{1/T_m}$ para mantener la energía total de la señal, y se genera la señal temporal a ser transmitida.

Transmisión y recepción de la señal modulada

La transmisión se realiza en modo cíclico desde el buffer del PlutoSDR a una frecuencia central de 915 MHz, con una tasa de muestreo de 20 MHz y un ancho de banda de transmisión de 8 MHz.

En la recepción, la señal se normaliza y se sincroniza utilizando correlación cruzada con el preámbulo conocido. Para afinar la sincronización, se estima el desfase con interpolación cuadrática del pico de correlación. Además, se evalúa si existe una inversión de fase, corrigiéndola si se detecta.

Cada tanda recibida se sincroniza y, tras procesar todas las tandas, se concatenan para obtener la señal completa.

4.5 Transmisión y recepción de señal QAM paso banda

4.5.1 Transmisión corta

Generación de la señal QAM paso banda

La señal transmitida se ha generado a partir de una secuencia aleatoria de 394 bits, precedida de un preámbulo conocido. Estos bits se han agrupado en símbolos mediante el esquema de modulación 16-QAM, es decir, modulación en amplitud en cuadratura con $M = 16$, codificando 4 bits por símbolo. A diferencia de la modulación 8-PSK, que representa los bits mediante variaciones exclusivamente en la fase, la 16-QAM combina variaciones tanto en fase como en amplitud, permitiendo una mayor eficiencia espectral.

La señal modulada en cuadratura se ha convertido a paso banda multiplicándola por una portadora de frecuencia f_0, desplazando así su contenido espectral hacia la banda adecuada para la transmisión por radiofrecuencia. Esta conversión es análoga a la realizada en el experimento con 8-PSK.

A diferencia de las anteriores modulaciones, se ha tenido que realizar un cambio en los parámetros de la modulación, ya que el valor de M ahora es 16 en lugar de 8. Es por ello, además, que se ha modificado el número de muestras, duplicándolas en este caso para obtener una mayor resolución de la señal.

Transmisión y recepción de la señal modulada

La señal generada se ha transmitido mediante el dispositivo PlutoSDR, configurando los parámetros clave: frecuencia central f_0, tasa de muestreo, ganancia de transmisión, y ancho de banda del canal. Para este experimento, se ha usado una frecuencia de transmisión de 915 MHz.

Durante la recepción, se ha capturado una ventana temporal lo suficientemente amplia para asegurar la inclusión del preámbulo y de todos los símbolos de datos. Posteriormente, la señal se ha normalizado en amplitud y se ha sincronizado temporalmente mediante correlación cruzada con el preámbulo transmitido. Para ajustar posibles desplazamientos de fase residuales, se ha llevado a cabo una estimación de la fase óptima, maximizando la coincidencia entre señal transmitida y recibida.

Una vez alineadas ambas señales, se ha realizado una demodulación coherente, extrayendo los símbolos QAM y decodificando los bits originales. Por último, se ha calculado el número de errores y se han evaluado métricas de calidad como el BER, NMSE y EVM.

4.5.2 Transmisión larga

Generación de la señal QAM paso banda

En este experimento se ha realizado una transmisión de mayor envergadura, compuesta por un total de 10244 bits, agrupados en tandas de 394 bits cada una. A cada tanda, al igual que en la Subsección 4.5.1, se le antepone un preámbulo fijo que facilita la sincronización en recepción.

Al igual que en el experimento anterior, se emplea una modulación 16-QAM, y para adaptar la señal a una transmisión paso banda, se utiliza un pulso conformador basado en un coseno de frecuencia central f_0.

Transmisión y recepción de la señal modulada

Se ha transmitido la señal por tandas, recibiendo y almacenando cada una de ellas. Cada señal recibida ha sido normalizada, sincronizada mediante correlación cruzada con el preámbulo conocido, y corregida en fase si era necesario.

Este proceso se ha repetido para las 26 tandas que componen la transmisión total, alineando y concatenando los fragmentos para obtener la señal completa recibida.

4.6 Transmisión y recepción con dos dispositivos

4.6.1 Configuración de dos ADALM Pluto SDR

En este caso, se utilizan dos PlutoSDR conectados a través de direcciones IP distintas. Uno se dedica exclusivamente a la transmisión y el otro a la recepción. A continuación, se muestra la configuración correspondiente:

```
from adi import Pluto

# Inicializar el Pluto para transmisión
sdr_tx = Pluto(uri="ip:192.168.2.1")
sdr_tx.tx_lo = int(500e6)
sdr_tx.tx_rf_bandwidth = int(2e6)
sdr_tx.tx_cyclic_buffer = True
sdr_tx.tx_hardwaregain_chan0 = -10
sdr_tx.sample_rate = int(20e6)

# Inicializar el Pluto para recepción
sdr_rx = Pluto(uri="ip:192.168.2.2")
sdr_rx.rx_lo = int(500e6)
sdr_rx.rx_rf_bandwidth = int(2e6)
sdr_rx.rx_buffer_size = 4096
sdr_rx.rx_hardwaregain_chan0 = 10
sdr_rx.sample_rate = int(20e6)
```

Cada objeto (`sdr_tx` y `sdr_rx`) gestiona un dispositivo independiente. Se asegura que ambos compartan la misma frecuencia y tasa de muestreo para garantizar coherencia en la transmisión y recepción.

4.6.2 Transmisión y recepción de un tono

Este experimento consiste en la transmisión de un tono o señal senoidal, igual que la Sección 4.2, excepto por la diferencia de que ahora se utilizan dos dispositivos PlutoSDR, conectados individualmente mediante interfaces IP distintas. Uno de los dispositivos se encargará exclusivamente de transmitir la señal, mientras que el segundo recibirá dicha señal.

La configuración de ambos equipos se estableció a una frecuencia central de operación de 868 MHz, una frecuencia que para pruebas experimentales dentro del rango de frecuencias permitido, transmitiendo y recibiendo con sus antenas, en lugar de estar transmisor y receptor cortocircuitado por un cable.

La frecuencia de muestreo y sus otros parámetros han sido mantenidos con respecto al experimento inicial. Esta señal, también, se transmitió de forma cíclica, permitiendo al receptor capturar múltiples repeticiones de la misma sin pérdida de información.

Uno de los principales desafíos al utilizar dos dispositivos independientes para transmisión y recepción es la presencia de una deriva de fase, un fenómeno atribuido a la falta de sincronización entre los osciladores locales (LO) de cada dispositivo.

4.6.3 Transmisión y recepción de señal PAM paso banda

A continuación, se repite el experimento de transmisión de una señal PAM, pero utilizando una vez más, dos dispositivos distintos: uno para transmitir y otro para recibir. Ambos dispositivos fueron configurados con las mismas frecuencias de operación, pero al tratarse de módulos independientes, presentan los mismos inconvenientes observados previamente al transmitir una onda senoidal: al no compartir un reloj común, existe una desincronización de los osciladores locales (LO) entre el transmisor y el receptor. Esto genera un desfase inicial aleatorio y una deriva de fase progresiva que afecta directamente la forma en que se reciben los símbolos.

4.7 Detección de actividad WiFi

El objetivo de este experimento es analizar el espectro del canal 1 de WiFi (centrado en 2.412 GHz utilizando el dispositivo PlutoSDR. El objetivo específico es detectar la actividad del punto de acceso WiFi encendido y apagado en periodos de cada 5 segundos, y evaluar si se observa una diferencia en el espectro capturado.

Para llevar esto a cabo la observación, se utilizó la librería `ESP8266WiFi.h`. Es una librería específica para el microcontrolador ESP8266, que permite al dispositivo usar su módulo WiFi integrado para conectarse a redes o actuar como punto de acceso.

La configuración del experimento fue la siguiente:

- Frecuencia central: 2.412 GHz.
- Frecuencia de muestreo: 20 Msps.
- Tamaño de buffer: 16 384 muestras.
- Ganancia de la antena receptora: 0 dB.
- Uso de ventana Blackman para reducir los lóbulos laterales y mejorar la forma del espectro.
- Promedio de 10 espectros para suavizar el ruido.

El espectro de frecuencia fue calculado mediante la Transformada Rápida de Fourier (FFT), centrada en la frecuencia configurada, y visualizado en el dominio de la frecuencia (MHz).

5 Análisis de resultados

En este capítulo se presentan y analizan los resultados obtenidos a partir de los diferentes experimentos realizados, empleando distintos esquemas de modulación. como PAM, PSK y QAM.

Los análisis se centran en aspectos como la fidelidad de la señal recibida, la constelación obtenida, la eficiencia espectral y los errores introducidos en el proceso de modulación-demodulación. Asimismo, se discuten posibles limitaciones observadas durante la implementación práctica, como interferencias, desajustes en la sincronización o problemas asociados a la tasa de muestreo, tratados en apartados posteriores como inconvenientes.

5.1 Generación y tranmisión de un tono

En el dominio temporal, la forma de onda transmitida mantiene su forma senoidal tras la recepción, lo que indica una transmisión correcta (véase la *Figura 5.1 y Figura 5.2*). No obstante, se observan distorsiones muy leves, atribuibles a:

- No linealidades propias del hardware del PlutoSDR.
- Imperfecciones en el mezclador I/Q interno.

El resultado más representativo se observa en el espectro de frecuencia de la señal recibida, representado en la *Figura 5.3*. Tal y como se anticipaba, aparece un pico claramente definido en 501 MHz, que corresponde a la suma entre la frecuencia de la portadora (500 MHz) y la del tono (1 MHz). Esto valida que el sistema de transmisión y recepción opera adecuadamente.

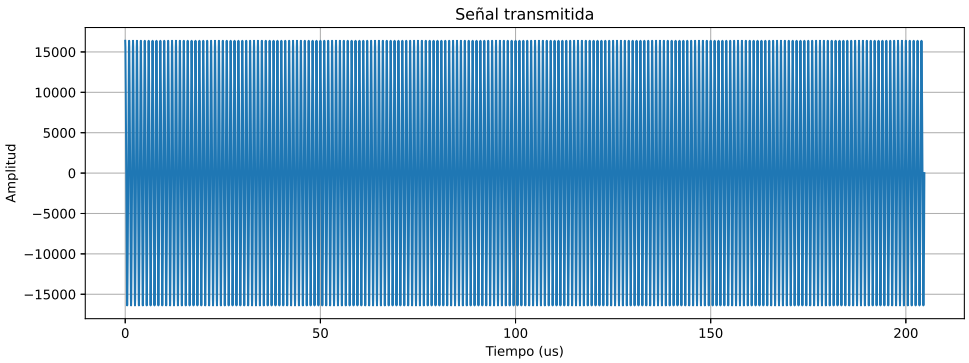

Figura 5.1 Señal senoidal transmitida.

5.2 Evaluación de la tasa de muestreo efectiva en PlutoSDR

La finalidad de este apartado es medir la tasa de muestreo efectiva del puerto USB al que se ha conectado el PlutoSDR, en condiciones reales de recepción. El dispositivo se puede configurar para poder funcionar con

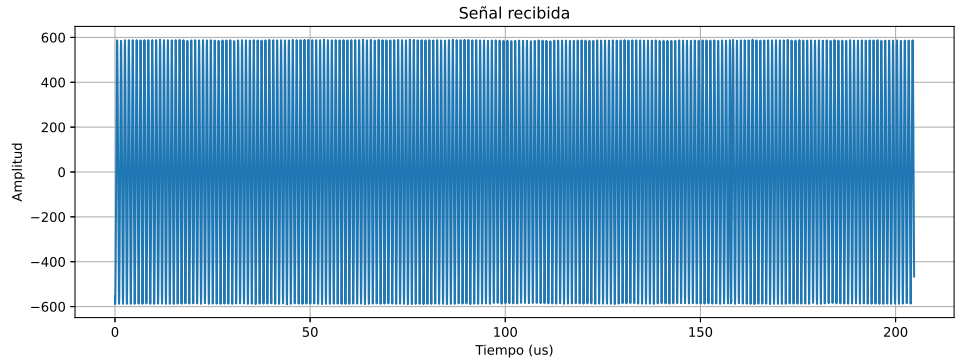

Figura 5.2 Señal senoidal recibida.

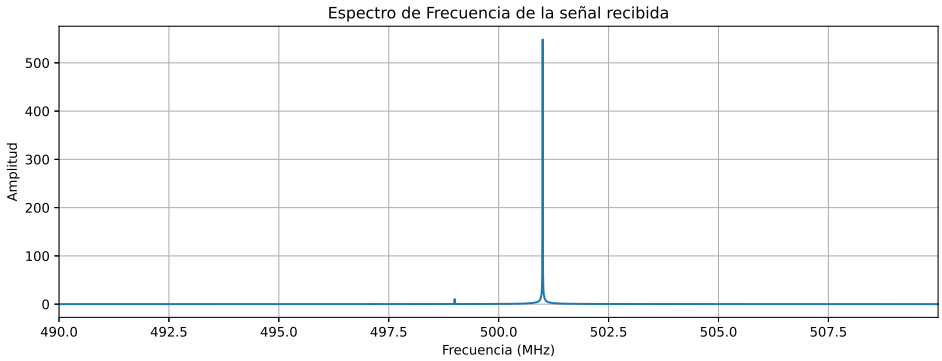

Figura 5.3 Espectro de la señal senoidal recibida.

una tasa de muestreo de, por ejemplo, 3 MS/s. Sin embargo, realmente, diversos factores como el tamaño del *buffer*, y el rendimiento del sistema pueden afectar la cantidad real de muestras procesadas por segundo.

El objetivo de este experimento es realizar una serie de recepciones utilizando el PlutoSDR, midiendo el tiempo total que tarda en completar todas. A partir de esto, se calcula la tasa de muestreo efectiva, es decir, cuántas muestras por segundo se están recibiendo realmente.

Extrapolando esta idea a la captura de señales de drones, es fundamental conocer la tasa de muestreo efectiva por varias razones:

- **Resolución temporal**: Si la tasa efectiva es demasiado baja, se pueden perder detalles fundamentales de la señal, como transiciones rápidas.
- **Ancho de banda**: La tasa de muestreo determina el ancho de banda, según el teorema de Nyquist. Si la tasa efectiva es más baja de lo esperado, podríamos no capturar completamente la señal del dron o perder componentes espectrales clave.
- **Sincronización y demodulación**: Muchos sistemas de drones utilizan protocolos de modulación complejos. Para demodular correctamente estas señales, la tasa de muestreo efectiva ha de ser conocida.

Por lo tanto, el objetivo principal de este experimento es lograr una tasa de muestreo efectiva que sea lo más cercana posible a la frecuencia de muestreo establecida por configuración. Es decir, buscamos que la cantidad de muestras que realmente se adquieren por segundo en el sistema sea prácticamente igual a la tasa configurada en el dispositivo PlutoSDR.

Para ello, se ha seguido el siguiente método:

- Se ha ido incrementando el valor de la frecuencia de muestreo progresivamente hasta obtener que la tasa de muestreo efectiva se alejaba de la dicha frecuencia
- A la par, se ha ido aumentando el tamaño del *buffer*, con el fin de recuperar la igualdad entre ambos.

Resultados

Usando un equilibrio entre ambos procedimientos de la metodología descrita, se han obtenido los resultados recopilados en la *Tabla 5.1*.

Tabla 5.1 Valores de la tasa de muestreo efectiva.

Métrica	Valor
Tiempo transcurrido	0.5537 segundos
Muestras recibidas	1638400
Frecuencia de muestreo	3×10^6 Hz
Tasa de muestreo efectiva	2959142.12 muestras/segundo

5.3 Análisis de la transmisión y recepción de una señal PAM paso banda

5.3.1 Transmisión corta

A continuación, se presentan las gráficas más relevantes que permiten analizar el comportamiento de la transmisión de la señal PAM en paso banda.

En la *Figura 5.4 y Figura 5.5* se pueden observar las dos señales, la transmitida y la recibida, en las que cada color representa un símbolo asociado. Si superponemos ambas señales para su comparación (véase la *Figura 5.6*), a simple vista, no se aprecia una diferencia muy notoria. Sin embargo, en la *Figura 5.7* se ha representado con aspas las diferencias entre ambas señales. Por tanto, se puede ver que la señal recibida mantiene la forma general de la transmitida, aunque con cierto nivel de distorsión y atenuación.

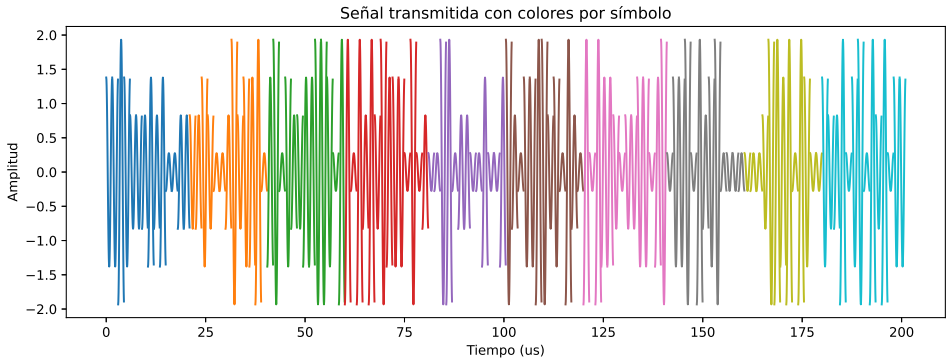

Figura 5.4 Señal 8-PAM transmitida.

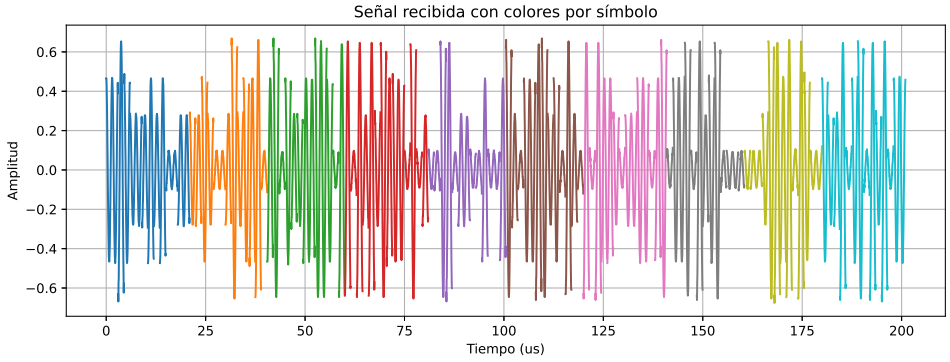

Figura 5.5 Señal 8-PAM recibida.

Además, para una mayor visualización de la comparativa, se ha representado en la *Figura 5.8 y Figura 5.9* los 8 primeros símbolos transmitidos y recibidos, respectivamente.

En relación a los espectros de frecuencia, se disponen éstos a continuacíón, en la *Figura 5.10, Figura 5.11 y Figura 5.12*, mediante el uso de la transformada de Fourier, como con el método Welch. Se puede apreciar una atenuación en la recepción posiblemente debida a una falta de ganancia del amplificador de transmisión o por pérdidas por el cable conductor.

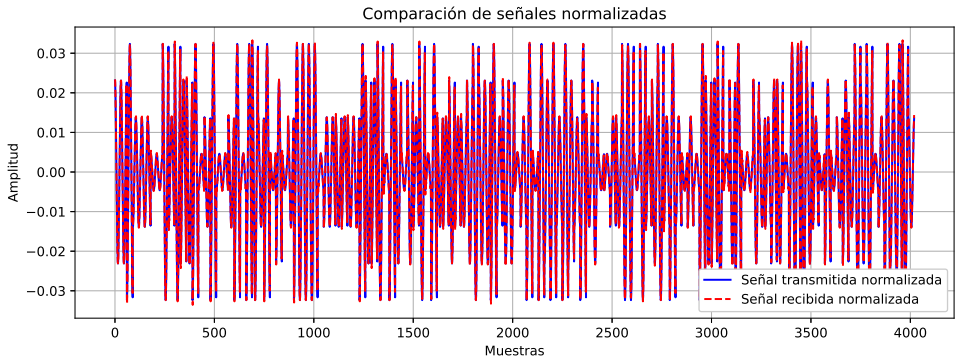

Figura 5.6 Comparación de señales normalizadas.

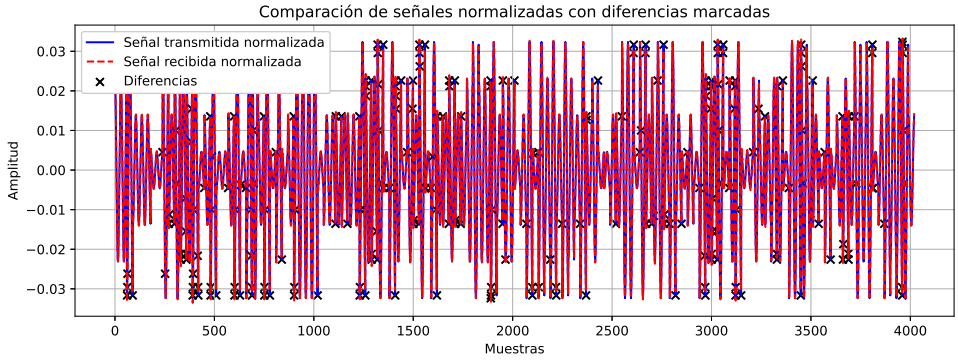

Figura 5.7 Comparación de señales normalizadas con markers.

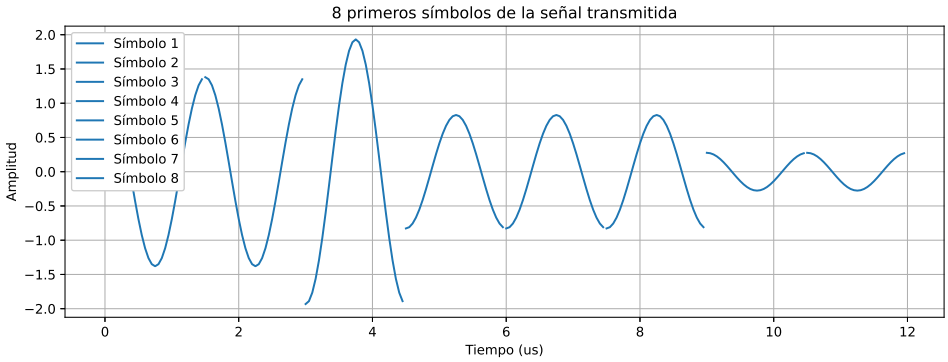

Figura 5.8 8 Primeros símbolos transmitidos.

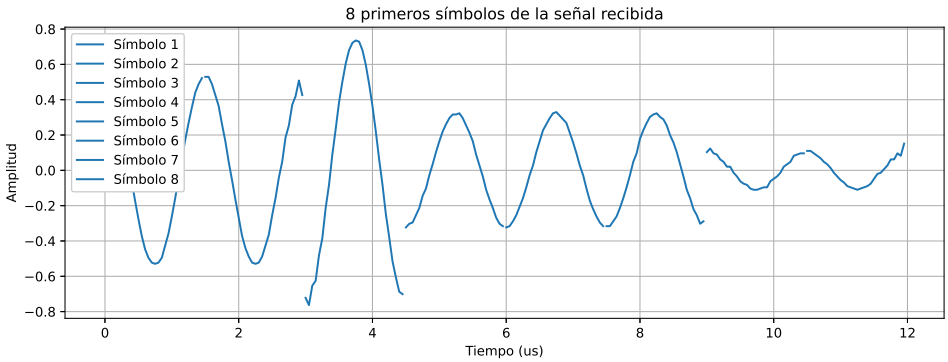

Figura 5.9 8 Primeros símbolos recibidos.

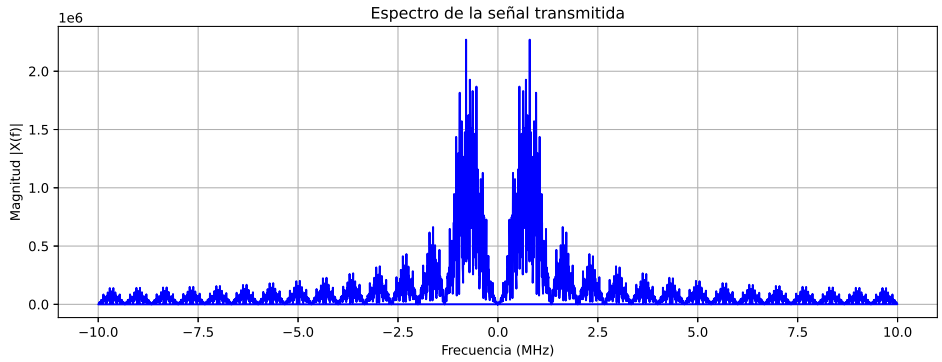

Figura 5.10 Espectro de la señal transmitida (FFT).

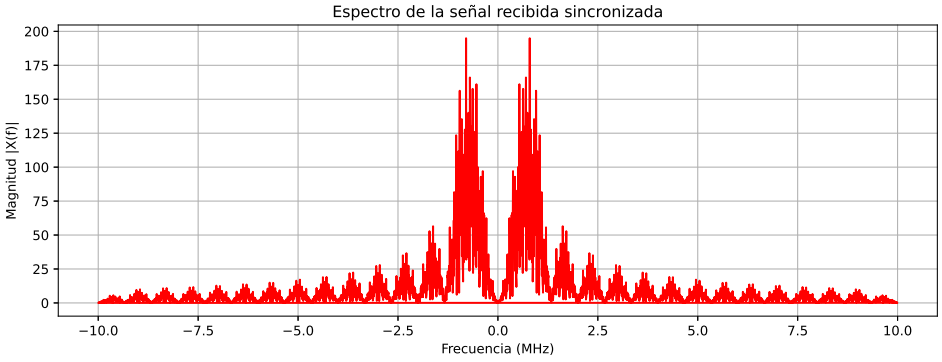

Figura 5.11 Espectro de la señal recibida (FFT).

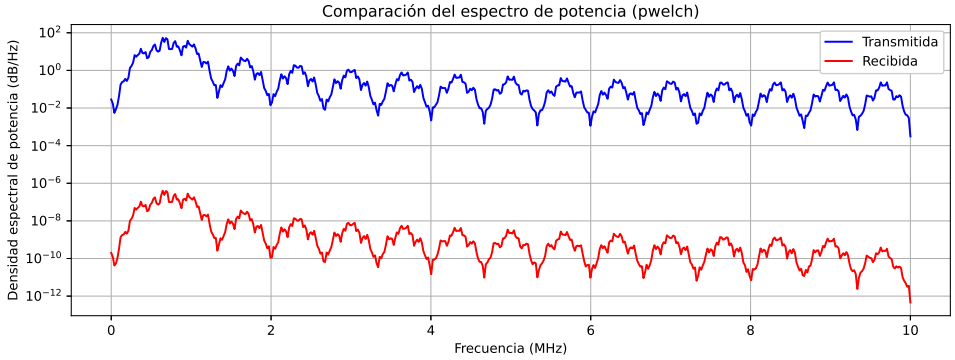

Figura 5.12 Espectro de la señal transmitida y recibida (WELCH).

Por último se han representado la constelación de bits transmitida y recibida, con motivo del posterior cálculo del EVM y del BER. Es por ello que la *Figura 5.13* muestra dichas constelaciones, representadas en el eje real.

La transmisión de señales PAM en paso banda mediante el sistema PlutoSDR ha demostrado ser funcional y relativamente robusta frente a distorsiones. Se ha logrado una correcta sincronización utilizando un preámbulo y correlación cruzada.

Los valores recogido en la *Tabla 5.2* indican una transmisión de alta fidelidad. El valor de NMSE negativo refleja una baja diferencia de energía entre la señal transmitida y la recibida. La EVM, cercana al 0 %, sugiere una modulación eficiente con un error muy leve, mientras que una tasa de error de bits (BER) cercana a cero confirma una comunicación confiable y sin pérdidas significativas.

Se han observado efectos típicos de un canal real, como ruido aditivo, variación en la amplitud, y pequeñas alteraciones en la fase, aunque la estructura de la señal es perfectamente reconocible y útil para su posterior demodulación.

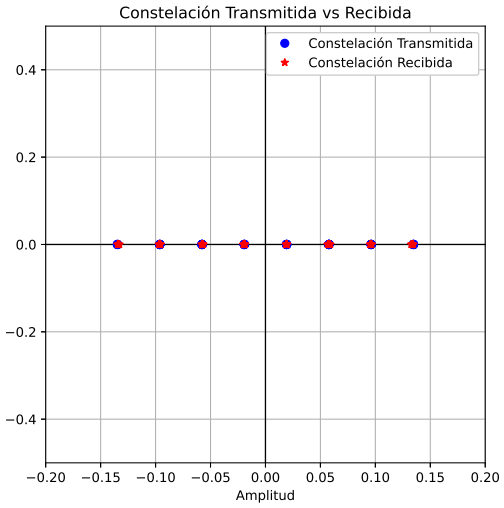

Figura 5.13 Constelación de símbolos transmitidos y recibidos (Transmisión corta).

Tabla 5.2 Valores representativos de la calidad de la transmisión (Transmisión corta).

Métrica	Valor
NMSE	−28.7 dB
EVM	0.50 %

5.3.2 Transmisión larga

Para evaluar el rendimiento de la transmisión, se ha calculado el NMSE (error cuadrático medio normalizado) entre la señal transmitida y la recibida. El resultado obtenido es de -19.02 dB, lo cual indica una buena similitud entre ambas señales pese a las inevitables distorsiones del canal.

En la *Figura 5.14* se han graficado los 8 primeros símbolos transmitidos y recibidos para mostrar la alineación temporal lograda y visualizar directamente el comportamiento de la modulación en las primeras posiciones.

Adicionalmente, se han representado las constelaciones de los primeros símbolos transmitidos y recibidos, observándose cómo la estructura original se conserva en gran medida, aunque con ligeras desviaciones en amplitud y posibles desplazamientos causados por el canal (véase la *Figura 5.15*).

Inconvenientes en la transmisión

Como se ha mencionado anteriormente, la señal puede verse modificada o alterada durante la transmisión y recepción. Para ilustrar este concepto, se presentan dos ejemplos para respaldar esta idea:

- En la *Figura 5.16* se puede apreciar que la señal recibida y la transmitida distan de ser iguales. Esto se debe a factores como el ruido aditivo, efectos del canal, falta de sincronización, o una mezcla de todos ellos. En consecuencia, aun a pesar del comportamiento oscilatorio intuido en la señal recibida, resulta bastante difícil detectar símbolos claros.
- En contraste, en la *Figura 5.17*, el inconveniente del ruido no aparece. A priori, puede parecer que la señal recibida pueda estar invertida con respecto a la transmitida. Sin embargo, atendiendo a los dos últimos símbolos, se puede apreciar que hay un error en la sincronización de la señal, e incluso un posible residuo de transmisión en el envío anterior.

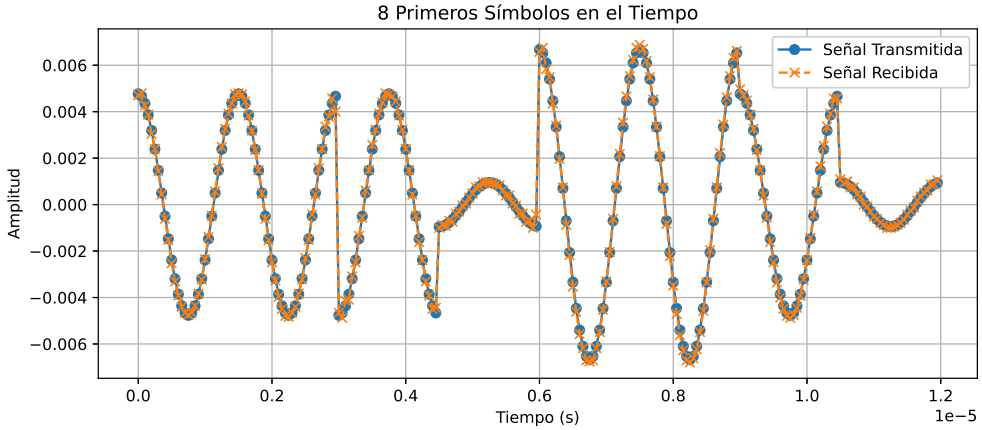

Figura 5.14 8 Primeros símbolos transmitidos y recibidos.

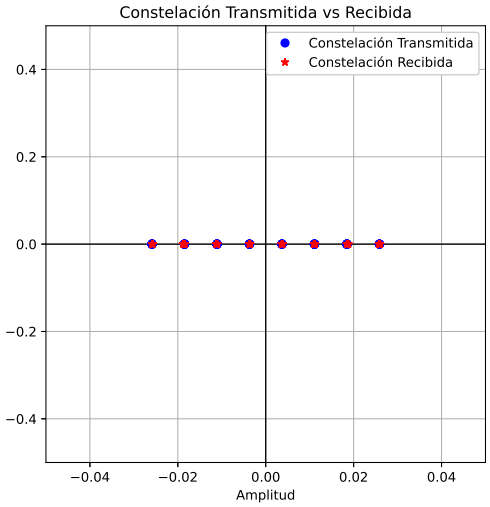

Figura 5.15 Constelación de símbolos transmitidos y recibidos (Transmisión larga).

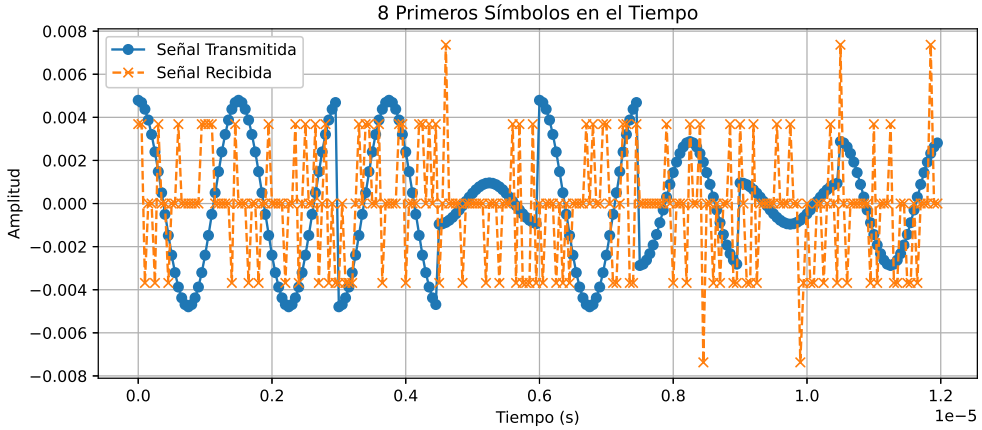

Figura 5.16 8 Primeros símbolos transmitidos y recibidos en caso de error.

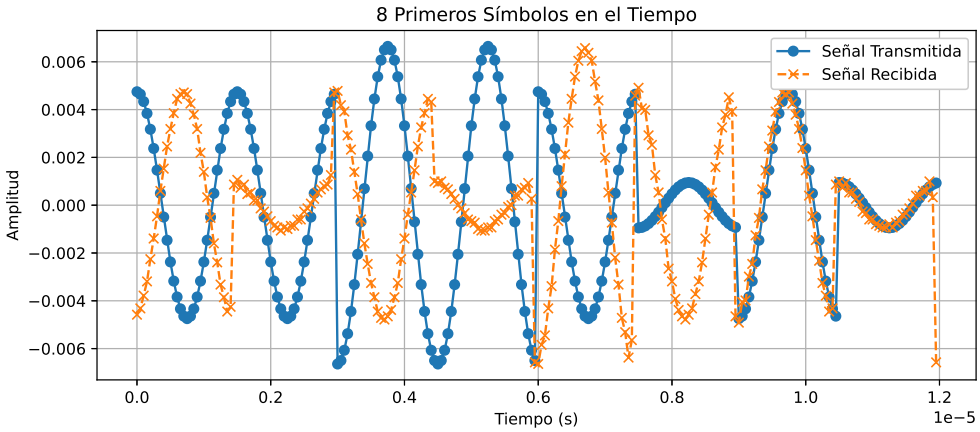

Figura 5.17 8 Primeros símbolos transmitidos y recibidos en caso de error.

El sistema de transmisión PAM en paso banda para un envío grande ha funcionado satisfactoriamente. Se ha demostrado la escalabilidad del esquema al transmitir más de 10.000 bits con una tasa de errores baja y buena sincronización temporal y en fase.

Los valores recogido en la *Tabla 5.3* indican una transmisión de alta fidelidad. El valor de NMSE negativo refleja una baja diferencia de energía entre la señal transmitida y la recibida. La EVM, por debajo del 5 %, sugiere una modulación eficiente con bajo error, mientras que una tasa de error de bits (BER) cercana a cero confirma una comunicación confiable y sin pérdidas significativas.

Tabla 5.3 Valores representativos de la calidad de la transmisión (Transmisión larga).

Métrica	Valor
NMSE	−19.0 dB
EVM	3.67 %

La comparación entre señales, junto con las constelaciones, refuerzan la idea de la validez del experimento. Las técnicas aplicadas (normalización, correlación cruzada, corrección de fase, filtrado) han resultado efectivas para mitigar las distorsiones típicas del canal.

5.4 Análisis de la transmisión y recepción de una señal PSK paso banda

5.4.1 Transmisión corta

A continuación, se representan las señales transmitida y recibida. En la *Figura 5.18* se puede ver la forma paso banda de la señal transmitida, mientras que en la *Figura 5.19* se observa la señal captada por el receptor, ya sincronizada. Posteriormente se ha superpuesto una sobre otra, para una mayor comparación (véase la *Figura 5.20*). Además, para recalcar esta idea, en la *Figura 5.21* se han marcado con aspas las diferencias entre ambas, puesto que a simple vista podría parecer que son muy parecidas.

La comparación entre ambas señales muestra una clara correspondencia en la estructura de la onda, aunque con la presencia de ruido y posibles ligeros desplazamientos de fase. Haciendo zoom en los primeros 8 símbolos, para tener una visión más nítida, se obtendría lo representado en la *Figura 5.22 y Figura 5.23*.

A nivel espectral, en la *Figura 5.24 y Figura 5.25*, se observa cómo la señal recibida presenta una leve atenuación respecto a la transmitida, por las atenuaciones y las pérdidas del canal mencionadas en secciones

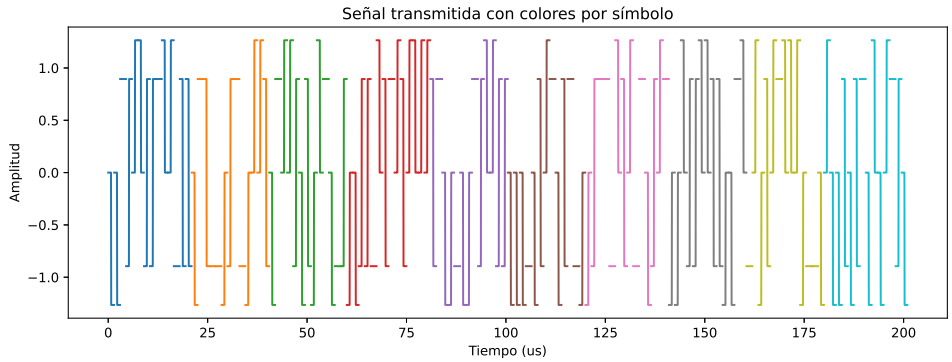

Figura 5.18 Señal 8-PSK transmitida.

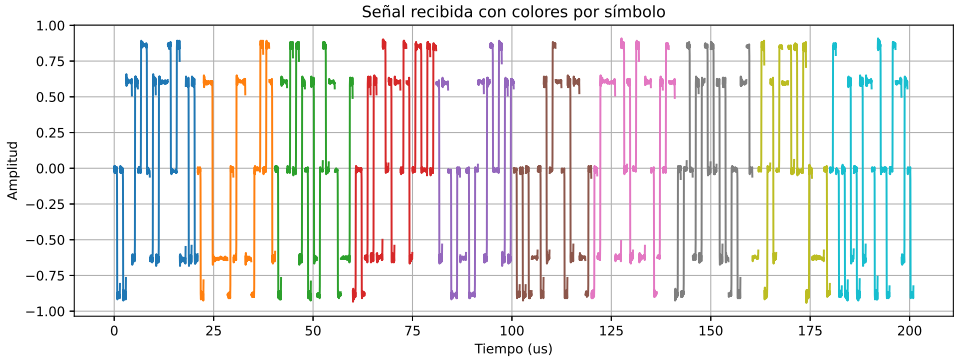

Figura 5.19 Señal 8-PSK recibida.

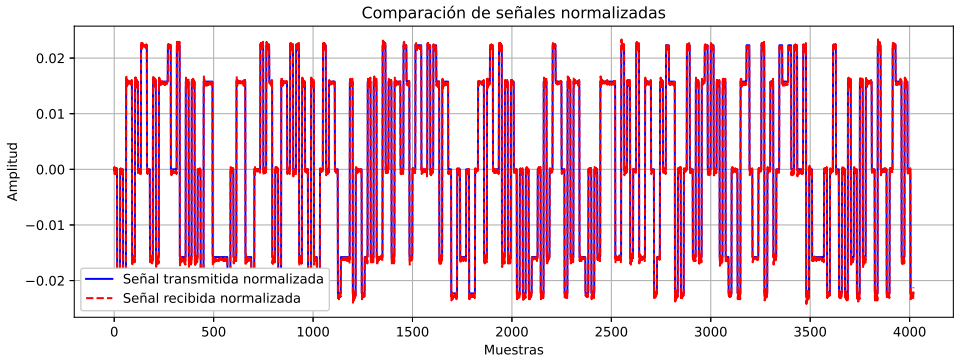

Figura 5.20 Comparación de las señales normalizadas.

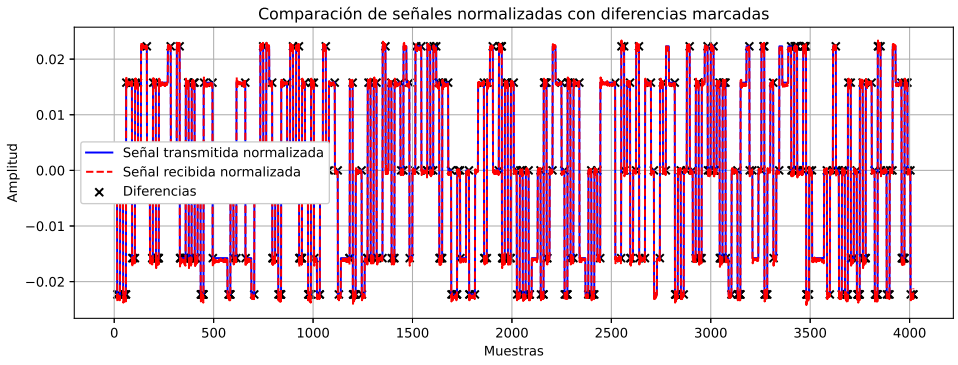

Figura 5.21 Comparación de las señales normalizadas con markers.

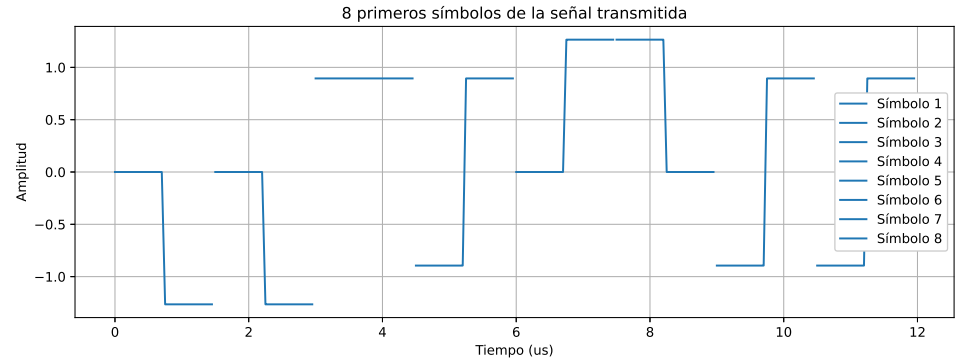

Figura 5.22 8 Primeros símbolos transmitidos.

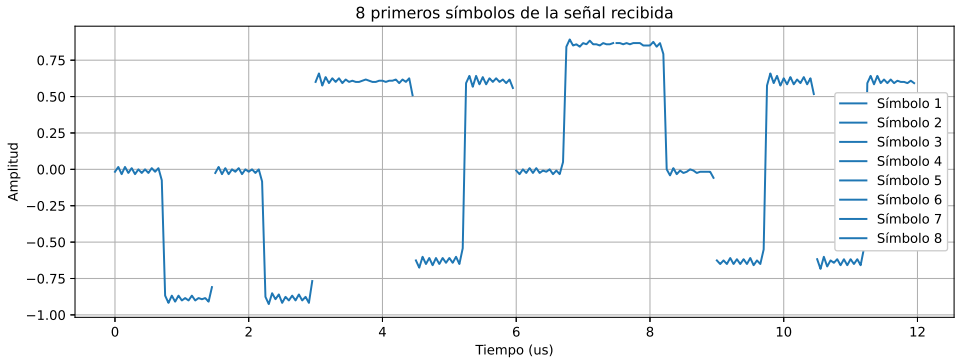

Figura 5.23 8 Primeros símbolos recibidos.

anteriores, pero mantiene la ocupación espectral típica de la 8-PSK. Adicionalmente, se dispone en la *Figura 5.12* la comparación de los espectros de potencia de ambas señales para incidir más sobre esta idea.

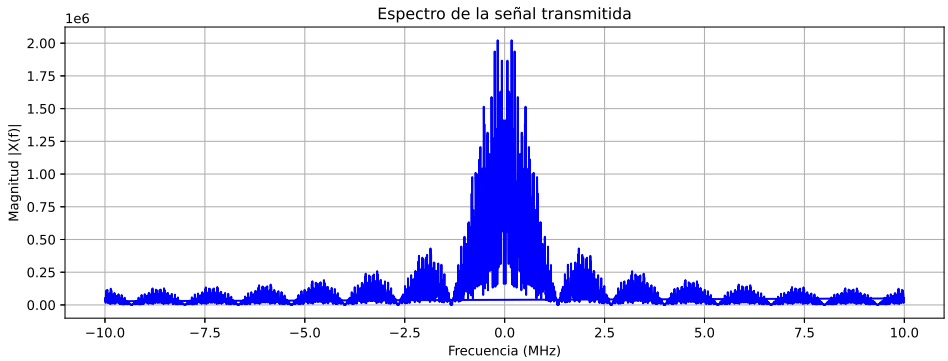

Figura 5.24 Espectro de la señal transmitida (FFT).

En la *Figura 5.27*, se representa la constelación recibida frente a la ideal. La dispersión en torno a cada punto ideal permite estimar la calidad del canal y calcular métricas como EVM.

El sistema de transmisión 8-PSK en paso banda ha demostrado ser efectivo y robusto, permitiendo una transmisión fiable con baja tasa de error. La sincronización mediante preámbulo y correlación cruzada permite recuperar la señal con precisión.

Las rendimientos obtenidos, indicadas en la *Tabla 5.4*, muestran una calidad elevada en la transmisión. La EVM es baja, el BER cercano a cero y el NMSE indica una buena coincidencia de energía entre señal transmitida y recibida.

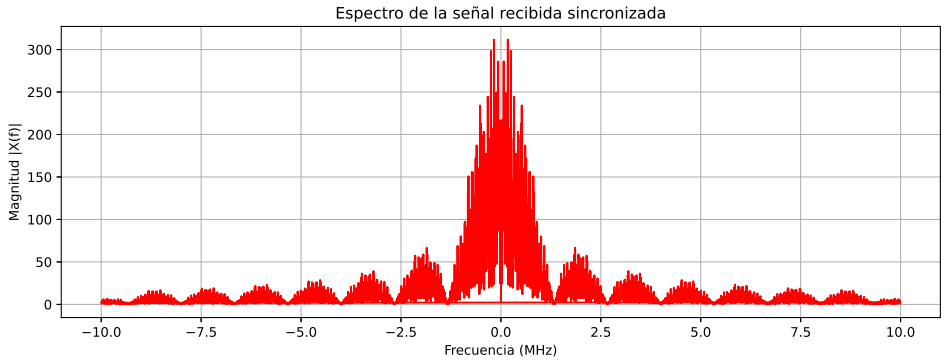

Figura 5.25 Espectro de la señal recibida (FFT).

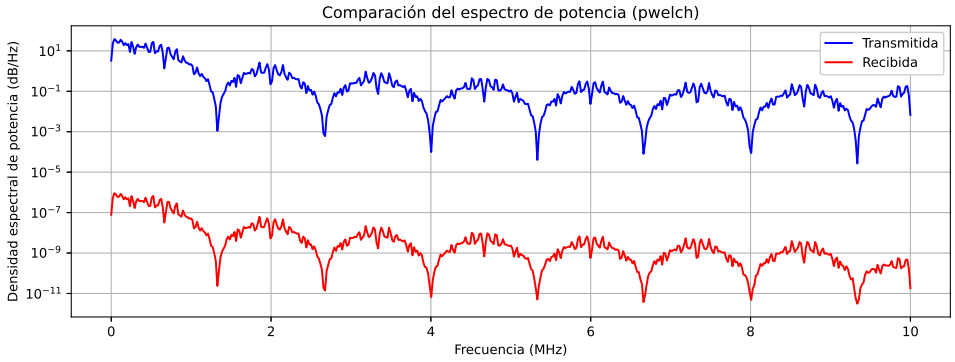

Figura 5.26 Espectro de la señal transmitida y recibida (WELCH).

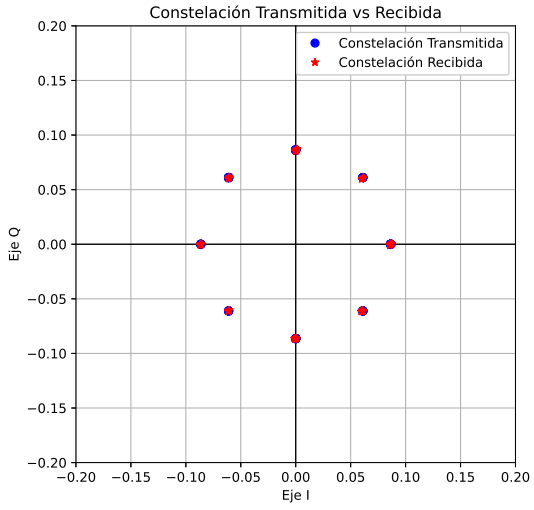

Figura 5.27 Constelación de símbolos transmitidos y recibidos (Transmisión corta).

5.4.2 Transmisión larga

Se ha procedido a calcular la diferencia energética entre la señal transmitida y la recibida una vez alineadas y normalizadas. El resultado obtenido es de -19.2 dB, lo que indica un buen parecido entre ambas señales, pese al ruido y las distorsiones que puedan aparecer. En apoyo a este resultado, se muestra en la *Figura 5.28* ambas señales transmitida y recibida superpuestas una sobre la otra.

Tabla 5.4 Valores representativos de la calidad de la transmisión (Transmisión corta).

Métrica	Valor
NMSE	−27.3 dB
EVM	1.56 %

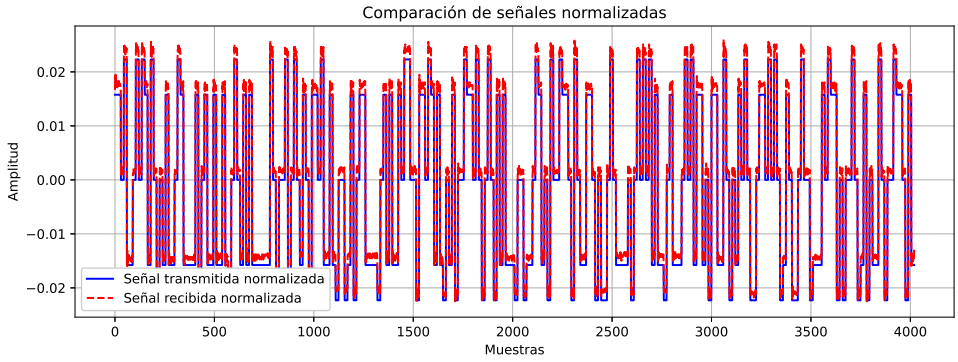

Figura 5.28 Comparación de las señales normalizadas.

En la *Figura 5.29* se han representado los 8 primeros símbolos transmitidos y recibidos para mostrar la sincronización lograda y visualizar directamente el comportamiento en las primeras posiciones.

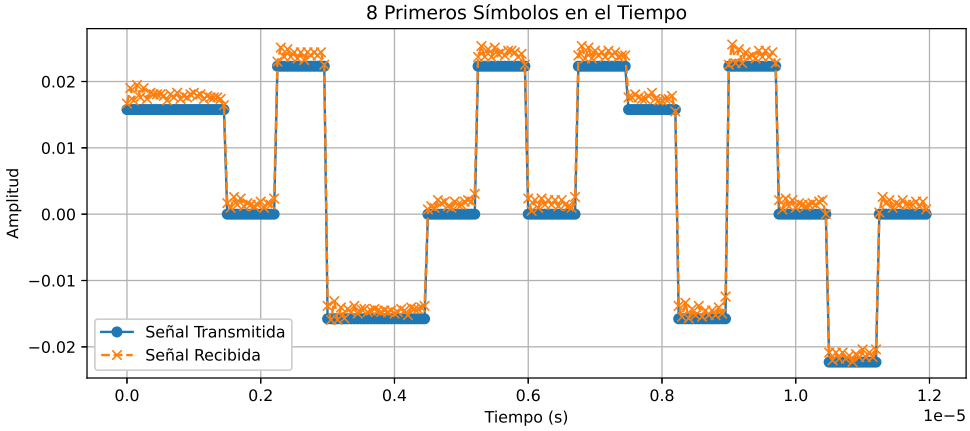

Figura 5.29 8 Primeros símbolos transmitidos y recibidos.

En la *Figura 5.30*, se observa la constelación de símbolos recibidos frente a la original. Se aprecia cómo la mayoría de los puntos se agrupan adecuadamente en torno a los centros ideales, aunque con ligeras distorsiones por ruido y efectos del canal.

Este experimento demuestra la capacidad del sistema de transmisión basado en modulación 8-PSK para manejar volúmenes grandes de datos con alta fiabilidad.

Los resultados que se recogen en la *Tabla 5.5* confirman el buen desempeño del sistema: una NMSE de aproximadamente -19.235 dB, una EVM cercana al 5 % y una BER cercana a cero.

Los métodos de preprocesado (normalización, sincronización por correlación, corrección de fase, filtrado y demodulación) han demostrado ser eficaces para mitigar las distorsiones del canal y recuperar la información

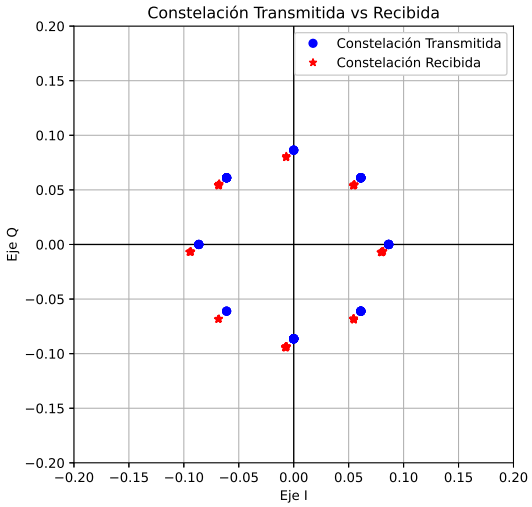

Figura 5.30 Constelación de símbolos transmitidos y recibidos (Transmisión larga).

Tabla 5.5 Valores representativos de la calidad de la transmisión (Transmisión larga).

Métrica	Valor
NMSE	−19.2 dB
EVM	7.29 %

con alta fiabilidad. Esto valida la solidez del sistema y su aplicabilidad en contextos reales de transmisión digital.

5.5 Análisis de la transmisión y recepción de una señal QAM paso banda

5.5.1 Transmisión corta

A continuación, se representan las señales transmitida y recibida. En la *Figura 5.31* se presenta la comparación de ambas señales tras su normalización. A pesar del ruido introducido por el canal y de posibles atenuaciones, se observa una coincidencia clara entre ambas formas de onda. Incidiendo en esto, se ha decidido poner unos *markers* en las diferencias entre ambas, a fin de compararlas en mayor profundidad (véase la *Figura 5.32*).

Para un análisis más detallado, en la *Figura 5.33* y *Figura 5.34* se presentan los ocho primeros símbolos, tanto en transmisión como en recepción.

En el dominio frecuencial, la *Figura 5.35* y *Figura 5.36* muestran los espectros de la señal transmitida y recibida, respectivamente, obtenidos mediante la Transformada de Fourier. Se aprecia una ligera atenuación en la señal recibida, aunque conserva el ancho de banda característico de una señal 16-QAM.

Adicionalmente, en la *Figura 5.37* se presenta la comparación de los espectros de potencia mediante el método de Welch.

Finalmente, en la *Figura 5.38* se representa la constelación de los símbolos recibidos, superpuesta a la ideal de 16-QAM. Se puede apreciar una ligera dispersión alrededor de los puntos ideales, lo cual refleja el efecto del canal sobre la señal y permite evaluar la calidad mediante métricas como EVM.

La transmisión de la señal 16-QAM en paso banda mediante PlutoSDR ha sido exitosa. La sincronización por preámbulo y correlación cruzada ha permitido una correcta recuperación de la señal. A pesar de la

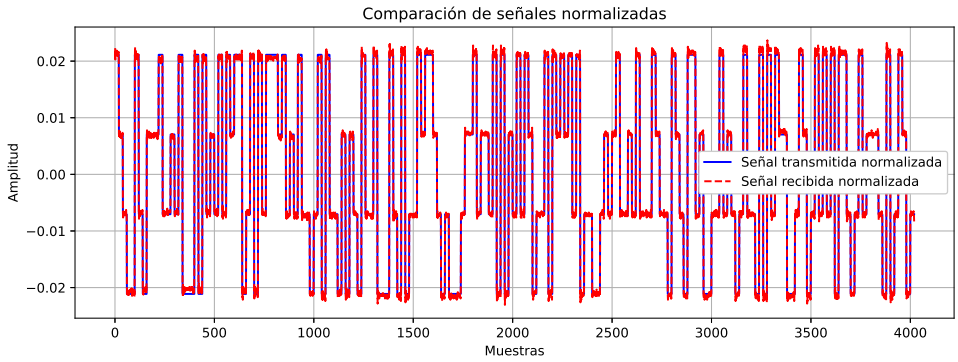

Figura 5.31 Comparación de las señales normalizadas.

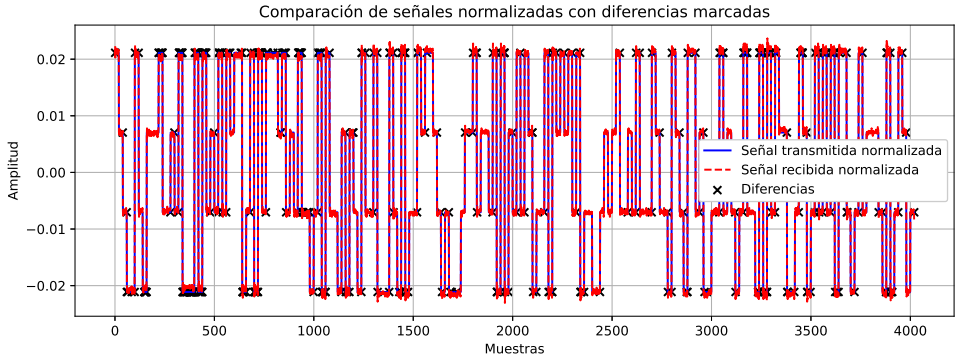

Figura 5.32 Comparación de las señales normalizadas con markers.

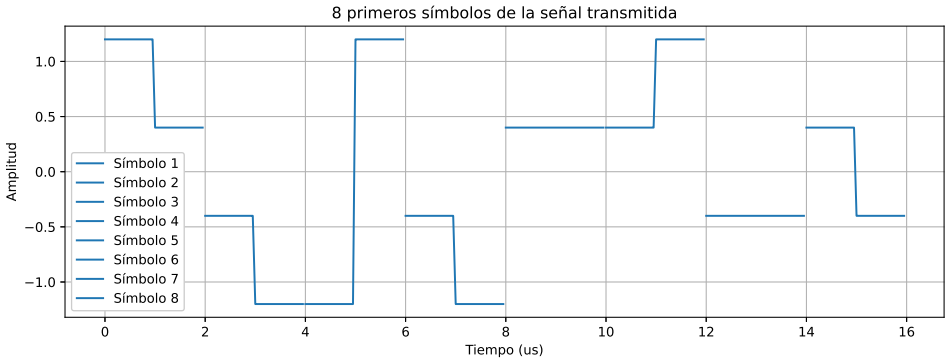

Figura 5.33 8 Primeros símbolos transmitidos.

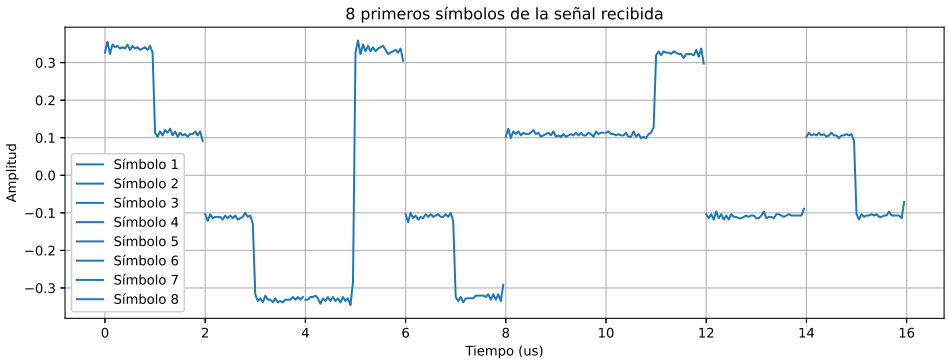

Figura 5.34 8 Primeros símbolos recibidos.

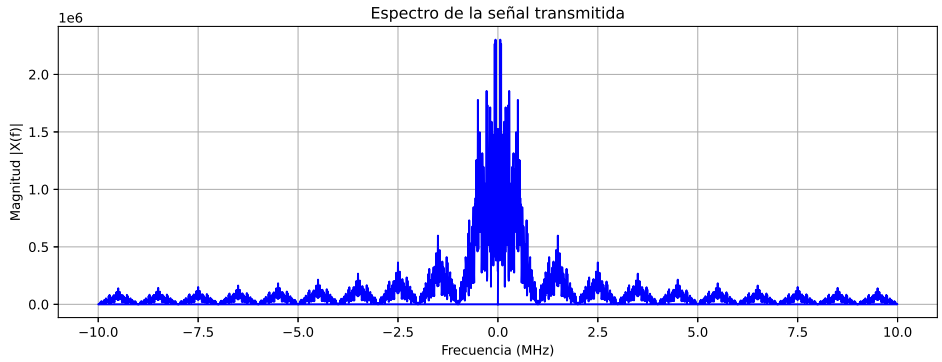

Figura 5.35 Espectro de la señal transmitida (FFT).

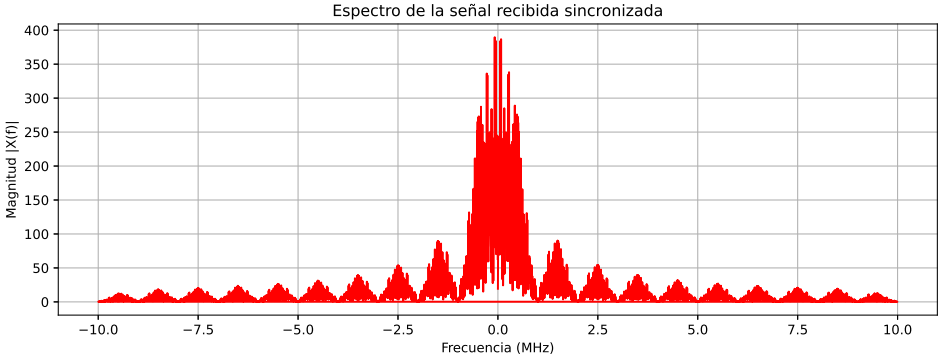

Figura 5.36 Espectro de la señal recibida (FFT).

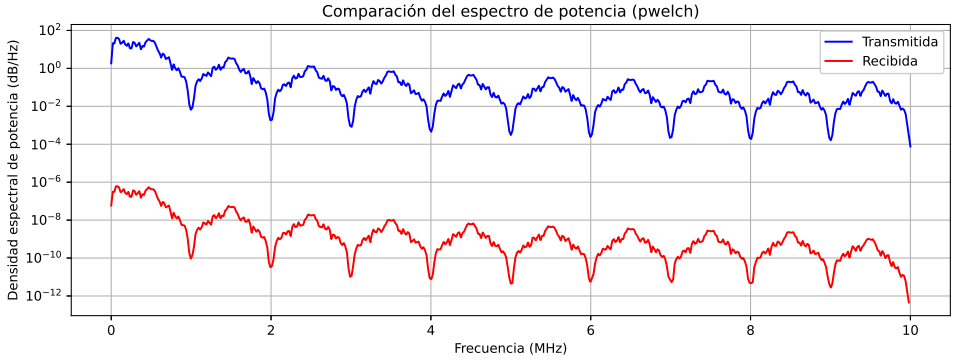

Figura 5.37 Espectro de la señal transmitida y recibida (WELCH).

complejidad adicional respecto al esquema 8-PSK, la señal ha sido demodulada con precisión, mostrando un bajo nivel de error y una constelación bien definida.

En la *Tabla 5.6* se recogen las métricas más relevantes del experimento, donde destaca un NMSE bajo y una EVM del orden del 3 %.

5.5.2 Transmisión larga

Para evaluar el rendimiento de la transmisión, se ha calculado el NMSE (Error Cuadrático Medio Normalizado) entre la señal transmitida y la recibida. El resultado obtenido es de -29.4 dB, lo cual indica una bastante buena similitud entre ambas señales pese a las inevitables distorsiones del canal.

Aparentemente la señal recibida y la transmitida presentan diferencias bastante ligeras. En la *Figura 5.39* se han graficado ambas señales superpuestas para dar pie a su comparación.

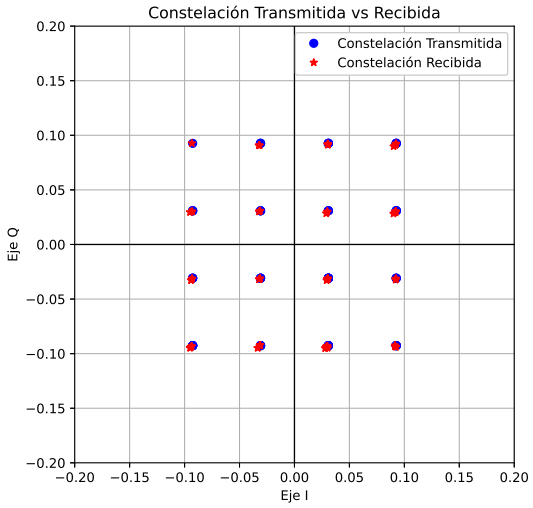

Figura 5.38 Constelación de símbolos transmitidos y recibidos (Transmisión corta).

Tabla 5.6 Valores representativos de la calidad de la transmisión (Transmisión corta).

Métrica	Valor
NMSE	−27.9 dB
EVM	1.72 %

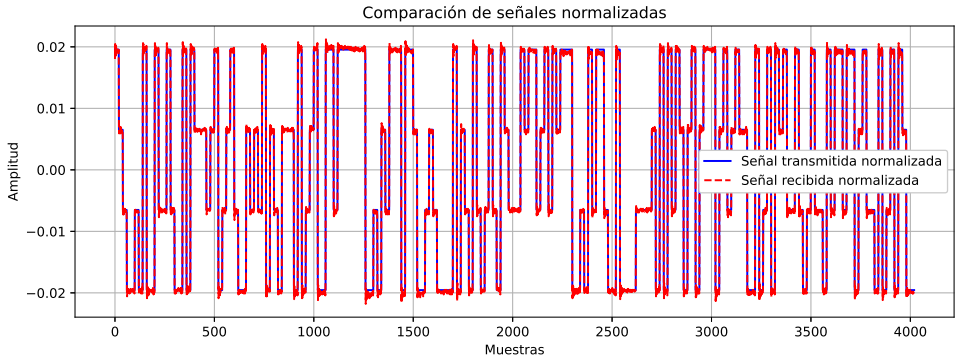

Figura 5.39 Comparación de las señales normalizadas.

En la *Figura 5.40* se han graficado los 8 primeros símbolos transmitidos y recibidos para mostrar la alineación temporal lograda y visualizar directamente el comportamiento de la modulación en las primeras posiciones.

Además, se han representado las constelaciones de los símbolos transmitidos y recibidos, observándose cómo la estructura original es prácticamente idéntica a la recibida (véase la *Figura 5.41*).

Inconvenientes en la transmisión

Durante el proceso de transmisión por tandas, se ha observado la presencia de residuos de transmisión al finalizar cada envío. Estos residuos, correspondientes a fragmentos residuales de la señal previamente trans-

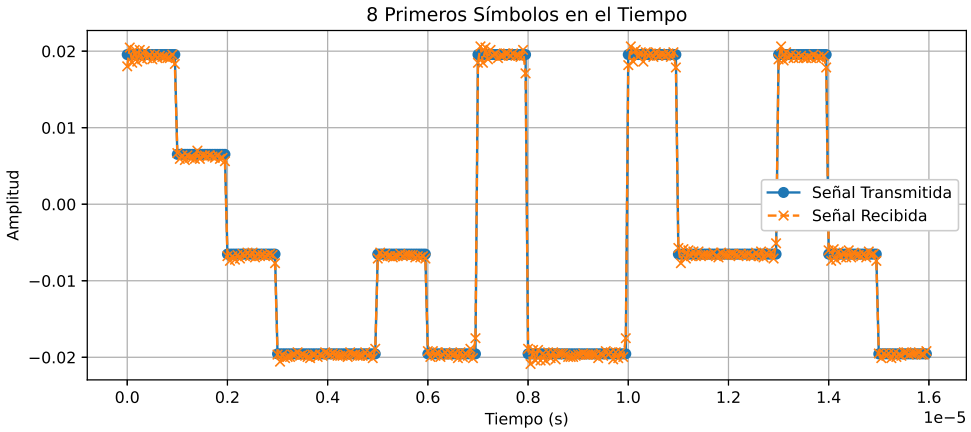

Figura 5.40 Primeros 8 símbolos transmitidos y recibidos.

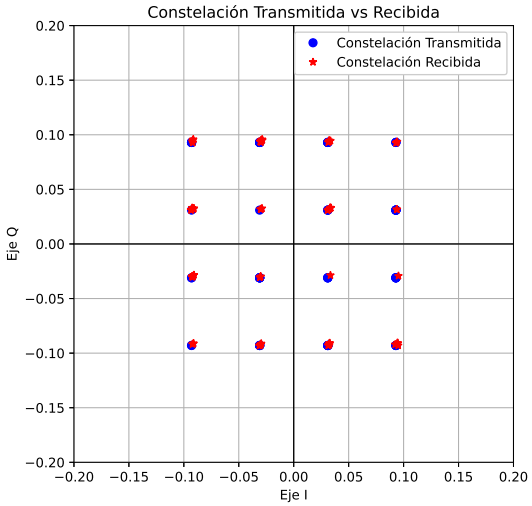

Figura 5.41 Constelación de símbolos transmitidos y recibidos (Transmisión larga).

mitida, interferían con el inicio de la siguiente tanda, dificultando la correcta sincronización y demodulación de la señal (véase la *Figura 5.42* y *Figura 5.44*). Para una mayor claridad, en la *Figura 5.43* se muestra otro ejemplo de la existencia de el residuo de transmisiones anteriores ya mencionado.

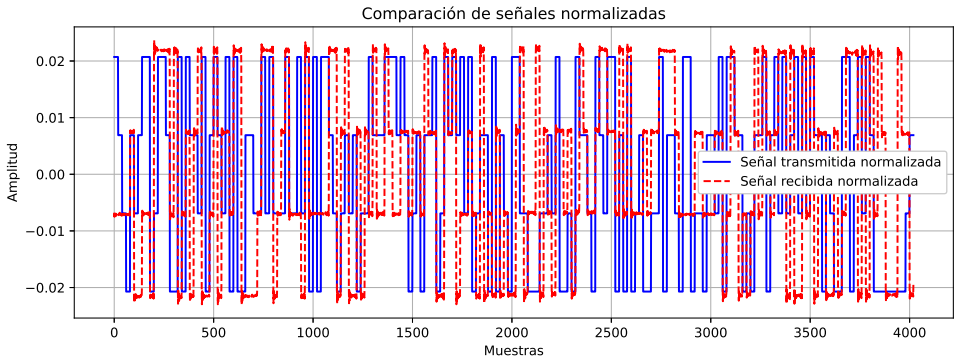

Figura 5.42 Señales mal sincronizadas por residuo de transmisión.

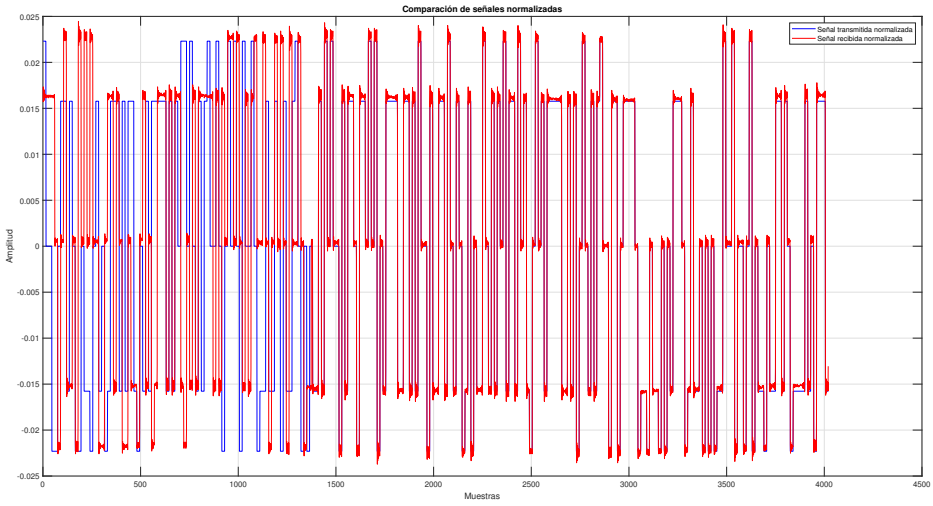

Figura 5.43 Señales mal sincronizadas por residuo de transmisión.

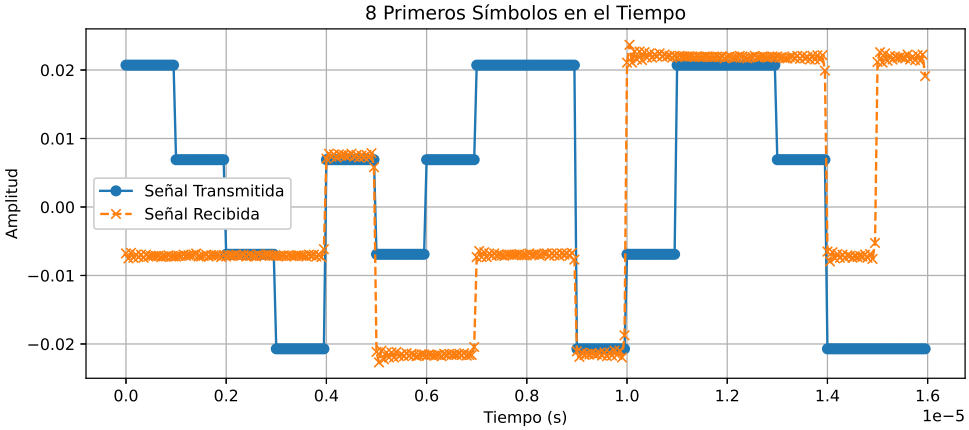

Figura 5.44 8 primeros símbolos mal sincronizados por residuo de transmisión.

Para mitigar este efecto, se han implementado dos soluciones clave:

- **Tiempo de espera entre tandas:** Se ha introducido un retardo entre el final de una transmisión y el inicio de la siguiente. Este intervalo permite que el sistema de transmisión se estabilice y que los residuos de la tanda anterior se disipen antes de comenzar con la siguiente.
- **Desecho de muestras en recepción:** En la recepción, se ha descartado una porción inicial de la señal recibida tras cada nueva tanda, eliminando así cualquier componente residual proveniente de la tanda anterior. Esto ha sido crucial para evitar errores en la detección del preámbulo y garantizar una correcta alineación temporal.

Ambas medidas han demostrado ser efectivas, permitiendo una recepción limpia y continua de las 26 tandas que conforman la transmisión total. Esta estrategia ha resultado esencial para mantener la integridad de la señal y la fiabilidad del sistema.

El sistema de transmisión QAM en paso banda para un envío grande ha funcionado satisfactoriamente. Se ha demostrado la escalabilidad del esquema al transmitir más de 10.000 bits con una tasa de errores baja y buena sincronización temporal y en fase.

Los valores recogidos en la *Tabla 5.7* indican una transmisión de alta fidelidad. El valor de NMSE negativo refleja una baja diferencia de energía entre la señal transmitida y la recibida. La EVM, por debajo del 5 %, sugiere una modulación eficiente con bajo error, mientras que una tasa de error de bits (BER) cercana a cero confirma una comunicación confiable y sin pérdidas significativas.

Tabla 5.7 Valores representativos de la calidad de la transmisión (Transmisión larga).

Métrica	Valor
NMSE	−29.4 dB
EVM	1.38 %

La comparación entre señales, junto con las constelaciones, refuerza la validez del enfoque. Las técnicas aplicadas (normalización, correlación cruzada, corrección de fase y establecimiento de tiempo de espera entre tandas) han resultado efectivas para mitigar las distorsiones típicas del canal.

5.6 Transmisión y recepción entre dos plutos

5.6.1 Transmisión y recepción de un tono

La *Figura 5.45* muestra el espectro resultante de la señal recibida, centrado correctamente en 869 MHz, lo que confirma el funcionamiento correcto del sistema de transmisión y recepción. Por el contrario, las figuras correspondientes al dominio temporal permiten comparar visualmente la señal transmitida y la recibida, observándose una buena correspondencia general, aunque con ligeras distorsiones atribuibles al canal y al hardware.

Aunque ambos PlutoSDR se configuren para operar a la misma frecuencia central, en este caso a 868 MHz, cada uno genera esta frecuencia mediante su propio oscilador interno, lo que provoca ligeras discrepancias tanto en frecuencia como en fase. Estos efectos son claramente visibles al observar la señal recibida en el dominio temporal, cuya envolvente presenta una modulación extraña (véase la *Figura 5.46* y la *Figura 5.47*).

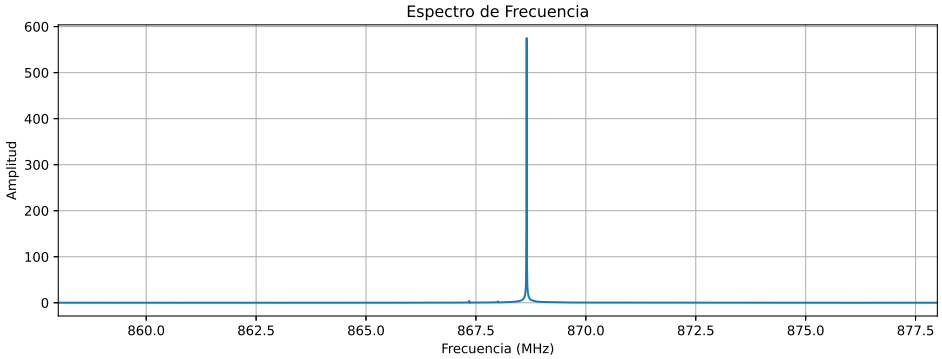

Figura 5.45 Espectro de frecuencia de la señal recibida centrada en 500 MHz.

5.6.2 Transmisión y recepción de señal PAM paso banda

En la *Figura 5.48* se ilustra una vez más lo ocurrido con el seno anteriormente, pero esta vez de un modo más acusado. La envolvente también es modulada de modo que la amplitud aumenta y disminuye en el tiempo de manera significativa. Para su mayor entendimiento, se ha representado en la *Figura 5.49* cada una de la envolvente de las señales transmitidas y recibidas, notándose que la recibida varía notablemente con respecto a la transmitida, todo debido a los errores de sincronización de los osciladores locales.

Este experimento pone en evidencia que, aunque la comunicación con dos PlutoSDR separados es factible, resulta fundamental aplicar técnicas de compensación de fase y sincronización para poder recuperar correctamente la información transmitida.

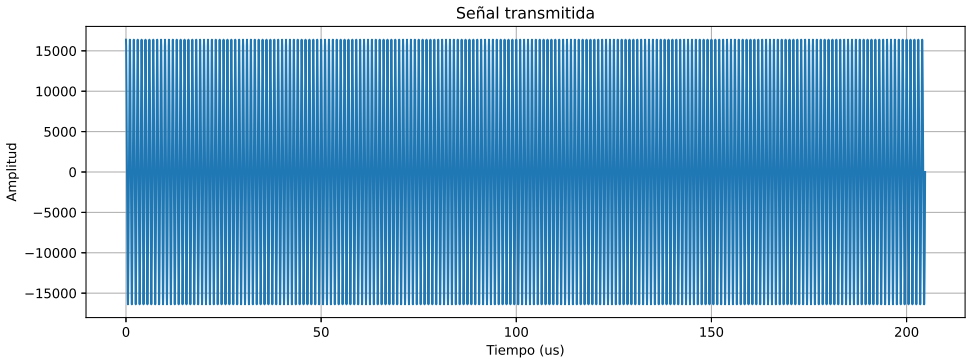

Figura 5.46 Señal senoidal transmitida.

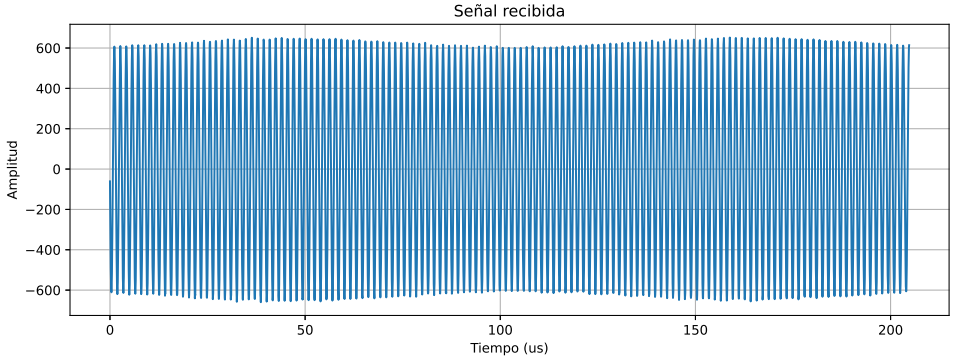

Figura 5.47 Señal senoidal recibida.

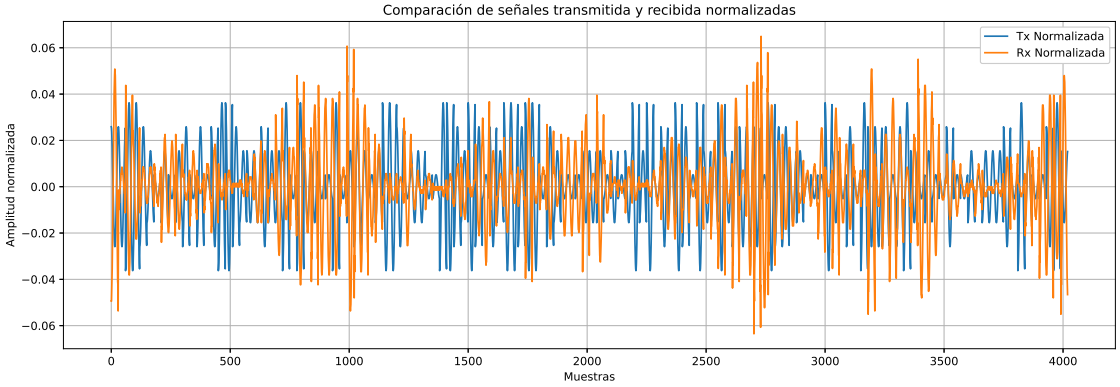

Figura 5.48 Señal 8-PAM transmitida.

5.7 Detección de actividad WiFi

Este último experimento tiene por objetivo la detección del PlutoSDR de la conmutación de la actividad WiFi. Esta señal será generada por una microcontrolador ESP8266-12F, de manera que conmute esta señal, es decir, apague y encienda el WiFi cada 5 segundos.

Para una mayor visualización de los resultados, se ha optado por el uso del dominio de la frecuencia, para poder así visualizar con mayor facilidad el pico a la frecuencia de interés. Como resultados a este experimento, se observa en la *Figura 5.50* que durante el período en el que el WiFi se encuentra apagado el dispositivo en el dominio de la frecuencia no ve nada significativo. Sin embargo, en la *Figura 5.51*, una vez que el WiFi ha sido activado aparece un pico notorio a la frecuencia del canal 1, a 2.412 GHz.

El experimento permitió observar el espectro del canal WiFi 1 de la forma esperada, capturando una banda de 20 MHz mediante una frecuencia de muestreo adecuada. Se identificó, por tanto, un pico notorio en la

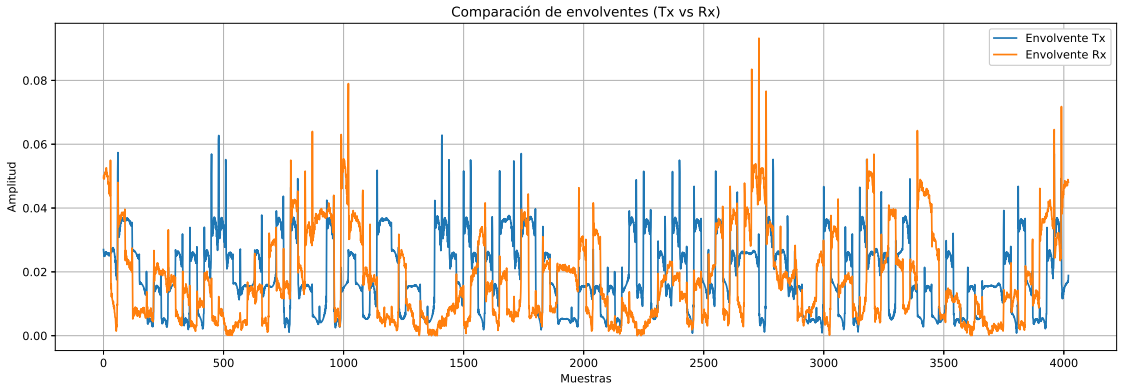

Figura 5.49 Señal 8-PAM recibida.

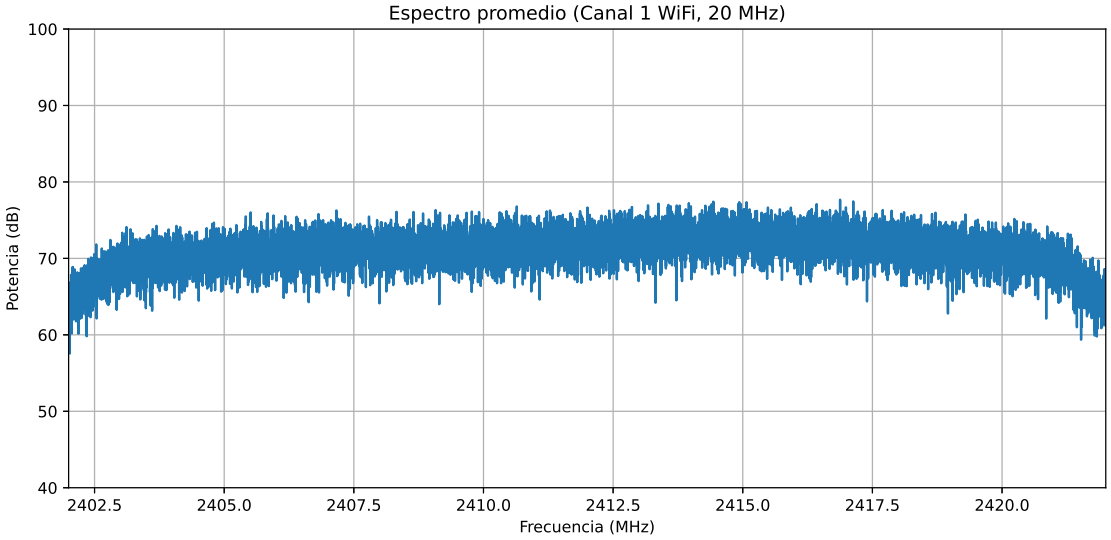

Figura 5.50 Espectro de la señal WiFi apagada (canal 1).

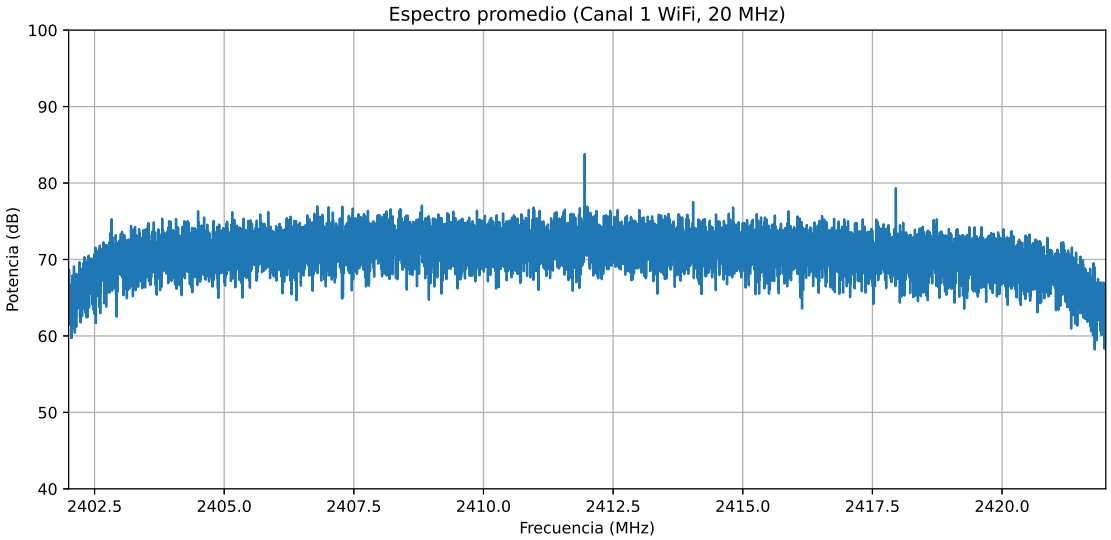

Figura 5.51 Espectro de la señal WiFi encendida (canal 1).

frecuencia central (2.412 GHz).

6 Conclusiones y líneas futuras

En este capítulo se presentan las conclusiones obtenidas a partir de todos los experimentos realizados en este proyecto. Tras haber analizado los objetivos planteados y la metodología seguida, se exponen los principales resultados alcanzados, así como las implicaciones de los mismos en el ámbito de estudio. Es por ello que se reflexiona sobre si se han cumplido las metas inicialmente planteadas y se plantean posibles líneas de trabajo futuro que podrían ampliar o mejorar los resultados obtenidos.

6.1 Cumplimiento de objetivos

A lo largo de este trabajo se ha llevado a cabo un estudio detallado sobre las distintas técnicas de modulación utilizadas en la transmisión de datos, abarcando desde sus fundamentos teóricos hasta su aplicación práctica, bajo el uso de el ADALM PlutoSDR.

Uno de los principales logros de este proyecto ha sido evidenciar cómo los diferentes esquemas de modulación (en este caso PAM, QPSK y QAM) presentan una fuerte unión entre velocidad de transmisión y resistencia al ruido. Esto permite a los dispositivos adaptar dinámicamente la modulación en función de las condiciones del canal, optimizando así el rendimiento de la red.

También se han investigado acerca de la normativa general europea de frecuencias, así como las distintas bandas de transmisión, profundizando en los servicios a los que se les atribuye dichas frecuencias. A su vez, se ha hablado, con el objetivo de tener una visión más próxima y cercana, del Cuadro Nacional de Atribución de Frecuencias (CNAF), publicado en el Boletín Oficial del Estado (BOE).

Además, también se ha obtenido la tasa de muestreo efectiva en el PlutoSDR, de manera que, junto con el experimento de la detección de activación de la señal WiFi, podrán ser utilizadas para experimentos futuros.

En definitiva, el trabajo ha cumplido con el objetivo de proporcionar una visión integral y comparativa de las técnicas de modulación en WiFi. Se concluye que la modulación no es solo un componente técnico, sino un factor estratégico en la evolución y el diseño de redes inalámbricas eficientes y fiables.

6.2 Líneas futuras para la continuación

Una de las principales líneas futuras para la continuación de este trabajo se centran en la aplicación de los conocimientos adquiridos sobre la plataforma SDR desarrollada a tareas de detección, reconocimiento e identificación (DRI) de drones. Para ello, resulta fundamental conocer las principales modulaciones y protocolos de comunicación empleados en estos sistemas aéreos no tripulados. Estos protocolos permiten el envío de comandos del piloto al dron y la recepción de datos como estado de batería, posición GPS, telemetría, etc.

6.2.1 Modulaciones y protocolos comunes en drones

Algunos de los protocolos y modulaciones más comunes en drones que se abordarán incluyen:

- **FSK (Frequency Shift Keying):** Utilizado en algunos sistemas de radiocontrol de baja frecuencia (en 433 MHz o 915 MHz) debido a su alta eficiencia y alcance. Se puede observar cómo la frecuencia cambia entre dos valores distintos (por ejemplo, 1000 Hz y 2000 Hz) dependiendo de los bits transmitidos (1 o 0).

Tabla 6.1 Recopilación de los resultados de los experimentos abordados.

Experimento	Métrica	Valor
PAM (Transmisión corta)	NMSE EVM	-28.7 dB 0.50 %
PAM (Transmisión larga)	NMSE EVM	-19.0 dB 3.67 %
PSK (Transmisión corta)	NMSE EVM	-27.3 dB 1.56 %
PSK (Transmisión larga)	NMSE EVM	-19.2 dB 7.29 %
QAM (Transmisión corta)	NMSE EVM	-27.9 dB 1.72 %
QAM (Transmisión larga)	NMSE EVM	-29.4 dB 1.38 %
Tasa de muestreo efectiva	Muestras recibidas	3×10^6 muestras/s
Conmutación WiFi	-	✓

- **DSSS (Direct Sequence Spread Spectrum):** Empleado en protocolos como FrSky ACCST, mejora la resistencia a interferencias. Con DSSS, la señal original se expande utilizando un código de dispersión (como un código Barker), lo que aumenta su ancho de banda. Posteriormente, la señal es modulada en una portadora para su transmisión.
- **FHSS (Frequency Hopping Spread Spectrum):** Usado en sistemas como Spektrum DSMX y FrSky ACCESS, cambia rápidamente de frecuencia en una secuencia predefinida para evitar interferencias y aumentar la robustez de la comunicación. La implementación de la detección y seguimiento de FHSS con SDR implica la monitorización rápida del espectro o el uso de técnicas de procesamiento específicas para identificar los saltos de frecuencia.

La comprensión y el análisis de estas modulaciones y protocolos son esenciales para el desarrollo de algoritmos de procesamiento de señal digital capaces de detectar, reconocer y, finalmente, identificar las emisiones de RF de los drones.

Otras áreas clave para la continuación incluyen:

- **Ampliación de la base de datos de señales:** Recopilar y analizar una gran variedad de señales de diferentes tipos de drones y protocolos de comunicación para mejorar la capacidad de detección y reconocimiento.
- **Desarrollo de algoritmos avanzados de procesamiento:** Implementar técnicas más sofisticadas de procesamiento digital de señales y aprendizaje automático para la identificación automática de modelos de drones específicos a partir de sus emisiones de RF.
- **Pruebas en entornos reales:** Validar el sistema SDR en diferentes entornos operativos, considerando factores como la distancia, los obstáculos y la interferencia, para evaluar su rendimiento en situaciones prácticas.
- **Integración con otros sensores:** Explorar la combinación de la información de RF con datos de otros sensores (por ejemplo, cámaras, micrófonos) para mejorar la precisión y robustez del sistema DRI.
- **Implementación en tiempo real:** Optimizar los algoritmos y el software para permitir la detección, reconocimiento e identificación de drones en tiempo real utilizando la plataforma ADALM Pluto SDR u otro hardware SDR más potente.
- **Análisis de contramedidas electrónicas:** Investigar posibles técnicas de interferencia o engaño de señales de drones y desarrollar métodos para detectarlas y mitigarlas utilizando el sistema SDR.

La consecución de estas líneas futuras permitirá avanzar significativamente en el desarrollo de sistemas efectivos y versátiles para la vigilancia y seguridad del espacio aéreo frente a la creciente proliferación de drones.

Apéndice A
Códigos

En este apéndice se incluye el código fuente desarrollado como parte del Trabajo de Fin de Grado. El objetivo es proporcionar una referencia completa y detallada del software implementado, facilitando su comprensión, análisis y posible reutilización.

El código está debidamente comentado y estructurado. Se presenta dividido en secciones, de manera que cada una se corresponde con los códigos empleados en cada experimento, con el fin de mejorar su claridad y mantenimiento.

A.1 Generación y transmisión de un tono

Código A.1 Generación y transmisión de un tono.

```
# %% [markdown]
# # Proyecto: Transmisión de un tono con PlutoSDR
# **Autor:** Alejandro Madero Vílchez
# **Fecha:** 4 de abril de 2025
# **Grado:** Ingeniería Aeroespacial

# %% [markdown]
# ## Librerías y Parámetros

# %%
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import windows
from adi import Pluto

# Configuración de parámetros
fs = int(20e6)                                                    # Frecuencia de muestreo (20 MHz)
tx_freq = int(500e6)                                              # Frecuencia de transmisión (500 MHz)
rx_freq = int(500e6)                                              # Frecuencia de recepción (500 MHz)
tone_freq = 1e6                                                   # Tono a 1 MHz
num_samples = 4096                                                # Número de muestras

# Conectar con PlutoSDR
sdr = Pluto(uri="ip:192.168.2.1")


# Configuración de transmisión
sdr.tx_lo = tx_freq                                               # TX en 500 MHz
sdr.tx_hardwaregain = -50                                         # Potencia de transmisión (-50 dB)
```

```
sdr.tx_cyclic_buffer = True                                                    # Habilitar transmisión cíclica
sdr.sample_rate = fs                                                           # Configuramos fs del pluto

# Configuración de recepción
sdr.rx_lo = rx_freq                                                            # RX en 500 MHz
sdr.rx_rf_bandwidth = int(20e6)                                                # Ancho de banda de 20 MHz
sdr.rx_buffer_size = num_samples                                               # Tamaño del buffer
sdr.rx_hardwaregain_chan0 = 0.0

# %% [markdown]
# ## Transmisión del tono y recepción

# %%
# Generar el tono I/Q
t = np.arange(num_samples) / fs
tx_signal = 2**14 * np.exp(2j * np.pi * tone_freq * t)                         # Señal I/Q

# Transmitir la señal
sdr.tx(tx_signal)


# Clear buffer just to be safe
for i in range (0, 10):
    raw_data = sdr.rx()# Receive samples
    rx_samples = sdr.rx()
# Capturar datos
rx_signal = sdr.rx()


# Aplicar ventana Hann para mejorar la FFT
samples_windowed = rx_signal
# Calcular FFT y corregir frecuencia
X_f = np.fft.fft(samples_windowed)
X_f = np.fft.fftshift(np.abs(X_f) / len(rx_signal ))                           # Normalización correcta

# Ajustar escala de frecuencia correctamente
freqs_mhz = np.fft.fftshift(np.fft.fftfreq(len(rx_signal ), d=1/fs)) * (1e-6) + (rx_freq * 1e-6)
    # Ahora centrado correctamente

# %% [markdown]
# ## Graficar señales en el tiempo

# %%
# Graficar señal transmitida en el dominio del tiempo
plt.figure(figsize=(12, 4))
plt.plot(t*1e6, np.real(tx_signal))                                            # Eje X ahora en Hz
plt.title("Señal transmitida")
plt.xlabel("Tiempo (us)")
plt.ylabel("Amplitud")
plt.grid()
plt.ticklabel_format(useOffset=False, style='plain')
#plt.savefig("seno_1pluto_señal_tx.eps", format="eps", dpi=300)


# Graficar señal recibida en el dominio del tiempo
plt.figure(figsize=(12, 4))
plt.plot(t*1e6, np.real(rx_signal))                                            # Eje X ahora en Hz
plt.title("Señal recibida")
plt.xlabel("Tiempo (us)")
plt.ylabel("Amplitud")
plt.grid()
plt.ticklabel_format(useOffset=False, style='plain')
```

```
#plt.savefig("seno_1pluto_señal_rx.eps", format="eps", dpi=300)




plt.show()

# %% [markdown]
# ## Graficar espectro de la señal recibida

# %%
# Graficar espectro de frecuencia
plt.figure(figsize=(12, 4))
plt.plot(freqs_mhz, X_f) # Eje X ahora en Hz
plt.title("Espectro de Frecuencia de la señal recibida")
plt.xlabel("Frecuencia (MHz)")
plt.ylabel("Amplitud")
plt.grid()
plt.ticklabel_format(useOffset=False, style='plain')                                    # Forzar escala correcta
plt.xlim(freqs_mhz.min(), freqs_mhz.max())
#plt.savefig("seno_1pluto_espectro_señal_rx.eps", format="eps", dpi=300)
plt.show()

# %%
sdr.tx_destroy_buffer()
```

A.2 Evaluación de la tasa de muestreo efectiva en PlutoSDR

Código A.2 Evaluación de la tasa de muestreo efectiva en PlutoSDR.

```
# # Proyecto: Transmisión de un tono con PlutoSDR
# **Autor:** Alejandro Madero Vílchez
# **Fecha:** 4 de abril de 2025
# **Grado:** Ingeniería Aeroespacial

# Librerias necesarias
import numpy as np
import adi
import time

# Configuración inicial
sample_rate = 3e6                                       # Frecuencia de muestreo en Hz
center_freq = 915e6                                     # Frecuencia central en Hz
rx_buffer_size = 16384                                  # Tamaño del buffer de recepción
num_iterations = 100                                    # Número de iteraciones para medir el rendimiento

# Inicializar Pluto-SDR
sdr = adi.Pluto("ip:192.168.2.1")
sdr.sample_rate = int(sample_rate)
sdr.rx_rf_bandwidth = int(sample_rate)                  # Ancho de banda del filtro
sdr.rx_lo = int(center_freq)
sdr.rx_buffer_size = rx_buffer_size                     # Configurar tamaño del buffer

# Medir tasa de muestreo efectiva
start_time = time.time()
total_samples = 0

for _ in range(num_iterations):
    samples = sdr.rx()                                  # Recibir muestras
    total_samples += len(samples)                       # Contar el número de muestras recibidas

end_time = time.time()

time_elapsed = end_time - start_time
actual_sample_rate = total_samples / time_elapsed
```

```
36
37
38 # Mostrar valores por pantalla
39 print(f"Tiempo transcurrido: {time_elapsed:.4f} segundos")
40 print(f"Muestras recibidas: {total_samples}")
41 print(f"Tasa de muestreo efectiva: {actual_sample_rate:.2f} muestras/segundo")
```

A.3 Transmisión y recepción de una señal PAM paso banda

Código A.3 Codificación Gray.

```
1  import numpy as np
2
3  def gray2de(b):
4      '''
5      Convierte cada fila de la matriz formada por dígitos binarios b en un vector
6      columna de los valores decimales correspondientes.
7
8      b: numpy.array
9          Matriz donde cada fila representa un número en código Gray.
10     '''
11     # Crear una matriz de ceros del mismo tamaño que b
12     c = np.zeros_like(b)
13     c[:, 0] = b[:, 0] # El primer bit de Gray es igual al primer bit binario
14
15     # Realizar la operación XOR para convertir de Gray a binario
16     for i in range(1, b.shape[1]):
17         c[:, i] = np.logical_xor(c[:, i-1], b[:, i])
18
19     # Convertir la matriz resultante de bits binarios a decimal
20     c = np.fliplr(c) # Invertir las columnas para que el bit menos significativo quede a la
           derecha
21
22     # Si la matriz tiene dimensiones incorrectas, devolver una lista vacía
23     if min(c.shape) < 1:
24         d = []
25         return d
26
27     # Calcular el valor decimal para cada fila
28     d = np.dot(c, 2**np.arange(c.shape[1]))
29
30     return d
```

Código A.4 Transmisor PAM.

```
1  import numpy as np
2  from gray2de import gray2de
3
4  def transmisorpam(Bn, Eb, M, p, L):
5      '''
6      [Xn, Bn, An, phi, alfabetopam] = transmisorpam(Bn, Eb, M, p, L)
7
8      Parámetros:
9      Bn = Secuencia de dígitos binarios
10     Eb = Energía media por bit transmitida en Julios
11     M = Número de símbolos del código PAM
12     p = Pulso conformador
13     L = Número de puntos utilizados en la representación de un símbolo
14
15     Devuelve:
16     Xn = Señal de información (discreta)
17     Bn = Secuencia de dígitos binarios realmente transmitidos
18     An = Secuencia de niveles de amplitud transmitidos
19     phi = Pulso básico real normalizado de energía unitaria
20     alfabetopam = Niveles de amplitud asociados a cada símbolo transmitido
```

```
    '''

    # Determinar cuántos bits hay en cada símbolo
    k = int(np.ceil(np.log2(M)))

    # Ajustar M a una potencia de dos
    M = 2**k

    # Calcular el alfabeto PAM
    alfabetopam = np.sqrt(3 * Eb * np.log2(M) / (M**2 - 1)) * (2 * np.arange(M) - M + 1)

    # Ajustar la longitud de Bn para que sea múltiplo de k
    Nb = len(Bn)
    Bn = np.pad(Bn, [0, int(k * np.ceil(Nb / k) - Nb)], mode='constant')
    Nb = len(Bn)
    Ns = Nb // k # Número de símbolos a transmitir

    # Convertir la secuencia binaria en niveles de amplitud
    if M > 2:
        An = alfabetopam[gray2de(np.reshape(Bn, [Ns, k]))]
    else:
        An = alfabetopam[Bn]

    # Ajustar el pulso básico para que tenga exactamente L muestras
    Ls = len(p)
    if Ls < L:
        p = np.pad(p, [0, L - Ls], mode='constant')
    else:
        print('La duración del pulso se ha truncado a {} muestras'.format(str(L)))
        p = p[:L]

    # Normalizar la energía del pulso suministrado
    phi = (1 / np.sqrt(np.dot(p, p.T))) * p

    # Obtener el tren de pulsos
    Xn = np.kron(An, phi)
    Xn = np.array(Xn)

    return [Xn, Bn, An, phi, alfabetopam]
```

Código A.5 Transmisión y recepción de una señal PAM paso banda (Transmisión corta).

```
# %% [markdown]
# # Proyecto: Transmision pequena 8PAM con PlutoSDR
# **Autor:** Alejandro Madero Vilchez
# **Fecha:** 4 de abril de 2025
# **Grado:** Ingenieria Aeroespacial

# %% [markdown]
# ## Librerias y parametros

# %%
# Librerias necesarias
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import welch
from adi import Pluto
from transmisorpam import transmisorpam

# Configuracion de parametros
sample_rate = 20e6
num_samples = 4020
center_freq = 915e6

# Conectar con PlutoSDR
sdr = Pluto(uri="ip:192.168.2.2")
sdr.sample_rate = int(sample_rate)

# Configuracion de transmision
```

```
sdr.tx_rf_bandwidth = int(sample_rate)
sdr.tx_lo = int(center_freq)
sdr.tx_hardwaregain_chan0 = -10

# Configuracion de recepcion
sdr.rx_lo = int(center_freq)
sdr.rx_rf_bandwidth = int(sample_rate)
sdr.rx_buffer_size = num_samples
sdr.gain_control_mode_chan0 = 'manual'
sdr.rx_hardwaregain_chan0 = 0.0

# Datos de la transmision
preambulo = np.array([1, 0, 1, 1, 0, 1, 0, 0]) # Secuencia fija de sincronizacion
Bn = np.concatenate([preambulo, np.random.randint(0, 2, 394)]) # Mensaje con preambulo
Eb = 8 # Energia por bit
L = 30 # Numero de muestras por simbolo
Rb = 2 * 10**6 # Tasa de bits (2 Mbps)
M = 8  # Orden del esquema PAM
k = np.log2(M) # Numero de bits por simbolo
Ns = int(len(Bn) / k) # Numero de simbolos transmitidos

# Forzar numero de muestras por simbolo a ser par
L = int(np.ceil(L / 2) * 2)

# Parametros de tiempo continuo
Tb = 1 / Rb # Duracion de bit en segundos
Tm = ((Tb * np.log2(M)) / L) # Intervalo de muestreo
fs = 1 / Tm # Frecuencia de muestreo
Nb = len(Bn) # Numero de bits transmitidos
Td = Tb * Nb # Duracion total de la senal
t = np.arange(0, Td, Tm) # Vector de tiempo
f0 = fs / L # Frecuencia central

# Pulso conformador paso de banda
g = np.cos(2 * np.pi * f0 * np.arange(L) * Tm)

## IMPRIMIR TODO
print("\n--- PARAMETROS DE CONFIGURACION ---")
print(f"Sample rate: {sample_rate}")
print(f"Numero de muestras: {num_samples}")
print(f"Frecuencia central: {center_freq/1e6} MHz")
print(f"Tasa de bits (Rb): {Rb} bps")
print(f"Duracion del bit (Tb): {Tb} s")
print(f"Intervalo de muestreo (Tm): {Tm} s")
print(f"Frecuencia de muestreo (fs): {fs} Hz")
print(f"Frecuencia del pulso paso banda (f0): {f0} Hz")
print(f"Orden de PAM (M): {M}")
print(f"Numero de bits por simbolo (k): {k}")
print(f"Numero de muestras por simbolo (L): {L}")

# %% [markdown]
# ## Transmision y recepcion

# %%
# Generar senal discreta PAM
[Xn, Bn, An, phi, alfabeto] = transmisorpam(Bn, Eb, M, g, L)

# Normalizar senal para transmision con PlutoSDR
Xt = np.sqrt(1 / Tm) * Xn

# Transmitir la senal
sdr.tx_cyclic_buffer = True
sdr.tx(Xt)

# Capturar datos
rx_signal = sdr.rx()
#rx_signal = rx_signal - np.mean(rx_signal) # Restar la media
rx_signal = rx_signal / np.max(np.abs(rx_signal)) # Normalizar senal recibida

# ---- Calculo del desfase optimo ----
corr = np.correlate(np.real(rx_signal), np.real(Xn), mode='full')
```

```
lag = np.arange(-len(Xn) + 1, len(rx_signal)) * Tm * 1e6 # Convertir a microsegundos

# Encontrar el desfase optimo
lag_max = lag[np.argmax(corr)]
sample_offset = np.argmax(corr) - (len(Xn) - 1) # Indice de desfase en muestras

print(f"Desfase optimo: {lag_max:.2f} us")

# ---- Sincronizacion de la senal recibida ----
rx_signal_sync = np.roll(rx_signal, -sample_offset) # Desplazar la senal corregida

# ---- Separar en simbolos ----
Xn_reshaped = Xn[:len(Xn) - (len(Xn) % L)].reshape(-1, L)
rx_signal_reshaped = rx_signal_sync[:len(rx_signal_sync) - (len(rx_signal_sync) % L)].reshape(-1,
      L)

num_symbols_tx = Xn_reshaped.shape[0]
num_symbols_rx = rx_signal_reshaped.shape[0]

# ---- Codigo de colores ----
num_colors = max(num_symbols_tx, num_symbols_rx)
colors_tx = plt.cm.get_cmap("tab10", num_colors)
colors_rx = plt.cm.get_cmap("tab10", num_colors)

# %% [markdown]
# ## Representacion de las senales en el tiempo

# %%
# ---- REPRESENTACION DE LA SENAL TRANSMITIDA EN EL TIEMPO ----
plt.figure(figsize=(12, 4))
for i in range(num_symbols_tx):
    tiempo = t[i * L: (i + 1) * L] * 1e6
    plt.plot(tiempo, np.real(Xn_reshaped[i, :]), color=colors_tx(i), linewidth=1.5)

plt.title("Senal transmitida con colores por simbolo")
plt.xlabel("Tiempo (us)")
plt.ylabel("Amplitud")
plt.savefig("PAM_pequeno_senaltx_color.eps", format="eps", dpi=300)
plt.grid()

# ---- REPRESENTACION DE LA SENAL RECIBIDA EN EL TIEMPO ----
plt.figure(figsize=(12, 4))
for i in range(num_symbols_rx):
    tiempo = t[i * L: (i + 1) * L] * 1e6
    plt.plot(tiempo, np.real(rx_signal_reshaped[i, :]), color=colors_rx(i), linewidth=1.5)

plt.title("Senal recibida con colores por simbolo")
plt.xlabel("Tiempo (us)")
plt.ylabel("Amplitud")
plt.grid()
plt.savefig("PAM_pequeno_senalrx_color.eps", format="eps", dpi=300)
plt.show()

# %% [markdown]
# ## Representacion de los 8 primeros simbolos en el tiempo

# %%
# ---- 8 PRIMERO SIMBOLOS DE LA SENAL TRANSMITIDA ----
num_symbols_to_plot = 8
plt.figure(figsize=(12, 4))
for i in range(num_symbols_to_plot):
    tiempo = t[i * L: (i + 1) * L] * 1e6
    plt.plot(tiempo, np.real(Xn_reshaped[i, :]), color=colors_tx(i), label=f'Simbolo {i+1}')

plt.title("8 primeros simbolos de la senal transmitida")
plt.xlabel("Tiempo (us)")
plt.ylabel("Amplitud")
plt.legend()
plt.grid()
#plt.savefig("simbolos_tx.eps", format="eps", dpi=300)

```

```
169
170 # ---- 8 PRIMEROS SIMBOLOS DE LA SENAL RECIBIDA ----
171 plt.figure(figsize=(12, 4))
172 for i in range(num_symbols_to_plot):
173     tiempo = t[i * L: (i + 1) * L] * 1e6
174     plt.plot(tiempo, np.real(rx_signal_reshaped[i, :]), color=colors_rx(i), label=f'Simbolo {i+1}'
          )
175
176 plt.title("8 primeros simbolos de la senal recibida")
177 plt.xlabel("Tiempo (us)")
178 plt.ylabel("Amplitud")
179 plt.legend()
180 plt.grid()
181 #plt.savefig("simbolos_rx.eps", format="eps", dpi=300)
182 plt.show()
183
184 # %% [markdown]
185 # ## Espectros de las senales FFT
186
187 # %%
188 # ---- ESPECTRO DE LA SENAL TRANSMITIDA ----
189 Xf = np.fft.fft(Xt)
190 freqs = np.fft.fftfreq(len(Xt), d=Tm) / 1e6
191
192 plt.figure(figsize=(12, 4))
193 plt.plot(freqs, np.abs(Xf), color='b', linewidth=1.5)
194 plt.title("Espectro de la senal transmitida")
195 plt.xlabel("Frecuencia (MHz)")
196 plt.ylabel("Magnitud |X(f)|")
197 plt.grid()
198 #plt.savefig("espectro_senal_tx.eps", format="eps", dpi=300)
199
200 # ---- ESPECTRO DE LA SENAL RECIBIDA ----
201 Xf_rx = np.fft.fft(rx_signal_sync)
202 freqs_rx = np.fft.fftfreq(len(rx_signal_sync), d=Tm) / 1e6
203
204 plt.figure(figsize=(12, 4))
205 plt.plot(freqs_rx, np.abs(Xf_rx), color='r', linewidth=1.5)
206 plt.title("Espectro de la senal recibida sincronizada")
207 plt.xlabel("Frecuencia (MHz)")
208 plt.ylabel("Magnitud |X(f)|")
209 plt.grid()
210 #plt.savefig("espectro_senal_rx.eps", format="eps", dpi=300)
211 plt.show()
212
213 # %% [markdown]
214 # ## Espectros de las senales PWELCH
215
216 # %%
217 # ---- ESPECTRO DE FRECUENCIA TRANSMITIDA PWELCH ----
218 f_tx, Pxx_tx = welch(Xt, fs=1/Tm, nperseg=1024)
219
220 plt.figure(figsize=(12, 4))
221 plt.semilogy(f_tx / 1e6, Pxx_tx, color='b', linewidth=1.5) # Escala logaritmica
222 plt.title("Espectro de potencia de la senal transmitida (pwelch)")
223 plt.xlabel("Frecuencia (MHz)")
224 plt.ylabel("Densidad espectral de potencia (dB/Hz)")
225 plt.grid()
226 #plt.savefig("welch_tx.eps", format="eps", dpi=300)
227
228 # ---- ESPECTRO DE FRECUENCIA RECIBIDA PWELCH ----
229 f_rx, Pxx_rx = welch(rx_signal_sync, fs=1/Tm, nperseg=1024)
230
231 plt.figure(figsize=(12, 4))
232 plt.semilogy(f_rx / 1e6, Pxx_rx, color='r', linewidth=1.5) # Escala logaritmica
233 plt.title("Espectro de potencia de la senal recibida (pwelch)")
234 plt.xlabel("Frecuencia (MHz)")
235 plt.ylabel("Densidad espectral de potencia (dB/Hz)")
236 plt.grid()
237 #plt.savefig("welch_rx.eps", format="eps", dpi=300)
238 plt.show()
```

```
239
240
241 #### AMBAS
242 # Filtrar solo las frecuencias positivas
243 mask_tx = f_tx >= 0
244 mask_rx = f_rx >= 0
245 f_tx_positive = f_tx[mask_tx]
246 Pxx_tx_positive = Pxx_tx[mask_tx]
247 f_rx_positive = f_rx[mask_rx]
248 Pxx_rx_positive = Pxx_rx[mask_rx]
249
250 # Graficar ambos espectros en una misma figura
251 plt.figure(figsize=(12, 4))
252 plt.semilogy(f_tx_positive / 1e6, Pxx_tx_positive, color='b', linewidth=1.5, label="Transmitida")
253 plt.semilogy(f_rx_positive / 1e6, Pxx_rx_positive, color='r', linewidth=1.5, label="Recibida")
254
255 plt.title("Comparacion del espectro de potencia (pwelch)")
256 plt.xlabel("Frecuencia (MHz)")
257 plt.ylabel("Densidad espectral de potencia (dB/Hz)")
258 plt.legend()
259 plt.grid()
260 #plt.savefig("welch_ambas.eps", format="eps", dpi=300)
261 plt.show()
262
263 # %% [markdown]
264 # ## Calculo del NMSE (Error cuadratico medio normalizado)
265
266 # %%
267 # ---- CALCULO DEL NMSE ----
268 min_len = min(len(Xn), len(rx_signal_sync))
269 Xn_trimmed = Xn[:min_len] # Usar Xn en lugar de Xt
270 rx_signal_sync = np.real(rx_signal_sync) # Convertir a real
271 rx_signal_trimmed = rx_signal_sync[:min_len]
272
273 # Normalizar las senales
274 Xn_normalized = Xn_trimmed / np.linalg.norm(Xn_trimmed)
275 rx_signal_normalized = rx_signal_trimmed / np.linalg.norm(rx_signal_trimmed)
276
277 # Calculo del NMSE
278 nmse = 20 * np.log10(np.linalg.norm(rx_signal_normalized - Xn_normalized) / np.linalg.norm(
      Xn_normalized))
279 print(f"NMSE entre la senal transmitida y recibida (en dB): {nmse:.6f} dB")
280
281 # Verificacion de normalizacion
282 print(f"Norma de Xn_normalized: {np.linalg.norm(Xn_normalized)}")
283 print(f"Norma de rx_signal_normalized: {np.linalg.norm(rx_signal_normalized)}")
284 print(f"Tamano de Xn_normalized: {len(Xn_normalized)}")
285 print(f"Tamano de rx_signal_normalized: {len(rx_signal_normalized)}")
286
287 # ---- REPRESENTACION DE LAS SENALES NORMALIZADAS ----
288 plt.figure(figsize=(12, 4))
289 plt.plot(np.real(Xn_normalized), label="Senal transmitida normalizada", color='b')
290 plt.plot(np.real(rx_signal_normalized), label="Senal recibida normalizada", color='r', linestyle='
      dashed')
291 plt.title("Comparacion de senales normalizadas")
292 plt.xlabel("Muestras")
293 plt.ylabel("Amplitud")
294 plt.legend()
295 plt.grid()
296 #plt.savefig("senales_normalizadas.eps", format="eps", dpi=300)
297 plt.show()
298
299 # ---- Verificacion de tipos y partes imaginarias ----
300 print(f"Tipo de Xn: {Xn.dtype}, Tipo de rx_signal: {rx_signal.dtype}")
301 print(f"Maximo de la parte imaginaria de Xn: {np.max(np.abs(np.imag(Xn)))}")
302 print(f"Maximo de la parte imaginaria de rx_signal: {np.max(np.abs(np.imag(rx_signal)))}")
303
304 # %% [markdown]
305 # ## Grafica de las senales normalizadas con markers
306
307 # %%
```

```
# Calcular diferencias significativas entre las senales
diferencias = np.abs(np.real(Xn_normalized) - np.real(rx_signal_normalized))
umbral = 1e-3 # Umbral para considerar que los puntos no coinciden
indices_diferencia = np.where(diferencias > umbral)[0] # Indices donde hay diferencias

# Graficar senales
plt.figure(figsize=(12, 4))
plt.plot(np.real(Xn_normalized), label="Senal transmitida normalizada", color='b')
plt.plot(np.real(rx_signal_normalized), label="Senal recibida normalizada", color='r', linestyle='
    dashed')

# Agregar marcadores en los puntos de diferencia
plt.scatter(indices_diferencia, np.real(Xn_normalized[indices_diferencia]),
           color='black', marker='x', label="Diferencias")

# Configuracion de la grafica
plt.title("Comparacion de senales normalizadas con diferencias marcadas")
plt.xlabel("Muestras")
plt.ylabel("Amplitud")
plt.legend()
plt.grid()
#plt.savefig("senales_normalizadas_markers.eps", format="eps", dpi=300)
plt.show()

# %% [markdown]
# ## Constelacion de bits Tx y Rx

# %%
# CONSTELACION DE BITS Tx Y Rx
hr1 = np.flipud(phi) # Filtro de recepcion
yn1 = np.convolve(hr1, Xn_normalized, 'valid') # Salida del filtro adaptado

sn1 = yn1[np.arange(0, Ns * L, L)] # Muestras en los instantes correctos

xmax = max(sn1) + max(sn1)/10 + 0.5
xmin = -(abs(min(sn1)) + abs(min(sn1))/10) - 0.5


# ---- Constelacion recibida ----
# Filtrado adaptado para la senal recibida
yn1_rx = np.convolve(hr1, rx_signal_normalized, 'valid')
sn1_rx = yn1_rx[np.arange(0, Ns * L, L)] # Muestreo en instantes adecuados

Nmax = min(64, len(sn1), len(sn1_rx)) # Tomar el minimo de 64 y la cantidad de simbolos
    disponibles

# ---- Constelacion transmitida ----
plt.figure(figsize=(6, 6))

# Pinta los ejes
plt.axhline(0, color='black', linewidth=1) # Eje X
plt.axvline(0, color='black', linewidth=1) # Eje Y

# Puntos de la constelacion transmitida
plt.plot(sn1[:Nmax], np.zeros_like(sn1[:Nmax]), 'bo', label="Constelacion Transmitida")

# Puntos de la constelacion recibida
plt.plot(sn1_rx[:Nmax], np.zeros_like(sn1_rx[:Nmax]), 'r*', label="Constelacion Recibida")

# Configuracion del grafico
plt.xlim(xmin, xmax)
plt.ylim(-0.5, 0.5) # La senal es unidimensional
plt.title("Constelacion Transmitida vs Recibida")
plt.xlabel("Amplitud")
plt.legend()
plt.grid()
#plt.savefig("constelacion.eps", format="eps", dpi=300)
plt.show()

# %% [markdown]
# ## Calculo del EVM (Error Vector Magnitude)
```

```

# %%
# ---- CALCULO DEL EVM ----
# Ajustar el tamano de los simbolos transmitidos y recibidos al minimo comun
N_symbols = min(len(sn1), len(sn1_rx))
sn1_tx_trimmed = sn1[:N_symbols]
sn1_rx_trimmed = sn1_rx[:N_symbols]

# Calculo del EVM en porcentaje
evm = np.sqrt(np.sum(np.abs(sn1_rx_trimmed - sn1_tx_trimmed) ** 2) / N_symbols) / np.max(np.abs(
    sn1_tx_trimmed)) * 100

print(f"EVM (%): {evm:.2f} %")
print(len(Xn))
print(len(rx_signal))

# %% [markdown]
# ## Calculo del BER (Bit Error Rate)

# %%
# ---- DECODIFICACION DE LOS SIMBOLOS RECIBIDOS ----
# Para un sistema PAM-M, el demapeo a bits se hace asignando los niveles de la constelacion a
    bits.

# Normalizar los simbolos recibidos y transmitidos
sn1_rx = sn1_rx[:Ns] # Recortar al numero de simbolos transmitidos
sn1 = sn1[:Ns]

# Obtener la constelacion PAM-M (alfabeto)
alfabeto = np.linspace(-M + 1, M - 1, M) # Simbolos PAM-M

# Asignar cada simbolo recibido al mas cercano en la constelacion
indices_rx = np.argmin(np.abs(sn1_rx[:, None] - alfabeto), axis=1).astype(int)
indices_tx = np.argmin(np.abs(sn1[:, None] - alfabeto), axis=1).astype(int)

# Convertir los indices a bits
Bn_rx = np.hstack([np.binary_repr(i, width=int(k)) for i in indices_rx]) # k debe ser entero
Bn_rx = np.array(list(''.join(Bn_rx)), dtype=int) # Convertir a array de bits

Bn_tx = np.hstack([np.binary_repr(i, width=int(k)) for i in indices_tx]) # k debe ser entero
Bn_tx = np.array(list(''.join(Bn_tx)), dtype=int) # Convertir a array de bits

# ---- CALCULO DEL BER ----
bit_errors = np.sum(Bn_rx[:len(Bn_tx)] != Bn_tx) # Contar bits erroneos
BER = bit_errors / len(Bn_tx) # Calcular tasa de error de bits

print(f"BER (Tasa de Error de Bits): {BER:.50f}")

# %%
# ---- Liberar buffer SDR ----
sdr.tx_destroy_buffer()
```

Código A.6 Transmisión y recepción de una señal PAM paso banda (Transmisión larga).

```
# %% [markdown]
# # Proyecto: Transmision grande 8PAM con PlutoSDR
# **Autor:** Alejandro Madero Vilchez
# **Fecha:** 4 de abril de 2025
# **Grado:** Ingenieria Aeroespacial

# %% [markdown]
# ## Librerias y parametros

# %%
import numpy as np
import matplotlib.pyplot as plt
from adi import Pluto
from transmisorpam import transmisorpam
from scipy.fft import fft, fftfreq
```

```
from scipy.signal import correlate

# Configuracion de parametros
sample_rate = 20e6
num_samples = 4020
center_freq = 915e6
bits_por_tanda = 394
bits_totales = 10244 #10244
num_tandas = int(bits_totales/bits_por_tanda)

# Conectar con PlutoSDR
sdr = Pluto(uri="ip:192.168.2.1")
sdr.sample_rate = int(sample_rate)

# Configuracion de transmision
sdr.tx_rf_bandwidth = int(sample_rate)
sdr.tx_lo = int(center_freq)
sdr.tx_hardwaregain_chan0 = -10

# Configuracion de recepcion
sdr.rx_lo = int(center_freq)
sdr.rx_rf_bandwidth = int(sample_rate)
sdr.rx_buffer_size = num_samples
sdr.gain_control_mode_chan0 = 'manual'
sdr.rx_hardwaregain_chan0 = 0.0

# Parametros de modulacion
Eb = 8 # Energia por bit
M = 8  # Orden PAM
k = int(np.log2(M)) # Bits por simbolo
L = 30 # Muestras por simbolo
Rb = 2e6 # Tasa de bits
Tb = 1 / Rb
Tm = (Tb * k) / L
fs = 1 / Tm
f0 = fs / L

# Forzar numero de muestras por simbolo a ser par
L = int(np.ceil(L / 2) * 2)

g = np.cos(2 * np.pi * f0 * np.arange(L) * Tm) # Pulso conformador

## IMPRIMIR TODO
print("\n--- PARAMETROS DE CONFIGURACION ---")
print(f"Sample rate: {sample_rate}")
print(f"Numero de muestras: {num_samples}")
print(f"Frecuencia central: {center_freq/1e6} MHz")
print(f"Tasa de bits (Rb): {Rb} bps")
print(f"Duracion del bit (Tb): {Tb} s")
print(f"Intervalo de muestreo (Tm): {Tm} s")
print(f"Frecuencia de muestreo (fs): {fs} Hz")
print(f"Frecuencia del pulso paso banda (f0): {f0} Hz")
print(f"Orden de PAM (M): {M}")
print(f"Numero de bits por simbolo (k): {k}")
print(f"Numero de muestras por simbolo (L): {L}")

# %% [markdown]
# ## Envio por tandas

# %%
# Almacenar transmisiones
Xn_total = []
rx_signal_total = np.zeros(num_samples * num_tandas, dtype=np.complex64) # Preasignar espacio

for i in range(num_tandas):
    preambulo = np.array([1, 0, 1, 1, 0, 1, 0, 0]) # Secuencia fija de sincronizacion
    Bn = np.concatenate([preambulo, np.random.randint(0, 2, bits_por_tanda)]) # Mensaje con
        preambulo
    [Xn, Bn, An, phi, alfabeto] = transmisorpam(Bn, Eb, M, g, L) # Modulacion PAM
    Xt = np.sqrt(1 / Tm) * Xn

```

```
    # Transmitir
    sdr.tx_cyclic_buffer = True
    sdr.tx(Xt)

    # Recibir
    rx_signal = sdr.rx()
    rx_signal = rx_signal / np.max(np.abs(rx_signal)) # Normalizar amplitud

    # ---- Estimacion del retardo mediante FFT ----
    Xf = fft(Xn)
    Xf_rx = fft(rx_signal)
    freqs = fftfreq(len(Xn), d=Tm)
    phase_diff = np.angle(Xf_rx) - np.angle(Xf)
    valid_idx = np.where((freqs > 0) & (freqs < fs / 2))
    p = np.polyfit(freqs[valid_idx], phase_diff[valid_idx], 1)
    estimated_a = -p[0] / (2 * np.pi)

    # Corregir retardo en la senal recibida
    sample_offset = int(round(estimated_a))
    rx_signal_sync = np.roll(rx_signal, -sample_offset)


    # ---- Calculo refinado del desfase ----
    Xn_norm = Xn / np.linalg.norm(Xn)
    rx_signal_norm = rx_signal / np.linalg.norm(rx_signal)

    corr = correlate(np.real(rx_signal_norm), np.real(Xn_norm), mode='full')
    lags = np.arange(-len(Xn_norm) + 1, len(rx_signal_norm)) * Tm * 1e6

    max_corr_idx = np.argmax(np.abs(corr))
    lag_max = lags[max_corr_idx]

    # Interpolacion cuadratica
    if 1 < max_corr_idx < len(corr) - 1:
        y0, y1, y2 = corr[max_corr_idx - 1], corr[max_corr_idx], corr[max_corr_idx + 1]
        shift_adjustment = (y0 - y2) / (2 * (y0 - 2 * y1 + y2))
        lag_max += shift_adjustment * Tm * 1e6

    sample_offset = int(np.round((lag_max * 1e-6) / Tm))
    print(f"Desfase optimo refinado: {lag_max:.6f} us")
    print(f"Indice de desfase en muestras: {sample_offset}")

    # Sincronizar senal recibida
    rx_signal_sync = np.roll(rx_signal, -sample_offset)

    phase_tx = np.angle(np.fft.fft(Xn))
    phase_rx = np.angle(np.fft.fft(rx_signal_sync))
    phase_diff = np.mean(phase_rx - phase_tx)

    if np.abs(phase_diff - np.pi) < 0.1: # Umbral para detectar inversion
        rx_signal_sync *= -1



    # Guardar directamente en el array sin usar np.concatenate()
    Xn_total = np.concatenate((Xn_total, Xn)) if len(Xn_total) > 0 else Xn
    rx_signal_total[i * num_samples : (i + 1) * num_samples] = rx_signal_sync

    # Liberar buffer SDR
    sdr.tx_destroy_buffer()


# %% [markdown]
# ## Calculo del NMSE

# %%
## CALCULO DEL NMSE (ERROR CUADRATICO MEDIO NORMALIZADO)
# ---- CALCULO DEL NMSE ----
min_len = min(len(Xn_total), len(rx_signal_total))
Xn_trimmed = Xn_total[:min_len] # Usar Xn en lugar de Xt
rx_signal_total = np.real(rx_signal_total) # Convertir a real
```

```
rx_signal_trimmed = rx_signal_total[:min_len]

# Normalizar las senales
Xn_normalized = Xn_trimmed / np.linalg.norm(Xn_trimmed)
rx_signal_normalized = rx_signal_trimmed / np.linalg.norm(rx_signal_trimmed)

# Calculo del NMSE
nmse = 20 * np.log10(np.linalg.norm(rx_signal_normalized - Xn_normalized) / np.linalg.norm(
    Xn_normalized))
print(f"NMSE entre la senal transmitida y recibida (en dB): {nmse:.6f} dB")


# %% [markdown]
# ## Graficar constelacion Tx y Rx

# %%
## Constelacion Tx y Rx
# Filtrado y sincronizacion
hr1 = np.flipud(phi)
yn1 = np.convolve(hr1, Xn_normalized, 'valid')
yn1_rx = np.convolve(hr1, rx_signal_normalized, 'valid')

Ns = int(len(Bn) / k) # Numero de simbolos transmitidos
sn1 = yn1[np.arange(0, Ns * L, L)]
sn1_rx = yn1_rx[np.arange(0, Ns * L, L)]

# Definir los limites de la constelacion
xmax = max(sn1) + max(sn1) / 10 + 0.5
xmin = -(abs(min(sn1)) + abs(min(sn1)) / 10) - 0.5

Nmax = min(64, len(sn1), len(sn1_rx))

# ---- Graficar la constelacion ----
plt.figure(figsize=(6, 6))
plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.plot(sn1[:Nmax], np.zeros_like(sn1[:Nmax]), 'bo', label="Constelacion Transmitida")
plt.plot(sn1_rx[:Nmax], np.zeros_like(sn1_rx[:Nmax]), 'r*', label="Constelacion Recibida")
plt.xlim(xmin, xmax)
plt.xlim(-0.15, 0.15)
plt.ylim(-0.5, 0.5)
plt.title("Constelacion Transmitida vs Recibida")
plt.xlabel("Amplitud")
plt.legend()
plt.grid()
#plt.savefig("PAM_grande_constelacion.eps", format="eps", dpi=300)
plt.show()

# %% [markdown]
# ## Graficar los primeros 8 simbolos en el tiempo

# %%
# ---- Graficar los primeros 8 simbolos en el tiempo ----
t = np.arange(0, 8 * L) * Tm

plt.figure(figsize=(10, 4))
plt.plot(t, Xn_normalized[:8 * L], label="Senal Transmitida", linestyle='-', marker='o')
plt.plot(t, rx_signal_normalized[:8 * L], label="Senal Recibida", linestyle='--', marker='x')
plt.xlabel("Tiempo (s)")
plt.ylabel("Amplitud")
plt.title("8 Primeros Simbolos en el Tiempo")
plt.legend()
plt.grid()
#plt.savefig("simbolos.eps", format="eps", dpi=300)
plt.show()


# %% [markdown]
# ## Calculo del EVM (Error Vector Magnitude)

# %%
```

```
# ---- CALCULO DEL EVM ----
# Ajustar el tamano de los simbolos transmitidos y recibidos al minimo comun
N_symbols = min(len(sn1), len(sn1_rx))
sn1_tx_trimmed = sn1[:N_symbols]
sn1_rx_trimmed = sn1_rx[:N_symbols]

# Calculo del EVM en porcentaje
evm = np.sqrt(np.sum(np.abs(sn1_rx_trimmed - sn1_tx_trimmed) ** 2) / N_symbols) / np.max(np.abs(
    sn1_tx_trimmed)) * 100

print(f"EVM (%): {evm:.2f} %")
print(len(Xn))
print(len(rx_signal))


# %% [markdown]
# ## Calculo del BER (Bit Error Rate)

# %%
# ---- DECODIFICACION DE LOS SIMBOLOS RECIBIDOS ----
# Para un sistema PAM-M, el demapeo a bits se hace asignando los niveles de la constelacion a
    bits.

# Normalizar los simbolos recibidos y transmitidos
sn1_rx = sn1_rx[:Ns] # Recortar al numero de simbolos transmitidos
sn1 = sn1[:Ns]

# Obtener la constelacion PAM-M (alfabeto)
alfabeto = np.linspace(-M + 1, M - 1, M) # Simbolos PAM-M

# Asignar cada simbolo recibido al mas cercano en la constelacion
indices_rx = np.argmin(np.abs(sn1_rx[:, None] - alfabeto), axis=1)
indices_tx = np.argmin(np.abs(sn1[:, None] - alfabeto), axis=1)

# Convertir los indices a bits
Bn_rx = np.hstack([np.binary_repr(i, width=k) for i in indices_rx])
Bn_rx = np.array(list(''.join(Bn_rx)), dtype=int) # Convertir a array de bits

Bn_tx = np.hstack([np.binary_repr(i, width=k) for i in indices_tx])
Bn_tx = np.array(list(''.join(Bn_tx)), dtype=int) # Convertir a array de bits

# ---- CALCULO DEL BER ----
bit_errors = np.sum(Bn_rx[:len(Bn_tx)] != Bn_tx) # Contar bits erroneos
BER = bit_errors / len(Bn_tx) # Calcular tasa de error de bits

print(f"BER (Tasa de Error de Bits): {BER:.30f}")

# %%
# Liberar buffer SDR
sdr.tx_destroy_buffer()
```

A.4 Transmisión y recepción de una señal PSK paso banda

Código A.7 Transmisor PSK.

```
import numpy as np
from gray2de import gray2de
from math import ceil, pi

eps = np.finfo(float).eps

def transmisorpsk(Bn, Eb, M, g1, g2, L):
    '''
    [Xn, Bn, An, phi1, phi2, alfabeto] = transmisorpsk(Bn, Eb, M, g1, g2, L)

    Parámetros de entrada:
    ----------------------
```

```
    Bn  : Secuencia de dígitos binarios
    Eb  : Energía media por bit transmitida en Julios
    M   : Número de símbolos del código PSK
    g1  : Pulso conformador real de la componente en fase
    g2  : Pulso conformador real de la componente en cuadratura
    L   : Número de puntos utilizados en la representación de un símbolo

    Salida:
    -------
    Xn       : Señal de información discreta
    Bn       : Secuencia de dígitos binarios realmente transmitidos
    An       : Secuencia de símbolos complejos transmitidos
    phi1     : Pulso básico real normalizado de la componente en fase
    phi2     : Pulso básico real normalizado de la componente en cuadratura
    alfabeto : Alfabeto utilizado asociado a cada símbolo transmitido
    '''

    # Número de bits por símbolo
    k = np.ceil(np.log2(M))

    # Ajustamos M a una potencia de dos
    M = 2**(k)

    # Generación del alfabeto de símbolos PSK
    alfabeto = np.sqrt(Eb * k) * np.exp(1j * 2 * np.pi * np.arange(M) / M)

    # Asegurar que la secuencia Bn tenga longitud múltiplo de k
    Nb = len(Bn)
    Bn = np.pad(Bn, [0, int(k * np.ceil(Nb / k) - Nb)], mode='constant')

    # Número total de bits tras la corrección
    Nb = len(Bn)

    # Número de símbolos a transmitir
    Ns = int(Nb / k)

    # Mapear bits a símbolos del alfabeto PSK
    if M > 2:
        An = alfabeto[gray2de(np.reshape(Bn, [Ns, int(k)]))]
    else:
        An = alfabeto[Bn]

    # Verificación de los pulsos conformadores
    Ls1 = len(g1)
    Ls2 = len(g2)

    if Ls1 == 0 or Ls2 == 0:
        print('No es posible realizar la transmisión')
        return

    # Ajuste de la longitud de los pulsos
    if Ls1 < L:
        g1 = np.pad(g1, [0, int(L - Ls1)], mode='constant')
    else:
        g1 = g1[:L]

    if Ls2 < L:
        g2 = np.pad(g2, [0, int(L - Ls2)], mode='constant')
    else:
        g2 = g2[:L]

    # Normalización de los pulsos
    phi1 = (1 / np.sqrt(np.sum(g1 * g1))) * g1
    phi2 = (1 / np.sqrt(np.sum(g2 * g2))) * g2

    # Verificación de ortogonalidad
    if abs(np.sum(phi1 * phi2)) >= 1e0 * eps:
        print('No es posible realizar la transmisión')
        return

    # Pulso básico complejo
```

```
    phi = phi1 - 1j * phi2

    # Obtención del tren de pulsos
    Xn = np.real(np.kron(An, phi))
    Xn = np.array(Xn)

    return [Xn, Bn, An, phi1, phi2, alfabeto]
```

Código A.8 Transmisión y recepción de una señal PSK paso banda (Transmisión corta).

```
import numpy as np
import matplotlib.pyplot as plt

# Parametros
Fc = 1000 # Frecuencia de la portadora
Fs = 10000 # Frecuencia de muestreo
Tb = 1/1000 # Tiempo por bit
N = 8  # Numero de bits

# Generacion de bits aleatorios
bits = np.random.randint(0, 2, N)
print("Bits transmitidos:", bits)

# Tiempo por bit
t = np.linspace(0, Tb, int(Fs*Tb), endpoint=False)

# Senales para bit 0 y bit 1
portadora_0 = np.cos(2*np.pi*Fc*t)
portadora_1 = np.cos(2*np.pi*Fc*t + np.pi) # desfase de 180 grados

# Senal modulada
senal = np.array([])
for bit in bits:
    if bit == 0:
        senal = np.concatenate((senal, portadora_0))
    else:
        senal = np.concatenate((senal, portadora_1))

# Tiempo total
t_total = np.linspace(0, Tb*N, int(Fs*Tb*N), endpoint=False)

# Representacion de la senal
plt.figure(figsize=(10, 4))
plt.plot(t_total, senal)
plt.title("Senal PSK")
plt.xlabel("Tiempo (s)")
plt.ylabel("Amplitud")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Código A.9 Transmisión y recepción de una señal PSK paso banda (Transmisión larga).

```
# %% [markdown]
# # Proyecto: Transmision grande 8PSK con PlutoSDR
# **Autor:** Alejandro Madero Vilchez
# **Fecha:** 4 de abril de 2025
# **Grado:** Ingenieria Aeroespacial

# %% [markdown]
# ## Librerias y parametros

# %%
import numpy as np
import matplotlib.pyplot as plt
from adi import Pluto
from transmisorpsk import transmisorpsk
```

```
from scipy.signal import correlate
import scipy.io as sio
import time

# Configuracion de parametros
sample_rate = 20e6
num_samples = 4020
center_freq = 915e6
bits_por_tanda = 394
bits_totales = 10244
num_tandas = int(bits_totales / bits_por_tanda)

# Conectar con PlutoSDR
sdr = Pluto(uri="ip:192.168.2.1")
sdr.sample_rate = int(sample_rate)

# Configuracion de transmision
sdr.tx_rf_bandwidth = int(sample_rate)
sdr.tx_lo = int(center_freq)
sdr.tx_hardwaregain_chan0 = -10

# Configuracion de recepcion
sdr.rx_lo = int(center_freq)
sdr.rx_rf_bandwidth = int(sample_rate)
#sdr.rx_buffer_size = num_samples
sdr.rx_buffer_size = 2*num_samples
sdr.gain_control_mode_chan0 = 'manual'
sdr.rx_hardwaregain_chan0 = 10.0


# Parametros de modulacion
Eb = 8 # Energia por bit
M = 8  # Orden PAM
k = int(np.log2(M)) # Bits por simbolo
L = 30 # Muestras por simbolo
Rb = 2e6 # Tasa de bits
Tb = 1 / Rb
Tm = (Tb * k) / L
fs = 1 / Tm
f0 = fs / L

# Forzar numero de muestras por simbolo a ser par
L = int(np.ceil(L / 2) * 2)


# Definicion de los pulsos basicos
g1 = np.r_[np.ones(int(L / 2)), np.zeros(int(L / 2))]
g2 = np.r_[np.zeros(int(L / 2)), np.ones(int(L / 2))]

## IMPRIMIR TODO
print("\n--- PARAMETROS DE CONFIGURACION ---")
print(f"Sample rate: {sample_rate}")
print(f"Numero de muestras: {num_samples}")
print(f"Frecuencia central: {center_freq/1e6} MHz")
print(f"Tasa de bits (Rb): {Rb} bps")
print(f"Duracion del bit (Tb): {Tb} s")
print(f"Intervalo de muestreo (Tm): {Tm} s")
print(f"Frecuencia de muestreo (fs): {fs} Hz")
print(f"Frecuencia del pulso paso banda (f0): {f0} Hz")
print(f"Orden de PAM (M): {M}")
print(f"Numero de bits por simbolo (k): {k}")
print(f"Numero de muestras por simbolo (L): {L}")

# %% [markdown]
# ## Envios por tandas

# %%
# Almacenar transmisiones
Xn_total = []
rx_signal_total = np.zeros(num_samples * num_tandas, dtype=np.complex64)

```

```
for i in range(num_tandas):
    preambulo = np.random.randint(0, 2, 32)
    Bn = np.concatenate([preambulo, np.random.randint(0, 2, bits_por_tanda)])
    Ns = int(len(Bn) / k)
    Nb = len(Bn)
    Td = Tb * Nb
    t = np.arange(0, Td, Tm)

    [Xn, Bn, An, phi1, phi2, alfabeto] = transmisorpsk(Bn, Eb, M, g1, g2, L)
    Xt = np.sqrt(1 / Tm) * Xn

    sdr.tx_cyclic_buffer = True
    sdr.tx(np.zeros((len(Xt))))
    sdr.tx_destroy_buffer()
    sdr.tx_cyclic_buffer = True
    sdr.tx(Xt)
    time.sleep(1)

    for _ in range(5):
        _ = sdr.rx()
    rx_signal = sdr.rx()
    rx_signal = rx_signal - np.mean(rx_signal) # Restar la media
    rx_signal = rx_signal / np.max(np.abs(rx_signal))

    Xn_norm = Xn / np.linalg.norm(Xn)
    rx_signal_norm = rx_signal / np.linalg.norm(rx_signal)

    # Correlacion cruzada entre la senal recibida y transmitida
    corr = correlate(np.real(rx_signal_norm), np.real(Xn_norm), mode='full')
    lags = np.arange(-len(Xn_norm) + 1, len(rx_signal_norm)) * Tm * 1e6

    # Encontrar el maximo de la correlacion
    max_corr_idx = np.argmax(np.abs(corr)) # indice del maximo
    lag_max = lags[max_corr_idx] # retardo correspondiente

    # Refinamiento de desfase (ajuste cuadratico)
    if 1 < max_corr_idx < len(corr) - 1:
        y0, y1, y2 = corr[max_corr_idx - 1], corr[max_corr_idx], corr[max_corr_idx + 1]
        shift_adjustment = (y0 - y2) / (2 * (y0 - 2 * y1 + y2))
        lag_max += shift_adjustment * Tm * 1e6

    # Calculamos el desfase en muestras
    sample_offset = (lag_max * 1e-6) / Tm
    print(f"Desfase optimo refinado: {lag_max:.6f} us")
    print(f"Indice de desfase en muestras (fraccional): {sample_offset:.2f}")


    # Sincronizar la senal con el desfase calculado
    rx_signal = np.roll(rx_signal, -int(np.round(sample_offset)))
    rx_signal_sync = rx_signal[:num_samples]

    # Verificacion de inversion de fase utilizando `max(xorr(x, y))`
    # Calculamos la correlacion de la senal sincronizada
    corr_with_inversion = correlate(np.real(rx_signal_sync), np.real(Xn_norm), mode='full')
    corr_original = np.max(corr)
    corr_inverted = np.max(corr_with_inversion)

    # Si la correlacion con inversion es mayor, invertimos la senal
    if corr_inverted > corr_original: # Detectamos la inversion de fase
        print("Inversion de fase detectada, corrigiendo...")
        rx_signal_sync = -rx_signal_sync


    # Almacenamos la senal sincronizada
    Xn_total = np.concatenate((Xn_total, Xn)) if len(Xn_total) > 0 else Xn
    rx_signal_total[i * num_samples : (i + 1) * num_samples] = rx_signal_sync

    # Liberar el buffer del transmisor
    sdr.tx_destroy_buffer()
```

```
# %% [markdown]
# ## Calculo del NMSE

# %%
# ---- CALCULO DEL NMSE ----
rx_signal_sync = rx_signal_sync * (-1)
min_len = min(len(Xn), len(rx_signal_sync))
Xn_trimmed = Xn[:min_len] # Usar Xn en lugar de Xt
rx_signal_sync = np.real(rx_signal_sync) # Convertir a real
rx_signal_trimmed = rx_signal_sync[:min_len]

# Normalizar las senales
Xn_normalized = Xn_trimmed / np.linalg.norm(Xn_trimmed)
rx_signal_normalized = rx_signal_trimmed / np.linalg.norm(rx_signal_trimmed)

# Calculo del NMSE
nmse = 20 * np.log10(np.linalg.norm(rx_signal_normalized - Xn_normalized) / np.linalg.norm(
    Xn_normalized))
print(f"NMSE entre la senal transmitida y recibida (en dB): {nmse:.6f} dB")

# Verificacion de normalizacion
print(f"Norma de Xn_normalized: {np.linalg.norm(Xn_normalized)}")
print(f"Norma de rx_signal_normalized: {np.linalg.norm(rx_signal_normalized)}")
print(f"Tamano de Xn_normalized: {len(Xn_normalized)}")
print(f"Tamano de rx_signal_normalized: {len(rx_signal_normalized)}")


# %% [markdown]
# ## Representacion de los 8 primeros simbolos en el tiempo

# %%
# ---- Graficar los primeros 8 simbolos en el tiempo ----
t = np.arange(0, 8 * L) * Tm

plt.figure(figsize=(10, 4))
plt.plot(t, Xn_normalized[:8 * L], label="Senal Transmitida", linestyle='-', marker='o')
plt.plot(t, rx_signal_normalized[:8 * L], label="Senal Recibida", linestyle='--', marker='x')
plt.xlabel("Tiempo (s)")
plt.ylabel("Amplitud")
plt.title("8 Primeros Simbolos en el Tiempo")
plt.legend()
plt.grid()
plt.savefig("PSK_grande_simbolos.eps", format="eps", dpi=300)
plt.show()

# ---- Verificacion de tipos y partes imaginarias ----
print(f"Tipo de Xn: {Xn.dtype}, Tipo de rx_signal: {rx_signal.dtype}")
print(f"Maximo de la parte imaginaria de Xn: {np.max(np.abs(np.imag(Xn)))}")
print(f"Maximo de la parte imaginaria de rx_signal: {np.max(np.abs(np.imag(rx_signal)))}")

# %% [markdown]
# ## Representacion de las senales normalizadas en el tiempo

# %%
# ---- REPRESENTACION DE LAS SENALES NORMALIZADAS ----
plt.figure(figsize=(12, 4))
plt.plot(np.real(Xn_normalized), label="Senal transmitida normalizada", color='b')
plt.plot(np.real(rx_signal_normalized), label="Senal recibida normalizada", color='r', linestyle='
    dashed')
plt.title("Comparacion de senales normalizadas")
plt.xlabel("Muestras")
plt.ylabel("Amplitud")
plt.legend()
plt.grid()
plt.savefig("PSK_grande_senales_normalizadas.eps", format="eps", dpi=300)
plt.show()

# %% [markdown]
# ## Constelacion Tx y Rx

# %%
```

```
# CONSTELACION DE BITS Tx Y Rx
hr1 = np.flipud(phi1) # Filtro de recepcion en eje X
hr2 = np.flipud(phi2) # Filtro de recepcion en eje Y

# Filtrado adaptado para la senal transmitida
yn1 = np.convolve(hr1, Xn_normalized, 'valid')
yn2 = np.convolve(hr2, Xn_normalized, 'valid')

# Muestreo en los instantes adecuados
valid_indices = np.arange(0, min(len(yn1), Ns * L), L)
sn1 = yn1[valid_indices]
sn2 = yn2[valid_indices]

# Obtiene valores maximos y minimos para la representacion
xmax = max(sn1) + max(sn1) / 10 + 0.5
xmin = -(abs(min(sn1)) + abs(min(sn1)) / 10) - 0.5
ymax = max(sn2) + max(sn2) / 10 + 0.5
ymin = -(abs(min(sn2)) + abs(min(sn2)) / 10) - 0.5

# Filtrado adaptado para la senal recibida
yn1_rx = np.convolve(hr1, rx_signal_normalized, 'valid')
yn2_rx = np.convolve(hr2, rx_signal_normalized, 'valid')

# Muestreo en los instantes adecuados
valid_indices_rx = np.arange(0, min(len(yn1_rx), Ns * L), L)
sn1_rx = yn1_rx[valid_indices_rx]
sn2_rx = yn2_rx[valid_indices_rx]

# Tomar el minimo entre 64 y la cantidad de simbolos disponibles
Nmax = min(64, len(sn1), len(sn2), len(sn1_rx), len(sn2_rx))

# ---- Constelacion transmitida vs recibida ----
plt.figure(figsize=(6, 6))

# Pinta los ejes
plt.axhline(0, color='black', linewidth=1) # Eje X
plt.axvline(0, color='black', linewidth=1) # Eje Y

# Puntos de la constelacion transmitida
plt.plot(sn1[:Nmax], sn2[:Nmax], 'bo', label="Constelacion Transmitida")

# Puntos de la constelacion recibida
plt.plot(sn1_rx[:Nmax], sn2_rx[:Nmax], 'r*', label="Constelacion Recibida")

# Configuracion del grafico
plt.xlim(xmin, xmax)
plt.ylim(ymin, ymax)
plt.title("Constelacion Transmitida vs Recibida")
plt.xlabel("Eje I")
plt.ylabel("Eje Q")
plt.legend()
plt.grid()
plt.savefig("PSK_grande_constelacion.eps", format="eps", dpi=300)
plt.show()


# %% [markdown]
# ## Calculo del EVM (Error Vector Magnitude)

# %%
# ---- CALCULO DEL EVM ----
# Ajustar el tamano de los simbolos transmitidos y recibidos al minimo comun
N_symbols = min(len(sn1), len(sn1_rx))
sn1_tx_trimmed = sn1[:N_symbols]
sn1_rx_trimmed = sn1_rx[:N_symbols]

# Calculo del EVM en porcentaje
evm = np.sqrt(np.sum(np.abs(sn1_rx_trimmed - sn1_tx_trimmed) ** 2) / N_symbols) / np.max(np.abs(
    sn1_tx_trimmed)) * 100

print(f"EVM (%): {evm:.2f} %")
```

```
print(len(Xn))
print(len(rx_signal))

# %% [markdown]
# ## Calculo del BER (Bit Error Rate)

# %%
# ---- DECODIFICACION DE LOS SIMBOLOS RECIBIDOS PARA 8-PSK ----

# Numero de simbolos (8-PSK)
M = 8
k = int(np.log2(M)) # bits por simbolo

# Asegurar longitud correcta
sn1_rx = sn1_rx[:Ns]
sn1 = sn1[:Ns]

# Constelacion 8-PSK ideal
alfabeto = np.exp(1j * 2 * np.pi * np.arange(M) / M)

# Funcion para encontrar el indice del punto mas cercano en la constelacion
def nearest_symbol_indices(symbols, constelacion):
    return np.argmin(np.abs(symbols[:, None] - constelacion), axis=1)

# Obtener los indices de simbolos transmitidos y recibidos
indices_rx = nearest_symbol_indices(sn1_rx, alfabeto)
indices_tx = nearest_symbol_indices(sn1, alfabeto)

# Convertir indices a bits
Bn_rx = np.hstack([np.binary_repr(i, width=k) for i in indices_rx])
Bn_rx = np.array(list(''.join(Bn_rx)), dtype=int)

Bn_tx = np.hstack([np.binary_repr(i, width=k) for i in indices_tx])
Bn_tx = np.array(list(''.join(Bn_tx)), dtype=int)

# ---- CALCULO DEL BER ----
bit_errors = np.sum(Bn_rx[:len(Bn_tx)] != Bn_tx)
BER = bit_errors / len(Bn_tx)

print(f"BER (Tasa de Error de Bits, 8-PSK): {BER:.30f}")

# %%
# Liberar buffer SDR
sdr.tx_destroy_buffer()
```

A.5 Transmisión y recepción de una señal QAM paso banda

Código A.10 Split QAM.

```
import numpy as np

def split(Bn, M1, M2):
    """
    Divide una secuencia de símbolos binarios en dos componentes (fase y cuadratura).

    Parámetros:
    Bn  - Secuencia de símbolos binarios.
    M1  - Número de símbolos de la componente en fase.
    M2  - Número de símbolos de la componente en cuadratura.

    Retorna:
    BnI - Secuencia de símbolos binarios de la componente en fase.
    BnQ - Secuencia de símbolos binarios de la componente en cuadratura.
    """
    k1 = int(np.log2(M1))
    k2 = int(np.log2(M2))
    k = k1 + k2
```

```
19
20      # Longitud de la secuencia
21      Nb = len(Bn)
22
23      if Nb % k != 0:
24          raise ValueError("La longitud de Bn debe ser múltiplo de k1 + k2")
25
26      # Reshape a matriz con Nb/k filas y k columnas
27      W = np.reshape(Bn, (Nb // k, k))
28
29      # Extraer la componente en fase
30      BnI = np.reshape(W[:, :k1], (k1 * (Nb // k)))
31
32      # Extraer la componente en cuadratura
33      BnQ = np.reshape(W[:, k1:], (k2 * (Nb // k)))
34
35      return BnI, BnQ
```

Código A.11 Transmisor QAM.

```
1  import numpy as np
2  from gray2de import gray2de
3  from math import ceil
4  from split import split
5
6  eps = np.finfo(float).eps
7
8  def transmisorqam(Bn, Eb, M1, M2, g1, g2, L):
9      """
10     [Xn, BnI, BnQ, AnI, AnQ, AI, AQ, phi1, phi2] = transmisorqam(Bn, Eb, M1, M2, g1, g2, L)
11
12     Parámetros:
13     Bn  - Secuencia de dígitos binarios
14     Eb  - Energía media transmitida en Julios
15     M1  - Número de símbolos de la componente en fase
16     M2  - Número de símbolos de la componente en cuadratura
17     g1  - Pulso conformador real para transmitir los símbolos en fase
18     g2  - Pulso conformador real para transmitir los símbolos en cuadratura
19     L   - Número de puntos que vamos a utilizar en la representación de un símbolo
20
21     Retorna:
22     Xn  - Señal de información digital
23     BnI - Bits transmitidos por la componente en fase
24     BnQ - Bits transmitidos por la componente en cuadratura
25     AnI - Niveles de amplitud transmitidos por la componente en fase
26     AnQ - Niveles de amplitud transmitidos por la componente en cuadratura
27     AI  - Niveles de amplitud usados en la componente en fase
28     AQ  - Niveles de amplitud usados en la componente en cuadratura
29     phi1 - Pulso básico normalizado usado en la componente en fase
30     phi2 - Pulso básico normalizado usado en la componente en cuadratura
31     """
32     # Ajustemos los parámetros
33     k1 = int(np.ceil(np.log2(M1))) # Número de bits de la componente en fase
34     M1 = 2**k1 # Valor de M1 tras la corrección
35     k2 = int(np.ceil(np.log2(M2))) # Número de bits de la componente en cuadratura
36     M2 = 2**k2 # Valor de M2 tras la corrección
37     k = k1 + k2 # Número de bits en cada símbolo QAM
38     Nb = len(Bn)
39
40     # Asegurar que la longitud de Bn sea múltiplo de k
41     Bn = np.pad(Bn, [0, int(k * np.ceil(Nb / k) - Nb)], mode='constant')
42
43     # Forcemos que el número de muestras por bit sea par
44     L = int(np.ceil(L / 2) * 2)
45
46     # Comprobación de las longitudes y otros datos de los pulsos básicos
47     Ls1 = len(g1)
48     Ls2 = len(g2)
49
```

```
    if Ls1 == 0 or Ls2 == 0:
        print('No es posible realizar la transmisión')
        return

    if Ls1 < L:
        g1 = np.r_[g1, np.zeros(L - Ls1)]
    else:
        g1 = g1[:L]

    if Ls2 < L:
        g2 = np.r_[g2, np.zeros(L - Ls2)]
    else:
        g2 = g2[:L]

    # Normalicemos las energías de los pulsos
    phi1 = (1 / np.sqrt(np.sum(g1 * g1))) * g1
    phi2 = (1 / np.sqrt(np.sum(g2 * g2))) * g2

    # Comprobemos la ortogonalidad
    if np.abs(np.sum(phi1 * phi2)) >= 1e0 * eps:
        print('No es posible realizar la transmisión')
        return

    # Obtener los niveles de amplitud de las componentes en fase y cuadratura
    A = np.sqrt(3 * Eb * np.log2(M1 * M2) / (M1**2 + M2**2 - 2))

    # El alfabeto con los niveles
    AI = A * (2 * np.arange(M1) - M1 + 1)
    AQ = A * (2 * np.arange(M2) - M2 + 1)

    # Dividir la secuencia en las componentes en fase y cuadratura
    BnI, BnQ = split(Bn, M1, M2)
    NbI = len(BnI)
    NbQ = len(BnQ)

    # Obtener la secuencia de símbolos
    if M1 > 2:
        AnI = AI[gray2de(np.reshape(BnI, [int(NbI / k1), k1]))]
    else:
        AnI = AI[BnI]

    if M2 > 2:
        AnQ = AQ[gray2de(np.reshape(BnQ, [int(NbQ / k2), k2]))]
    else:
        AnQ = AQ[BnQ]

    # Las componentes en fase, cuadratura y total
    XnI = np.kron(AnI, phi1)
    XnQ = np.kron(AnQ, phi2)
    Xn = XnI + XnQ

    return Xn, BnI, BnQ, AnI, AnQ, AI, AQ, phi1, phi2
```

Código A.12 Transmisión y recepción de una señal QAM paso banda (Transmisión corta).

```
# %% [markdown]
# # Proyecto: Transmision pequena 16QAM con PlutoSDR
# **Autor:** Alejandro Madero Vilchez
# **Fecha:** 4 de abril de 2025
# **Grado:** Ingenieria Aeroespacial

# %% [markdown]
# ## Librerias y parametros

# %%
import numpy as np
import matplotlib.pyplot as plt
from adi import Pluto
from transmisorqam import transmisorqam
```

```
from scipy.signal import correlate,welch
import scipy.io as sio
import time

# Configuracion de parametros
sample_rate = 20e6
num_samples = 4020
center_freq = 915e6

# Conectar con PlutoSDR
sdr = Pluto(uri="ip:192.168.2.1")
sdr.sample_rate = int(sample_rate)

# Configuracion de transmision
sdr.tx_rf_bandwidth = int(sample_rate)
sdr.tx_lo = int(center_freq)
sdr.tx_hardwaregain_chan0 = -10

# Configuracion de recepcion
sdr.rx_lo = int(center_freq)
sdr.rx_rf_bandwidth = int(sample_rate)
sdr.rx_buffer_size = 2*num_samples
sdr.gain_control_mode_chan0 = 'manual'
sdr.rx_hardwaregain_chan0 = 10.0

# Parametros de la modulacion
Eb = 8 # Energia media transmitida
L = 40 # Numero de muestras por simbolo
Rb = 2 * 10**6 # Tasa de bits
Tb = 1 / Rb
M = 16 # Orden de la modulacion (16-QAM)
k = int(np.log2(M)) # Bits por simbolo
Tm = (Tb * k) / L
M1 = 4 # Niveles en la componente en fase
M2 = 4 # Niveles en la componente en cuadratura
fs = 1 / Tm
f0 = fs / L

# Forzar numero de muestras por simbolo a ser par
L = int(np.ceil(L / 2) * 2)

# Pulsos basicos ortogonales
g1 = np.r_[np.ones(int(L/2)), np.zeros(int(L/2))] # Componente en fase
g2 = np.r_[np.zeros(int(L/2)), np.ones(int(L/2))] # Componente en cuadratura

## IMPRIMIR TODO
print("\n--- PARAMETROS DE CONFIGURACION ---")
print(f"Sample rate: {sample_rate}")
print(f"Numero de muestras: {num_samples}")
print(f"Frecuencia central: {center_freq/1e6} MHz")
print(f"Tasa de bits (Rb): {Rb} bps")
print(f"Duracion del bit (Tb): {Tb} s")
print(f"Intervalo de muestreo (Tm): {Tm} s")
print(f"Frecuencia de muestreo (fs): {fs} Hz")
print(f"Frecuencia del pulso paso banda (f0): {f0} Hz")
print(f"Orden de PAM (M): {M}")
print(f"M1: {M1}")
print(f"M2: {M2}")
print(f"Numero de bits por simbolo (k): {k}")
print(f"Numero de muestras por simbolo (L): {L}")

# %% [markdown]
# ## Transmision y recepcion

# %%
preambulo = np.array([1, 0, 1, 1, 0, 1, 0, 0]) # Secuencia de sincronizacion
Bn = np.concatenate([preambulo, np.random.randint(0, 2, 394)]) # Mensaje con preambulo

# Definicion del tiempo continuo
Ns = int(len(Bn) / k)
Nb = len(Bn) # Numero de bits transmitidos
```

```
Td = Tb * Nb # Duracion total de la senal
t = np.arange(0, Td, Tm) # Vector de tiempo

# Generacion de la senal modulada 16-QAM
[Xn, BnI, BnQ, AnI, AnQ, AI, AQ, phi1, phi2] = transmisorqam(Bn, Eb, M1, M2, g1, g2, L)

# Normalizar la senal para transmision con PlutoSDR
Xt = np.sqrt(1 / Tm) * Xn

sdr.tx_cyclic_buffer = True
sdr.tx(np.zeros((len(Xt))))
sdr.tx_destroy_buffer()
sdr.tx_cyclic_buffer = True
sdr.tx(Xt)
time.sleep(1)

for _ in range(5):
    _ = sdr.rx()
rx_signal = sdr.rx()
rx_signal = rx_signal - np.mean(rx_signal) # Restar la media
rx_signal = rx_signal / np.max(np.abs(rx_signal))

Xn_norm = Xn / np.linalg.norm(Xn)
rx_signal_norm = rx_signal / np.linalg.norm(rx_signal)

# Correlacion cruzada entre la senal recibida y transmitida
corr = correlate(np.real(rx_signal_norm), np.real(Xn_norm), mode='full')
lags = np.arange(-len(Xn_norm) + 1, len(rx_signal_norm)) * Tm * 1e6

# Encontrar el maximo de la correlacion
max_corr_idx = np.argmax(np.abs(corr)) # indice del maximo
lag_max = lags[max_corr_idx] # retardo correspondiente

# Refinamiento de desfase (ajuste cuadratico)
if 1 < max_corr_idx < len(corr) - 1:
    y0, y1, y2 = corr[max_corr_idx - 1], corr[max_corr_idx], corr[max_corr_idx + 1]
    shift_adjustment = (y0 - y2) / (2 * (y0 - 2 * y1 + y2))
    lag_max += shift_adjustment * Tm * 1e6

# Calculamos el desfase en muestras
sample_offset = (lag_max * 1e-6) / Tm
print(f"Desfase optimo refinado: {lag_max:.6f} us")
print(f"Indice de desfase en muestras (fraccional): {sample_offset:.2f}")


# Sincronizar la senal con el desfase calculado
rx_signal = np.roll(rx_signal, -int(np.round(sample_offset)))
rx_signal_sync = rx_signal[:num_samples]

# Verificacion de inversion de fase utilizando `max(xorr(x, y))`
# Calculamos la correlacion de la senal sincronizada
corr_with_inversion = correlate(np.real(rx_signal_sync), np.real(Xn_norm), mode='full')
corr_original = np.max(corr)
corr_inverted = np.max(corr_with_inversion)

# Si la correlacion con inversion es mayor, invertimos la senal
if corr_inverted > corr_original: # Detectamos la inversion de fase
    print("Inversion de fase detectada, corrigiendo...")
    rx_signal_sync = -rx_signal_sync


# ---- Separar en simbolos ----
Xn_reshaped = Xn[:len(Xn) - (len(Xn) % L)].reshape(-1, L)
rx_signal_reshaped = rx_signal_sync[:len(rx_signal_sync) - (len(rx_signal_sync) % L)].reshape(-1,
    L)

num_symbols_tx = Xn_reshaped.shape[0]
num_symbols_rx = rx_signal_reshaped.shape[0]

# ---- Codigo de colores ----
num_colors = max(num_symbols_tx, num_symbols_rx)
```

```
colors_tx = plt.cm.get_cmap("tab10", num_colors)
colors_rx = plt.cm.get_cmap("tab10", num_colors)


# %% [markdown]
# ## Representacion de los 8 primeros simbolos en el tiempo

# %%
# ---- 8 PRIMERO SIMBOLOS DE LA SENAL TRANSMITIDA ----
num_symbols_to_plot = 8
plt.figure(figsize=(12, 4))
for i in range(num_symbols_to_plot):
    tiempo = t[i * L: (i + 1) * L] * 1e6
    plt.plot(tiempo, np.real(Xn_reshaped[i, :]), color=colors_tx(i), label=f'Simbolo {i+1}')

plt.title("8 primeros simbolos de la senal transmitida")
plt.xlabel("Tiempo (us)")
plt.ylabel("Amplitud")
plt.legend()
plt.grid()
plt.savefig("QAM_pequeno_simbolos_tx.eps", format="eps", dpi=300)

# ---- 8 PRIMEROS SIMBOLOS DE LA SENAL RECIBIDA ----
plt.figure(figsize=(12, 4))
for i in range(num_symbols_to_plot):
    tiempo = t[i * L: (i + 1) * L] * 1e6
    plt.plot(tiempo, np.real(rx_signal_reshaped[i, :]), color=colors_rx(i), label=f'Simbolo {i+1}'
        )

plt.title("8 primeros simbolos de la senal recibida")
plt.xlabel("Tiempo (us)")
plt.ylabel("Amplitud")
plt.legend()
plt.grid()
plt.savefig("QAM_pequeno_simbolos_rx.eps", format="eps", dpi=300)
plt.show()

# %% [markdown]
# ## Representacion del espectro FFT

# %%
# ---- ESPECTRO DE LA SENAL TRANSMITIDA ----
Xf = np.fft.fft(Xt)
freqs = np.fft.fftfreq(len(Xt), d=Tm) / 1e6

plt.figure(figsize=(12, 4))
plt.plot(freqs, np.abs(Xf), color='b', linewidth=1.5)
plt.title("Espectro de la senal transmitida")
plt.xlabel("Frecuencia (MHz)")
plt.ylabel("Magnitud |X(f)|")
plt.grid()
plt.savefig("QAM_pequeno_espectro_tx.eps", format="eps", dpi=300)


# ---- ESPECTRO DE LA SENAL RECIBIDA ----
Xf_rx = np.fft.fft(rx_signal_sync)
freqs_rx = np.fft.fftfreq(len(rx_signal_sync), d=Tm) / 1e6

plt.figure(figsize=(12, 4))
plt.plot(freqs_rx, np.abs(Xf_rx), color='r', linewidth=1.5)
plt.title("Espectro de la senal recibida sincronizada")
plt.xlabel("Frecuencia (MHz)")
plt.ylabel("Magnitud |X(f)|")
plt.grid()
plt.savefig("QAM_pequeno_espectro_rx.eps", format="eps", dpi=300)
plt.show()

# %% [markdown]
# ## Representacion del espectro WELCH

# %%
```

```
# ---- ESPECTRO DE FRECUENCIA TRANSMITIDA PWELCH ----
f_tx, Pxx_tx = welch(Xt, fs=1/Tm, nperseg=1024)

plt.figure(figsize=(12, 4))
plt.semilogy(f_tx / 1e6, Pxx_tx, color='b', linewidth=1.5) # Escala logaritmica
plt.title("Espectro de potencia de la senal transmitida (pwelch)")
plt.xlabel("Frecuencia (MHz)")
plt.ylabel("Densidad espectral de potencia (dB/Hz)")
plt.grid()
#lt.savefig("welch_tx.eps", format="eps", dpi=300)

# ---- ESPECTRO DE FRECUENCIA RECIBIDA PWELCH ----
f_rx, Pxx_rx = welch(rx_signal_sync, fs=1/Tm, nperseg=1024)

plt.figure(figsize=(12, 4))
plt.semilogy(f_rx / 1e6, Pxx_rx, color='r', linewidth=1.5) # Escala logaritmica
plt.title("Espectro de potencia de la senal recibida (pwelch)")
plt.xlabel("Frecuencia (MHz)")
plt.ylabel("Densidad espectral de potencia (dB/Hz)")
plt.grid()
#plt.savefig("welch_rx.eps", format="eps", dpi=300)
plt.show()


#### AMBAS
# Filtrar solo las frecuencias positivas
mask_tx = f_tx >= 0
mask_rx = f_rx >= 0
f_tx_positive = f_tx[mask_tx]
Pxx_tx_positive = Pxx_tx[mask_tx]
f_rx_positive = f_rx[mask_rx]
Pxx_rx_positive = Pxx_rx[mask_rx]

# Graficar ambos espectros en una misma figura
plt.figure(figsize=(12, 4))
plt.semilogy(f_tx_positive / 1e6, Pxx_tx_positive, color='b', linewidth=1.5, label="Transmitida")
plt.semilogy(f_rx_positive / 1e6, Pxx_rx_positive, color='r', linewidth=1.5, label="Recibida")

plt.title("Comparacion del espectro de potencia (pwelch)")
plt.xlabel("Frecuencia (MHz)")
plt.ylabel("Densidad espectral de potencia (dB/Hz)")
plt.legend()
plt.grid()
plt.savefig("QAM_pequeno_welch_ambas.eps", format="eps", dpi=300)
plt.show()

# %% [markdown]
# ## Caculo del valor NMSE y representacion de senales normalizadas

# %%
## CALCULO DEL NMSE (ERROR CUADRATICO MEDIO NORMALIZADO)
# ---- CALCULO DEL NMSE ----
min_len = min(len(Xn), len(rx_signal_sync))
Xn_trimmed = Xn[:min_len] # Usar Xn en lugar de Xt
rx_signal_sync = np.real(rx_signal_sync) # Convertir a real
rx_signal_trimmed = rx_signal_sync[:min_len]

# Normalizar las senales
Xn_normalized = Xn_trimmed / np.linalg.norm(Xn_trimmed)
rx_signal_normalized = rx_signal_trimmed / np.linalg.norm(rx_signal_trimmed)

# Calculo del NMSE
nmse = 20 * np.log10(np.linalg.norm(rx_signal_normalized - Xn_normalized) / np.linalg.norm(
    Xn_normalized))
print(f"NMSE entre la senal transmitida y recibida (en dB): {nmse:.6f} dB")

# Verificacion de normalizacion
print(f"Norma de Xn_normalized: {np.linalg.norm(Xn_normalized)}")
print(f"Norma de rx_signal_normalized: {np.linalg.norm(rx_signal_normalized)}")
print(f"Tamano de Xn_normalized: {len(Xn_normalized)}")
print(f"Tamano de rx_signal_normalized: {len(rx_signal_normalized)}")
```

```
296
297 # ---- REPRESENTACION DE LAS SENALES NORMALIZADAS ----
298 plt.figure(figsize=(12, 4))
299 plt.plot(np.real(Xn_normalized), label="Senal transmitida normalizada", color='b')
300 plt.plot(np.real(rx_signal_normalized), label="Senal recibida normalizada", color='r', linestyle='
      dashed')
301 plt.title("Comparacion de senales normalizadas")
302 plt.xlabel("Muestras")
303 plt.ylabel("Amplitud")
304 plt.legend()
305 plt.grid()
306 plt.savefig("QAM_pequeno_senales_normalizadas.eps", format="eps", dpi=300)
307 plt.show()
308
309 # ---- Verificacion de tipos y partes imaginarias ----
310 print(f"Tipo de Xn: {Xn.dtype}, Tipo de rx_signal: {rx_signal.dtype}")
311 print(f"Maximo de la parte imaginaria de Xn: {np.max(np.abs(np.imag(Xn)))}")
312 print(f"Maximo de la parte imaginaria de rx_signal: {np.max(np.abs(np.imag(rx_signal)))}")
313
314
315 ## GRAFICA DE SENALES NORMALIZADAS CON MARKERS
316 # Calcular diferencias significativas entre las senales
317 diferencias = np.abs(np.real(Xn_normalized) - np.real(rx_signal_normalized))
318 umbral = 1e-3 # Umbral para considerar que los puntos no coinciden
319 indices_diferencia = np.where(diferencias > umbral)[0] # Indices donde hay diferencias
320
321 # Graficar senales
322 plt.figure(figsize=(12, 4))
323 plt.plot(np.real(Xn_normalized), label="Senal transmitida normalizada", color='b')
324 plt.plot(np.real(rx_signal_normalized), label="Senal recibida normalizada", color='r', linestyle='
      dashed')
325
326 # Agregar marcadores en los puntos de diferencia
327 plt.scatter(indices_diferencia, np.real(Xn_normalized[indices_diferencia]),
328             color='black', marker='x', label="Diferencias")
329
330 # Configuracion de la grafica
331 plt.title("Comparacion de senales normalizadas con diferencias marcadas")
332 plt.xlabel("Muestras")
333 plt.ylabel("Amplitud")
334 plt.legend()
335 plt.grid()
336 plt.savefig("QAM_pequeno_markers.eps", format="eps", dpi=300)
337 plt.show()
338
339 # %% [markdown]
340 # ## Constelacion Tx y Rx
341
342 # %%
343 # CONSTELACION DE BITS Tx Y Rx
344 # Obtencion de los parametros necesarios
345 L = len(phi1) # Numero de muestras por simbolo
346 Ns = int((len(BnI) + len(BnQ)) / (np.log2(len(AI) * len(AQ)))) # Numero de simbolos transmitidos
347
348 # Obtencion de las secuencias de niveles transmitidos
349 # Calculamos el tamano nuevo asegurando que sea multiplo de L
350 new_size = (len(Xn_normalized) // L) * L
351 padding = (L - (len(Xn_normalized) % L)) % L # Evita anadir de mas si ya es multiplo
352
353 if padding > 0: # Si hay que rellenar, anadir ceros al final
354     Xn_normalized = np.append(Xn_normalized, np.zeros(padding))
355
356 # ACTUALIZAR el tamano y Ns despues del relleno
357 new_size = len(Xn_normalized)
358 Ns = new_size // L
359 sn = np.dot(np.reshape(Xn_normalized, [Ns, L]), np.c_[phi1, phi2])
360 sn1 = sn[:, 0]
361 sn2 = sn[:, 1]
362
363 # ---- Constelacion recibida ----
364 # Filtrado adaptado para la senal recibida
```

```
hr1 = np.flipud(phi1)
hr2 = np.flipud(phi2)

yn1_rx = np.convolve(hr1, rx_signal_normalized, 'valid')
yn2_rx = np.convolve(hr2, rx_signal_normalized, 'valid')

# Ajuste de indices para evitar el error de out of bounds
max_index = min(len(yn1_rx), len(yn2_rx)) # Encuentra el menor tamano valido
sampling_indices = np.arange(0, max_index, L) # Asegura que los indices sean validos

sn1_rx = yn1_rx[sampling_indices]
sn2_rx = yn2_rx[sampling_indices]

# Obtencion de limites de los ejes
xmax = max(sn1) + max(sn1) / 10 + 0.5
xmin = -(abs(min(sn1)) + abs(min(sn1)) / 10) - 0.5
ymax = max(sn2) + max(sn2) / 10 + 0.5
ymin = -(abs(min(sn2)) + abs(min(sn2)) / 10) - 0.5

Nmax = min(64, len(sn1), len(sn1_rx)) # Tomar el minimo de 64 y la cantidad de simbolos
    disponibles

# ---- Representacion de la constelacion ----
plt.figure(figsize=(6, 6))

# Pinta los ejes
plt.axhline(0, color='black', linewidth=1) # Eje X
plt.axvline(0, color='black', linewidth=1) # Eje Y

# Puntos de la constelacion transmitida
plt.plot(sn1[:Nmax], sn2[:Nmax], 'bo', label="Constelacion Transmitida")

# Puntos de la constelacion recibida
plt.plot(sn1_rx[:Nmax], sn2_rx[:Nmax], 'r*', label="Constelacion Recibida")

# Configuracion del grafico
plt.xlim(xmin, xmax)
plt.ylim(ymin, ymax)
plt.title("Constelacion Transmitida vs Recibida")
plt.xlabel("Eje I")
plt.ylabel("Eje Q")
plt.legend()
plt.grid()
plt.savefig("QAM_pequeno_constelacion.eps", format="eps", dpi=300)
plt.show()

# %% [markdown]
# ## Calculo del EVM (Error Vector Magnitude)

# %%
# ---- CALCULO DEL EVM ----
# Ajustar el tamano de los simbolos transmitidos y recibidos al minimo comun
N_symbols = min(len(sn1), len(sn1_rx))
sn1_tx_trimmed = sn1[:N_symbols]
sn1_rx_trimmed = sn1_rx[:N_symbols]

# Calculo del EVM en porcentaje
evm = np.sqrt(np.sum(np.abs(sn1_rx_trimmed - sn1_tx_trimmed) ** 2) / N_symbols) / np.max(np.abs(
    sn1_tx_trimmed)) * 100

print(f"EVM (%): {evm:.2f} %")
print(len(Xn))
print(len(rx_signal))

# %% [markdown]
# ## Calculo del BER (Bit Error Rate)

# %%
# ---- DECODIFICACION DE LOS SIMBOLOS RECIBIDOS PARA 16-QAM ----

# Asegurar longitud correcta
```

```
sn1_rx = sn1_rx[:Ns]
sn1 = sn1[:Ns]

# Definir la constelacion 16-QAM (rejilla cuadrada)
iq = np.array([-3, -1, 1, 3])
alfabeto = np.array([i + 1j * q for i in iq for q in iq]) # 16 puntos en la constelacion

# Normalizar la potencia del alfabeto (opcional pero recomendable)
alfabeto /= np.sqrt(np.mean(np.abs(alfabeto) ** 2))

# Determinar el simbolo mas cercano en la constelacion
indices_rx = np.argmin(np.abs(sn1_rx[:, None] - alfabeto), axis=1)
indices_tx = np.argmin(np.abs(sn1[:, None] - alfabeto), axis=1)

# Convertir indices a cadenas binarias
k = 4  # 16-QAM usa 4 bits por simbolo
Bn_rx = np.array([np.binary_repr(i, width=k) for i in indices_rx])
Bn_tx = np.array([np.binary_repr(i, width=k) for i in indices_tx])

# Convertir a una unica secuencia de bits
Bn_rx = np.array(list(''.join(Bn_rx)), dtype=int)
Bn_tx = np.array(list(''.join(Bn_tx)), dtype=int)

# ---- CALCULO DEL BER ----
min_len = min(len(Bn_rx), len(Bn_tx)) # Asegurar tamanos iguales
bit_errors = np.sum(Bn_rx[:min_len] != Bn_tx[:min_len]) # Contar errores de bit
BER = bit_errors / min_len # Tasa de error de bit

# Mostrar resultados
print(f"BER (Tasa de Error de Bits, 16-QAM): {BER:.30f}")


# %%
# ---- Liberar buffer SDR y apagar la transmision----
sdr.tx_enabled = False # Desactiva la transmision
sdr.tx_destroy_buffer()
```

Código A.13 Transmisión y recepción de una señal QAM paso banda (Transmisión larga).

```
# %% [markdown]
# # Proyecto: Transmision grande 16QAM con PlutoSDR
# **Autor:** Alejandro Madero Vilchez
# **Fecha:** 4 de abril de 2025
# **Grado:** Ingenieria Aeroespacial

# %% [markdown]
# ## Librerias y parametros
#

# %%
import numpy as np
import matplotlib.pyplot as plt
from adi import Pluto
from transmisorqam import transmisorqam
from scipy.signal import correlate
import scipy.io as sio
import time

# Configuracion de parametros
sample_rate = 20e6
num_samples = 4020
center_freq = 915e6
bits_por_tanda = 394
bits_totales = 10244
num_tandas = int(bits_totales / bits_por_tanda)

# Conectar con PlutoSDR
sdr = Pluto(uri="ip:192.168.2.1")
sdr.sample_rate = int(sample_rate)
```

```
31
32 # Configuracion de transmision
33 sdr.tx_rf_bandwidth = int(sample_rate)
34 sdr.tx_lo = int(center_freq)
35 sdr.tx_hardwaregain_chan0 = -10
36
37 # Configuracion de recepcion
38 sdr.rx_lo = int(center_freq)
39 sdr.rx_rf_bandwidth = int(sample_rate)
40 #sdr.rx_buffer_size = num_samples
41 sdr.rx_buffer_size = 2*num_samples
42 sdr.gain_control_mode_chan0 = 'manual'
43 sdr.rx_hardwaregain_chan0 = 10.0
44
45 # Parametros de la modulacion
46 Eb = 8 # Energia media transmitida
47 L = 40 # Numero de muestras por simbolo
48 Rb = 2 * 10**6 # Tasa de bits
49 Tb = 1 / Rb
50 M = 16 # Orden de la modulacion (16-QAM)
51 k = int(np.log2(M)) # Bits por simbolo
52 Tm = (Tb * k) / L
53 M1 = 4 # Niveles en la componente en fase
54 M2 = 4 # Niveles en la componente en cuadratura
55 fs = 1 / Tm
56 f0 = fs / L
57
58 # Forzar numero de muestras por simbolo a ser par
59 L = int(np.ceil(L / 2) * 2)
60
61 # Pulsos basicos ortogonales
62 g1 = np.r_[np.ones(int(L/2)), np.zeros(int(L/2))] # Componente en fase
63 g2 = np.r_[np.zeros(int(L/2)), np.ones(int(L/2))] # Componente en cuadratura
64
65 ## IMPRIMIR TODO
66 print("\n--- PARAMETROS DE CONFIGURACION ---")
67 print(f"Sample rate: {sample_rate}")
68 print(f"Numero de muestras: {num_samples}")
69 print(f"Frecuencia central: {center_freq/1e6} MHz")
70 print(f"Tasa de bits (Rb): {Rb} bps")
71 print(f"Duracion del bit (Tb): {Tb} s")
72 print(f"Intervalo de muestreo (Tm): {Tm} s")
73 print(f"Frecuencia de muestreo (fs): {fs} Hz")
74 print(f"Frecuencia del pulso paso banda (f0): {f0} Hz")
75 print(f"Orden de PAM (M): {M}")
76 print(f"M1: {M1}")
77 print(f"M2: {M2}")
78 print(f"Numero de bits por simbolo (k): {k}")
79 print(f"Numero de muestras por simbolo (L): {L}")
80
81 # %% [markdown]
82 # # Envio por tandas
83
84 # %%
85 # Almacenar transmisiones
86 Xn_total = []
87 rx_signal_total = np.zeros(num_samples * num_tandas, dtype=np.complex64) # Preasignar espacio
88
89 for i in range(num_tandas):
90     preambulo = np.array([1, 0, 1, 1, 0, 1, 0, 0]) # Secuencia de sincronizacion
91     Bn = np.concatenate([preambulo, np.random.randint(0, 2, bits_por_tanda)]) # Mensaje con
          preambulo
92
93     # Definicion del tiempo continuo
94     Ns = int(len(Bn) / k)
95     Nb = len(Bn) # Numero de bits transmitidos
96     Td = Tb * Nb # Duracion total de la senal
97     t = np.arange(0, Td, Tm) # Vector de tiempo
98
99     # Generacion de la senal modulada 16-QAM
100    [Xn, BnI, BnQ, AnI, AnQ, AI, AQ, phi1, phi2] = transmisorqam(Bn, Eb, M1, M2, g1, g2, L)
```

```
101
102     # Normalizar la senal para transmision con PlutoSDR
103     Xt = np.sqrt(1 / Tm) * Xn
104
105     sdr.tx_cyclic_buffer = True
106     sdr.tx(np.zeros((len(Xt))))
107     sdr.tx_destroy_buffer()
108     sdr.tx_cyclic_buffer = True
109     sdr.tx(Xt)
110     time.sleep(1)
111
112     for _ in range(5):
113         _ = sdr.rx()
114     rx_signal = sdr.rx()
115     rx_signal = rx_signal - np.mean(rx_signal) # Restar la media
116     rx_signal = rx_signal / np.max(np.abs(rx_signal))
117
118     Xn_norm = Xn / np.linalg.norm(Xn)
119     rx_signal_norm = rx_signal / np.linalg.norm(rx_signal)
120
121     # Correlacion cruzada entre la senal recibida y transmitida
122     corr = correlate(np.real(rx_signal_norm), np.real(Xn_norm), mode='full')
123     lags = np.arange(-len(Xn_norm) + 1, len(rx_signal_norm)) * Tm * 1e6
124
125     # Encontrar el maximo de la correlacion
126     max_corr_idx = np.argmax(np.abs(corr)) # indice del maximo
127     lag_max = lags[max_corr_idx] # retardo correspondiente
128
129     # Refinamiento de desfase (ajuste cuadratico)
130     if 1 < max_corr_idx < len(corr) - 1:
131         y0, y1, y2 = corr[max_corr_idx - 1], corr[max_corr_idx], corr[max_corr_idx + 1]
132         shift_adjustment = (y0 - y2) / (2 * (y0 - 2 * y1 + y2))
133         lag_max += shift_adjustment * Tm * 1e6
134
135     # Calculamos el desfase en muestras
136     sample_offset = (lag_max * 1e-6) / Tm
137     print(f"Desfase optimo refinado: {lag_max:.6f} us")
138     print(f"Indice de desfase en muestras (fraccional): {sample_offset:.2f}")
139
140
141     # Sincronizar la senal con el desfase calculado
142     rx_signal = np.roll(rx_signal, -int(np.round(sample_offset)))
143     rx_signal_sync = rx_signal[:num_samples]
144
145     # Verificacion de inversion de fase utilizando `max(xorr(x, y))`
146     # Calculamos la correlacion de la senal sincronizada
147     corr_with_inversion = correlate(np.real(rx_signal_sync), np.real(Xn_norm), mode='full')
148     corr_original = np.max(corr)
149     corr_inverted = np.max(corr_with_inversion)
150
151     # Si la correlacion con inversion es mayor, invertimos la senal
152     if corr_inverted > corr_original: # Detectamos la inversion de fase
153         print("Inversion de fase detectada, corrigiendo...")
154         rx_signal_sync = -rx_signal_sync
155
156
157     # Almacenamos la senal sincronizada
158     Xn_total = np.concatenate((Xn_total, Xn)) if len(Xn_total) > 0 else Xn
159     rx_signal_total[i * num_samples : (i + 1) * num_samples] = rx_signal_sync
160
161     # Liberar el buffer del transmisor
162     sdr.tx_destroy_buffer()
163
164 # %% [markdown]
165 # ## Calculo del valor NMSE
166
167 # %%
168 # ---- CALCULO DEL NMSE ----
169 rx_signal_sync = rx_signal_sync *(-1)
170 min_len = min(len(Xn), len(rx_signal_sync))
171 Xn_trimmed = Xn[:min_len] # Usar Xn en lugar de Xt
```

```
rx_signal_sync = np.real(rx_signal_sync) # Convertir a real
rx_signal_trimmed = rx_signal_sync[:min_len]

# Normalizar las senales
Xn_normalized = Xn_trimmed / np.linalg.norm(Xn_trimmed)
rx_signal_normalized = rx_signal_trimmed / np.linalg.norm(rx_signal_trimmed)

# Calculo del NMSE
nmse = 20 * np.log10(np.linalg.norm(rx_signal_normalized - Xn_normalized) / np.linalg.norm(
    Xn_normalized))
print(f"NMSE entre la senal transmitida y recibida (en dB): {nmse:.6f} dB")

# Verificacion de normalizacion
print(f"Norma de Xn_normalized: {np.linalg.norm(Xn_normalized)}")
print(f"Norma de rx_signal_normalized: {np.linalg.norm(rx_signal_normalized)}")
print(f"Tamano de Xn_normalized: {len(Xn_normalized)}")
print(f"Tamano de rx_signal_normalized: {len(rx_signal_normalized)}")


# %% [markdown]
# ## Graficar los 8 primeros simbolos en el tiempo

# %%
# ---- Graficar los primeros 8 simbolos en el tiempo ----
rx_signal_normalized = rx_signal_normalized * (-1)
t = np.arange(0, 8 * L) * Tm

plt.figure(figsize=(10, 4))
plt.plot(t, Xn_normalized[:8 * L], label="Senal Transmitida", linestyle='-', marker='o')
plt.plot(t, rx_signal_normalized[:8 * L], label="Senal Recibida", linestyle='--', marker='x')
plt.xlabel("Tiempo (s)")
plt.ylabel("Amplitud")
plt.title("8 Primeros Simbolos en el Tiempo")
plt.legend()
plt.grid()
plt.savefig("QAM_grande_simbolos_.eps", format="eps", dpi=300)
plt.show()

# %% [markdown]
# ## Graficar las senales normalizadas en el tiempo

# %%
# ---- REPRESENTACION DE LAS SENALES NORMALIZADAS ----
plt.figure(figsize=(12, 4))
plt.plot(np.real(Xn_normalized), label="Senal transmitida normalizada", color='b')
plt.plot(np.real(rx_signal_normalized), label="Senal recibida normalizada", color='r', linestyle='
    dashed')
plt.title("Comparacion de senales normalizadas")
plt.xlabel("Muestras")
plt.ylabel("Amplitud")
plt.legend()
plt.grid()
plt.savefig("QAM_grande_senales_normalizadas.eps", format="eps", dpi=300)
plt.show()

# %% [markdown]
# ## Constelacion Tx y Rx

# %%
# CONSTELACION DE BITS Tx Y Rx
# Obtencion de los parametros necesarios
L = len(phi1) # Numero de muestras por simbolo
Ns = int((len(BnI) + len(BnQ)) / (np.log2(len(AI) * len(AQ)))) # Numero de simbolos transmitidos

# Obtencion de las secuencias de niveles transmitidos
# Calculamos el tamano nuevo asegurando que sea multiplo de L
new_size = (len(Xn_normalized) // L) * L
padding = (L - (len(Xn_normalized) % L)) % L # Evita anadir de mas si ya es multiplo

if padding > 0: # Si hay que rellenar, anadir ceros al final
    Xn_normalized = np.append(Xn_normalized, np.zeros(padding))
```

```

# ACTUALIZAR el tamano y Ns despues del relleno
new_size = len(Xn_normalized)
Ns = new_size // L
sn = np.dot(np.reshape(Xn_normalized, [Ns, L]), np.c_[phi1, phi2])
sn1 = sn[:, 0]
sn2 = sn[:, 1]

# ---- Constelacion recibida ----
# Filtrado adaptado para la senal recibida
hr1 = np.flipud(phi1)
hr2 = np.flipud(phi2)

yn1_rx = np.convolve(hr1, rx_signal_normalized, 'valid')
yn2_rx = np.convolve(hr2, rx_signal_normalized, 'valid')

# Ajuste de indices para evitar el error de out of bounds
max_index = min(len(yn1_rx), len(yn2_rx)) # Encuentra el menor tamano valido
sampling_indices = np.arange(0, max_index, L) # Asegura que los indices sean validos

sn1_rx = yn1_rx[sampling_indices]
sn2_rx = yn2_rx[sampling_indices]

# Obtencion de limites de los ejes
xmax = max(sn1) + max(sn1) / 10 + 0.5
xmin = -(abs(min(sn1)) + abs(min(sn1)) / 10) - 0.5
ymax = max(sn2) + max(sn2) / 10 + 0.5
ymin = -(abs(min(sn2)) + abs(min(sn2)) / 10) - 0.5

Nmax = min(64, len(sn1), len(sn1_rx)) # Tomar el minimo de 64 y la cantidad de simbolos
    disponibles

# ---- Representacion de la constelacion ----
plt.figure(figsize=(6, 6))

# Pinta los ejes
plt.axhline(0, color='black', linewidth=1) # Eje X
plt.axvline(0, color='black', linewidth=1) # Eje Y

# Puntos de la constelacion transmitida
plt.plot(sn1[:Nmax], sn2[:Nmax], 'bo', label="Constelacion Transmitida")

# Puntos de la constelacion recibida
plt.plot(sn1_rx[:Nmax], sn2_rx[:Nmax], 'r*', label="Constelacion Recibida")

# Configuracion del grafico
plt.xlim(xmin, xmax)
plt.ylim(ymin, ymax)
plt.title("Constelacion Transmitida vs Recibida")
plt.xlabel("Eje I")
plt.ylabel("Eje Q")
plt.legend()
plt.grid()
plt.savefig("QAM_grande_constelacion.eps", format="eps", dpi=300)
plt.show()

# %% [markdown]
# ## Calculo del valor EVM (Error Vector Magnitude)

# %%
# ---- CALCULO DEL EVM ----
# Ajustar el tamano de los simbolos transmitidos y recibidos al minimo comun
N_symbols = min(len(sn1), len(sn1_rx))
sn1_tx_trimmed = sn1[:N_symbols]
sn1_rx_trimmed = sn1_rx[:N_symbols]

# Calculo del EVM en porcentaje
evm = np.sqrt(np.sum(np.abs(sn1_rx_trimmed - sn1_tx_trimmed) ** 2) / N_symbols) / np.max(np.abs(
    sn1_tx_trimmed)) * 100

print(f"EVM (%): {evm:.2f} %")
```

```
print(len(Xn))
print(len(rx_signal))

# %% [markdown]
# ## Calculo del BER (Bit Error Rate)

# %%
# ---- DECODIFICACION DE LOS SIMBOLOS RECIBIDOS PARA 16-QAM ----

# Asegurar longitud correcta
sn1_rx = sn1_rx[:Ns]
sn1 = sn1[:Ns]

# Definir la constelacion 16-QAM (rejilla cuadrada)
iq = np.array([-3, -1, 1, 3])
alfabeto = np.array([i + 1j * q for i in iq for q in iq]) # 16 puntos en la constelacion

# Normalizar la potencia del alfabeto (opcional pero recomendable)
alfabeto /= np.sqrt(np.mean(np.abs(alfabeto) ** 2))

# Determinar el simbolo mas cercano en la constelacion
indices_rx = np.argmin(np.abs(sn1_rx[:, None] - alfabeto), axis=1)
indices_tx = np.argmin(np.abs(sn1[:, None] - alfabeto), axis=1)

# Convertir indices a cadenas binarias
k = 4  # 16-QAM usa 4 bits por simbolo
Bn_rx = np.array([np.binary_repr(i, width=k) for i in indices_rx])
Bn_tx = np.array([np.binary_repr(i, width=k) for i in indices_tx])

# Convertir a una unica secuencia de bits
Bn_rx = np.array(list(''.join(Bn_rx)), dtype=int)
Bn_tx = np.array(list(''.join(Bn_tx)), dtype=int)

# ---- CALCULO DEL BER ----
min_len = min(len(Bn_rx), len(Bn_tx)) # Asegurar tamanos iguales
bit_errors = np.sum(Bn_rx[:min_len] != Bn_tx[:min_len]) # Contar errores de bit
BER = bit_errors / min_len # Tasa de error de bit

# Mostrar resultados
print(f"BER (Tasa de Error de Bits, 16-QAM): {BER:.30f}")

# %%
# ---- Liberar buffer SDR y apagar la transmision----
sdr.tx_enabled = False # Desactiva la transmision
sdr.tx_destroy_buffer()
```

A.6 Transmisión y recepción de un tono con dos dispositivos

Código A.14 Transmisión y recepción de un tono con dos dispositivos.

```
# %% [markdown]
# # Proyecto: Transmision de un tono con 2 PlutoSDR
# **Autor:** Alejandro Madero Vilchez
# **Fecha:** 4 de abril de 2025
# **Grado:** Ingenieria Aeroespacial

# %% [markdown]
# ## Librerias y Parametros

# %%
# Librerias necesarias
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import windows
from adi import Pluto

# Configuracion de parametros
```

```
fs = int(20e6)                                                                  # Frecuencia de muestreo (20 MHz)
tx_freq = int(868e6)                                                            # Frecuencia de transmision (500 MHz)
rx_freq = int(868e6)                                                            # Frecuencia de recepcion (500 MHz)
tone_freq = 1e6                                                                 # Tono a 1 MHz
num_samples = 4096                                                              # Numero de muestras

# Conectar con PlutoSDR
sdr_tx = Pluto(uri="ip:192.168.2.2")
sdr_rx = Pluto(uri="ip:192.168.2.1")


# Configuracion de transmision
sdr_tx.tx_lo = tx_freq                                                            # TX en 500 MHz
sdr_tx.tx_hardwaregain = 0.0                                                      # Potencia de transmision (-50 dB)
sdr_tx.tx_cyclic_buffer = True                                                    # Habilitar transmision ciclica
sdr_tx.sample_rate = fs                                                           # Configuramos fs del pluto


# Configuracion de recepcion
sdr_rx.rx_lo = rx_freq                                                            # RX en 500 MHz
sdr_rx.rx_rf_bandwidth = int(20e6)                                                # Ancho de banda de 20 MHz
sdr_rx.rx_buffer_size = num_samples                                               # Tamano del buffer
sdr_rx.rx_hardwaregain_chan0 = 25.0

# %% [markdown]
# ## Transmision del tono y recepcion

# %%
# Generar el tono I/Q
t = np.arange(num_samples) / fs
tx_signal = 2**14 * np.exp(2j * np.pi * tone_freq * t)                          # Senal I/Q

# Transmitir la senal
sdr_tx.tx(tx_signal)

# Clear buffer just to be safe
for i in range (0, 10):
    raw_data = sdr_rx.rx()# Receive samples
    rx_samples = sdr_rx.rx()
# Capturar datos
rx_signal = sdr_rx.rx()


# Aplicar ventana Hann para mejorar la FFT
#window = windows.hann(len(rx_signal ))
#samples_windowed = rx_signal * window
samples_windowed = rx_signal
# Calcular FFT y corregir frecuencia
X_f = np.fft.fft(samples_windowed)
X_f = np.fft.fftshift(np.abs(X_f) / len(rx_signal ))                            # Normalizacion correcta

# Ajustar escala de frecuencia correctamente
freqs_mhz = np.fft.fftshift(np.fft.fftfreq(len(rx_signal ), d=1/fs)) * (1e-6) + (rx_freq * 1e-6)
    # Ahora centrado correctamente

# %% [markdown]
# ## Graficar senales en el tiempo
```

```
74
75 # %%
76 # Graficar espectro de frecuencia
77 plt.figure(figsize=(12, 4))
78 plt.plot(t*1e6, np.real(tx_signal))                                    # Eje X
       ahora en Hz
79 plt.title("Senal transmitida")
80 plt.xlabel("Tiempo (us)")
81 plt.ylabel("Amplitud")
82 plt.grid()
83 plt.ticklabel_format(useOffset=False, style='plain')
84 plt.savefig("seno_2plutos_senaltx.eps", format="eps", dpi=300)
85
86 # Graficar espectro de frecuencia
87 plt.figure(figsize=(12, 4))
88 plt.plot(t*1e6, np.real(rx_signal))                                    # Eje X
       ahora en Hz
89 plt.title("Senal recibida")
90 plt.xlabel("Tiempo (us)")
91 plt.ylabel("Amplitud")
92 plt.grid()
93 plt.ticklabel_format(useOffset=False, style='plain')
94 plt.savefig("seno_2plutos_senalrx.eps", format="eps", dpi=300)
95
96
97 plt.show()
98
99 # %% [markdown]
100 # ## Graficar espectro de la senal recibida
101
102 # %%
103 # Graficar espectro de frecuencia
104 plt.figure(figsize=(12, 4))
105 plt.plot(freqs_mhz, X_f) # Eje X ahora en Hz
106 plt.title("Espectro de Frecuencia")
107 plt.xlabel("Frecuencia (MHz)")
108 plt.ylabel("Amplitud")
109 plt.grid()
110 plt.ticklabel_format(useOffset=False, style='plain')                   # Forzar
       escala correcta
111 plt.xlim(freqs_mhz.min(), freqs_mhz.max())
112 plt.savefig("seno_2plutos_espectro.eps", format="eps", dpi=300)
113 plt.show()
114
115 # %%
116 sdr_tx.tx_destroy_buffer()
```

A.7 Transmisión y recepción de una señal PAM paso banda con dos dispositivos

Código A.15 Transmisión y recepción de una señal PAM paso banda con dos dispositivos.

```
1 # Proyecto: Recepcion grande 8PAM con 2 PlutoSDR
2 # Autor: Alejandro Madero Vilchez
3 # Fecha: 5 de abril de 2025
4 # Grado: Ingenieria Aeroespacial
5
6 # Librerias y parametros
7
8 import numpy as np
9 import matplotlib.pyplot as plt
10 from adi import Pluto
11 from transmisorpam import transmisorpam
12 from scipy.fft import fft, fftfreq
13 from scipy.signal import correlate
14 from scipy.signal import hilbert
15
16 # Configuracion de parametros
```

```
sample_rate = 5e6
num_samples = 4020
center_freq = 868e6
bits_por_tanda = 394
bits_totales = 394
num_tandas = int(bits_totales / bits_por_tanda)

# === Conectar con los dos PlutoSDR ===
# Reemplaza con las IPs correctas de tus dispositivos
sdr_tx = Pluto(uri="ip:192.168.2.2") # Transmisor
sdr_rx = Pluto(uri="ip:192.168.2.1") # Receptor

# Configuracion de transmision
sdr_tx.sample_rate = int(sample_rate)
sdr_tx.tx_rf_bandwidth = int(sample_rate)
sdr_tx.tx_lo = int(center_freq)
sdr_tx.tx_hardwaregain_chan0 = -10.0

# Configuracion de recepcion
sdr_rx.sample_rate = int(sample_rate)
sdr_rx.rx_lo = int(center_freq)
sdr_rx.rx_rf_bandwidth = int(sample_rate)
sdr_rx.rx_buffer_size = num_samples
sdr_rx.gain_control_mode_chan0 = 'manual'
sdr_rx.rx_hardwaregain_chan0 = 0.0

# Parametros de modulacion
Eb = 8
M = 8
k = int(np.log2(M))
L = 30
Rb = 2e6
Tb = 1 / Rb
Tm = (Tb * k) / L
fs = 1 / Tm
f0 = fs / L
L = int(np.ceil(L / 2) * 2)
g = np.cos(2 * np.pi * f0 * np.arange(L) * Tm)


## IMPRIMIR TODO
print("\n--- PARAMETROS DE CONFIGURACION ---")
print(f"Sample rate: {sample_rate}")
print(f"Numero de muestras: {num_samples}")
print(f"Frecuencia central: {center_freq/1e6} MHz")
print(f"Tasa de bits (Rb): {Rb} bps")
print(f"Duracion del bit (Tb): {Tb} s")
print(f"Intervalo de muestreo (Tm): {Tm} s")
print(f"Frecuencia de muestreo (fs): {fs} Hz")
print(f"Frecuencia del pulso paso banda (f0): {f0} Hz")
print(f"Orden de PAM (M): {M}")
print(f"Numero de bits por simbolo (k): {k}")
print(f"Numero de muestras por simbolo (L): {L}")

# Recepcion por tandas

# Almacenar transmisiones
Xn_total = []
rx_signal_total = np.zeros(num_samples * num_tandas, dtype=np.complex64)

for i in range(num_tandas):
    preambulo = np.array([1, 0, 1, 1, 0, 1, 0, 0])
    Bn = np.concatenate([preambulo, np.random.randint(0, 2, bits_por_tanda)])
    [Xn, Bn, An, phi, alfabeto] = transmisorpam(Bn, Eb, M, g, L)
    Xt = np.sqrt(1 / Tm) * Xn

    # Transmitir
    sdr_tx.tx_cyclic_buffer = True
    sdr_tx.tx(Xt)

    # Recibir
```

```
    rx_signal = sdr_rx.rx()
    rx_signal = rx_signal / np.max(np.abs(rx_signal))

    # ---- Estimacion del retardo mediante FFT ----
    Xf = fft(Xn)
    Xf_rx = fft(rx_signal)
    freqs = fftfreq(len(Xn), d=Tm)
    phase_diff = np.angle(Xf_rx) - np.angle(Xf)
    valid_idx = np.where((freqs > 0) & (freqs < fs / 2))
    p = np.polyfit(freqs[valid_idx], phase_diff[valid_idx], 1)
    estimated_a = -p[0] / (2 * np.pi)

    sample_offset = int(round(estimated_a))
    rx_signal_sync = np.roll(rx_signal, -sample_offset)

    # ---- Calculo refinado del desfase ----
    Xn_norm = Xn / np.linalg.norm(Xn)
    rx_signal_norm = rx_signal / np.linalg.norm(rx_signal)

    corr = correlate(np.real(rx_signal_norm), np.real(Xn_norm), mode='full')
    lags = np.arange(-len(Xn_norm) + 1, len(rx_signal_norm)) * Tm * 1e6

    max_corr_idx = np.argmax(np.abs(corr))
    lag_max = lags[max_corr_idx]

    if 1 < max_corr_idx < len(corr) - 1:
        y0, y1, y2 = corr[max_corr_idx - 1], corr[max_corr_idx], corr[max_corr_idx + 1]
        shift_adjustment = (y0 - y2) / (2 * (y0 - 2 * y1 + y2))
        lag_max += shift_adjustment * Tm * 1e6

    sample_offset = int(np.round((lag_max * 1e-6) / Tm))
    print(f"Desfase optimo refinado: {lag_max:.6f} us")
    print(f"Indice de desfase en muestras: {sample_offset}")

    rx_signal_sync = np.roll(rx_signal, -sample_offset)

    phase_tx = np.angle(np.fft.fft(Xn))
    phase_rx = np.angle(np.fft.fft(rx_signal_sync))
    phase_diff = np.mean(phase_rx - phase_tx)

    if np.abs(phase_diff - np.pi) < 0.1:
        rx_signal_sync *= -1

    Xn_total = np.concatenate((Xn_total, Xn)) if len(Xn_total) > 0 else Xn
    rx_signal_total[i * num_samples : (i + 1) * num_samples] = rx_signal_sync

    sdr_tx.tx_destroy_buffer()


# correlacion

# === Graficar correlacion ===
plt.figure(figsize=(10, 4))
plt.plot(np.abs(corr))
plt.title("Correlacion con preambulo")
plt.xlabel("Muestras")
plt.ylabel("Magnitud")
plt.grid()
plt.show()

# Graficar senales Tx y Rx

# Ajustar longitud para comparacion
min_len = min(len(Xn_total), len(rx_signal_total))
tx_signal_time = np.real(Xn_total[:min_len])
rx_signal_time = np.real(rx_signal_total[:min_len])

# Normalizar ambas senales
tx_signal_norm = tx_signal_time / np.linalg.norm(tx_signal_time)
rx_signal_norm = rx_signal_time / np.linalg.norm(rx_signal_time)

```

```
# Graficar
plt.figure(figsize=(14, 5))
plt.plot(tx_signal_norm, label="Tx Normalizada", alpha=0.7)
plt.plot(rx_signal_norm, label="Rx Normalizada", alpha=0.7)
plt.title("Comparacion de senales transmitida y recibida normalizadas")
plt.xlabel("Muestras")
plt.ylabel("Amplitud normalizada")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig("PAM_2plutos_senales.eps", format="eps", dpi=300)
plt.show()

# Comparacion de las envolventes de ambas senales

# Envolvente mediante transformada de Hilbert
env_tx = np.abs(hilbert(tx_signal_norm))
env_rx = np.abs(hilbert(rx_signal_norm))

plt.figure(figsize=(14, 5))
plt.plot(env_tx, label="Envolvente Tx", alpha=0.8)
plt.plot(env_rx, label="Envolvente Rx", alpha=0.8)
plt.title("Comparacion de envolventes (Tx vs Rx)")
plt.xlabel("Muestras")
plt.ylabel("Amplitud")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig("PAM_2plutos_envolventes.eps", format="eps", dpi=300)
plt.show()

## Constelacion Tx y Rx
# Filtrado y sincronizacion
Xn_normalized = Xn_total / np.linalg.norm(Xn_total)
rx_signal_normalized = rx_signal_total / np.linalg.norm(rx_signal_total)
hr1 = np.flipud(phi)
yn1 = np.convolve(hr1, Xn_normalized, 'valid')
yn1_rx = np.convolve(hr1, rx_signal_normalized, 'valid')

Ns = int(len(Bn) / k) # Numero de simbolos transmitidos
sn1 = yn1[np.arange(0, Ns * L, L)]
sn1_rx = yn1_rx[np.arange(0, Ns * L, L)]

# Definir los limites de la constelacion
xmax = max(sn1) + max(sn1) / 10 + 0.5
xmin = -(abs(min(sn1)) + abs(min(sn1)) / 10) - 0.5

Nmax = min(64, len(sn1), len(sn1_rx))

# ---- Graficar la constelacion ----
plt.figure(figsize=(6, 6))
plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.plot(sn1[:Nmax], np.zeros_like(sn1[:Nmax]), 'bo', label="Constelacion Transmitida")
plt.plot(sn1_rx[:Nmax], np.zeros_like(sn1_rx[:Nmax]), 'r*', label="Constelacion Recibida")
plt.xlim(xmin, xmax)
plt.xlim(-0.15, 0.15)
plt.ylim(-0.5, 0.5)
plt.title("Constelacion Transmitida vs Recibida")
plt.xlabel("Amplitud")
plt.legend()
plt.grid()
#plt.savefig("PAM_grande_constelacion.eps", format="eps", dpi=300)
plt.show()

# ---- Liberar buffer SDR ----
sdr_tx.tx_destroy_buffer()
```

A.8 Detección de actividad WiFi

Código A.16 Conmutación de señal WiFi.

```
// Proyecto: Conmutación de señal WiFi
/ **Autor:** Alejandro Madero Vílchez
/ **Fecha:** 3 de mayo de 2025
/ **Grado:** Ingeniería Aeroespacial


#include <ESP8266WiFi.h> // Incluir la librería para trabajar con WiFi en el
    ESP8266

const char *ssid = "Test_AP"; // Nombre de la red WiFi que se va a crear
const char *password = "12345678"; // Contraseña de la red WiFi
unsigned long previousMillis = 0; // Variable para almacenar el tiempo anterior
const long interval = 5000; // Intervalo de 5 segundos (5000 milisegundos)
bool wifiOn = false; // Variable booleana para llevar el control del estado del
     WiFi (encendido o apagado)

void setup() {
  Serial.begin(115200); // Inicia la comunicación serie a 115200 baudios para
      el monitoreo
  WiFi.softAP(ssid, password, 1); // Crea un punto de acceso WiFi con el nombre
       'Test_AP' y la contraseña '12345678' en el canal 1
  Serial.println("WiFi AP encendido en el canal 1"); // Imprime en el monitor
      serie que el AP WiFi está encendido
}

void loop() {
  unsigned long currentMillis = millis(); // Obtiene el tiempo actual en
      milisegundos desde que arrancó el microcontrolador

  // Compara si han pasado más de 5 segundos desde la última vez que se ejecutó
       este bloque
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis; // Actualiza el tiempo anterior para el
        siguiente ciclo de comparación

    // Si el WiFi está encendido
    if (wifiOn) {
      WiFi.softAPdisconnect(true); // Apaga el punto de acceso WiFi
      Serial.println("WiFi AP apagado"); // Imprime en el monitor serie que el
          AP ha sido apagado
      wifiOn = false; // Actualiza el estado del WiFi a apagado
    } else {
      WiFi.softAP(ssid, password, 1); // Vuelve a encender el AP WiFi con el
          nombre 'Test_AP' y la contraseña '12345678' en el canal 1
      Serial.println("WiFi AP encendido"); // Imprime en el monitor serie que
          el AP ha sido encendido
      wifiOn = true; // Actualiza el estado del WiFi a encendido
    }
  }
}
```

Código A.17 Detección de actividad WiFi.

```
## Proyecto: Deteccion de actividad WiFi
# **Autor:** Alejandro Madero Vílchez
# **Fecha:** 3 de mayo de 2025
# **Grado:** Ingeniería Aeroespacial


# Librerías necesarias
import numpy as np
import matplotlib.pyplot as plt
from adi import Pluto

# Conexión con PlutoSDR
pluto = Pluto("ip:192.168.2.1")        # Conectividad con el Pluto

# Configuración del receptor
pluto.rx_enabled_channels = [0]
pluto.sample_rate = 20_000_000         # 20 MSPS para capturar 20 MHz
pluto.rx_lo = 2_412_000_000            # Frecuencia central WiFi canal 1 (2.412 GHz)
pluto.rx_buffer_size = 16384           # Ajustarlo según la memoria disponible
pluto.rx_hardwaregain_chan0 = 0.0      # Ganancia máxima

# Preparar eje de frecuencias
freqs = np.fft.fftshift(np.fft.fftfreq(pluto.rx_buffer_size, 1/pluto.sample_rate)) / 1e6
freqs += pluto.rx_lo / 1e6             # Convertir a MHz

# Número de promedios
N = 10
avg_fft = np.zeros(pluto.rx_buffer_size)

# Promediar FFTs con ventana Blackman
for i in range(N):
    samples = pluto.rx()
    window = np.blackman(len(samples))
    samples_windowed = samples * window
    fft = 20 * np.log10(np.abs(np.fft.fftshift(np.fft.fft(samples_windowed))) + 1e-10) # Evitar
        log(0)
    avg_fft += fft

avg_fft /= N

# Graficar
plt.figure(figsize=(10, 5))
plt.plot(freqs, avg_fft)
plt.title("Espectro promedio (Canal 1 WiFi, 20 MHz)")
plt.xlabel("Frecuencia (MHz)")
plt.ylabel("Potencia (dB)")
plt.grid(True)
plt.ylim(40, 100)
plt.xlim(2402, 2422)
plt.tight_layout()
#plt.savefig("si_pilla_wifi1.eps", format="eps", dpi=300)
plt.show()
```

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] Couch, L. W. *Digital and Analog Communication Systems*.

[2] Hayes, M. H. *Statistical Digital Signal Processing and Modeling*. New York: John Wiley & Sons, 1996.

[3] Lyons, R. G. *Understanding Digital Signal Processing*.

[4] Oppenheim, A. V., & Willsky, A. S. *Signals and Systems*.

[5] Proakis, J. G. *Digital Communications*.

[6] Stoica, P., & Moses, R. *Spectral Analysis of Signals*. Upper Saddle River, NJ: Prentice Hall, 2005.

[7] Stewart, R. W., Crockett, L. H., Barlee, K. W., & Atkinson, D. S. W. *Software Defined Radio using MATLAB/Simulink and the RTL-SDR*. Estados Unidos: Strathclyde Academic Media, 2015.

[8] Sowjanya, P., & Satyanarayana, P. (2018). *Implementation of Transceiver Module for SDR System Using ADALM-PLUTO Platform*. Disponible en: *https://www.researchgate.net/publication/332140400*

[9] Post, J. E., & Silage, D. (2018). *Incorporating PlutoSDR in the Communication Laboratory and Classroom: Potential or Pitfall?* Disponible en: *https://www.google.com/search?q=Incorporating+PlutoSDR+in+the+Communication+Laboratory+and+Classroom%3A+Potential+or+Pitfall%3F*

[10] GlossaLAB. *Modulación por Amplitud de Pulsos (PAM)*. Disponible en: *https://www.glossalab.org/wiki/ModulaciÃşn_por_Amplitud_de_Pulsos_(PAM)*

[11] Analog Devices. *ADALM-PLUTO Wiki*. Disponible en: *https://wiki.analog.com/university/tools/pluto*

[12] Analog Devices. *ADALM-PLUTO Product Highlight*. Disponible en: *https://www.digikey.com/en/product-highlight/a/analog-devices/adalm-pluto*

[13] Mouser Electronics. *ADALM-PLUTO Overview*. Disponible en: *https://www.mouser.es/new/analog-devices/adi-adalm-pluto/*

[14] ResearchGate. *Block diagram representative of the internal architecture of the ADALM-PLUTO device*. Disponible en: *https://www.researchgate.net/figure/Block-diagram-representative-of-the-internal-architecture-of-the-Adalm-PLUTO-device_fig5_363510789*

[15] Unión Internacional de Telecomunicaciones (UIT). *Regulaciones de Radio – Art. 1*. Disponible en: *https://www.itu.int/en/ITU-R/terrestrial/broadcast/plans/Pages/plans.aspx*

[16] UIT. *Recomendación ITU-R V.431-8*.

[17] Douglas W. Witte y Jason M. Witte. *RFSoC and Software-Defined Radio: A Learning Guide for Students and Engineers*. Witte Books, 2021. Disponible en: *https://www.rfsocbook.com*

[18] Comisión Europea. *Decisión de Ejecución (UE) 2018/1538*.

[19] Ministerio de Asuntos Económicos y Transformación Digital. *Cuadro Nacional de Atribución de Frecuencias (CNAF)*. Disponible en: *https://avance.digital.gob.es/espectro/Paginas/cnaf.aspx*

[20] Atelan. *Cuadro Nacional de Atribución de Frecuencias (CUNABAF)*. Disponible en: *https://www.atelan.org/cuadro-nacional-atribucion-frecuencia/cunabaf/*

[21] Wikipedia. *United States Frequency Allocations Chart 2011 - The Radio Spectrum*. Disponible en: *https://en.wikipedia.org/wiki/File:United_States_Frequency_Allocations_Chart_2011_-_The_Radio_Spectrum.pdf*

[22] José Carlos Aradillas Jaramillo, F. Javier Payán Somet, Juan José Murillo Fuentes. *Manual de laboratorio de Comunicaciones Digitales: Python*. Universidad de Sevilla, 2019.