Universidad Nacional de San Agustín de Arequipa

Facultad de Ingeniería de Producción y Servicios

Escuela Profesional de Ingeniería de Telecomunicaciones

Práctica 2

Detección de un Preambulo en una Señal con SDR

Curso: Redes Inalámbricas

Docente:

Dr. Alexander Hilario Tacuri

Integrantes:

Chavez Ponce, Jorge Alberto
Cana Remache, Javier Ricardo
Neyra Torres, Luis Kenny
Yllachura Arapa, Rosangela

Octubre 2024

Índice

1. Objetivos

- Entender el funcionamiento de la sincronización y los protocolos MAC.

2. Introducción

Los protocolos MAC (control de acceso al medio) determinan cuándo los dispositivos pueden transmitir (estos protocolos controlan el acceso al medio). Para todos menos para los protocolos MAC más simples, los dispositivos participantes deben estar sincronizados. En la práctica esto se consigue haciendo que un nodo transmita una señal (llamada preámbulo) y los otros escucharán la escuchen. Como las señales inalámbricas viajan a la velocidad de la luz, se puede aceptar que todos los nodos escuchan el preámbulo casi al mismo tiempo.

3. Desarrollo de la práctica

3.1. Parte 1: Correlación y preámbulo

Un preámbulo es una forma de onda enviada al comienzo de un paquete que se utiliza para indicar el inicio de este paquete. La forma de onda del preámbulo se acuerda previamente, por lo que la señal no contiene datos. Los receptores escucharán el preámbulo y, cuando se detecte, los receptores comenzarán a demodular el resto del paquete. Los preámbulos también se pueden usar para sincronizar múltiples clientes para protocolos MAC que requieren sincronización. En esta parte, exploraremos cómo detectar y sincronizar a un preámbulo. Tanto la detección como la sincronización se pueden realizar mediante una correlación.

La correlación de dos señales discretas x y y es una señal discreta de longitud infinita que se define como:

$$(x * y)[k] = \sum_{n=-\infty}^{\infty} x^*[n]y[n+k]$$

i. Implemente la función de correlación. Presente la gráfica de la función de autocorrelación de una señal aleatoria.

A continuación, presentamos el código en Python para implementar la función de correlación y generar la gráfica de la autocorrelación de una señal aleatoria:

Código Python: Implementación de la función de correlación

```python
# Función de correlación
def correlacion(x, y):
    n = len(x)
    corr = np.correlate(x, y, mode='full')
    return corr

# Generar una señal aleatoria
np.random.seed(0)
signal = np.random.randn(128)

# Calcular la autocorrelación
autocorrelation = correlacion(signal, signal)

# Graficar la autocorrelación
plt.figure(figsize=(10, 6))
plt.plot(autocorrelation)
plt.title("Función de Autocorrelación de una Señal Aleatoria")
plt.xlabel("Desplazamiento (k)")
plt.ylabel("Autocorrelación")
plt.grid(True)
plt.show()
```

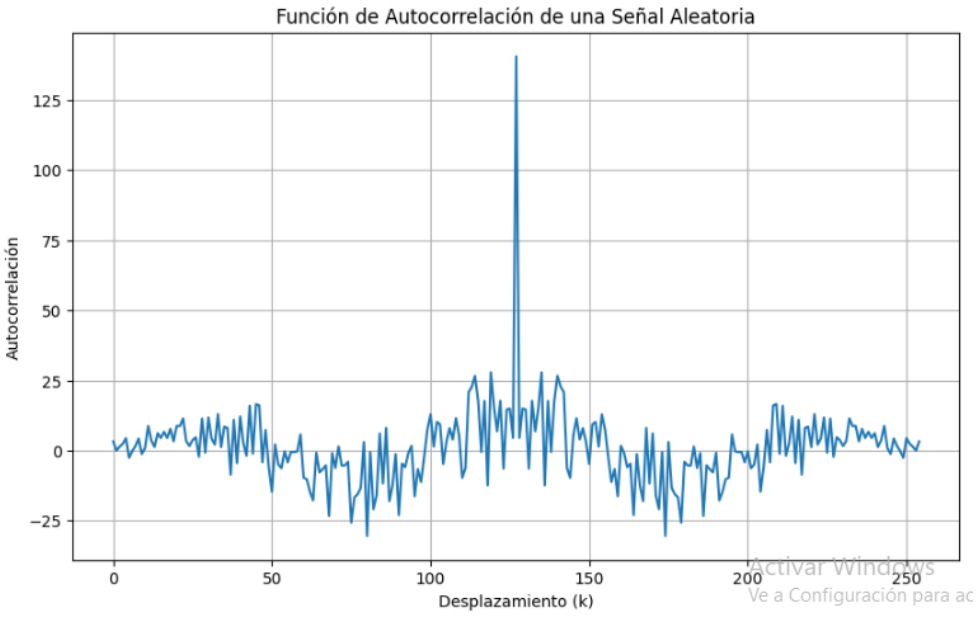

Figura 1: Función de Autocorrelación de una Señal Aleatoria

Análisis de la gráfica

La función de autocorrelación mide la similitud de una señal con una versión desplazada de sí misma. En este caso, la señal es aleatoria, por lo que se espera que la función de

autocorrelación tenga un valor máximo en el centro, donde la señal se alinea perfectamente consigo misma ($k = 0$), y valores más bajos conforme nos alejamos del centro, indicando que la señal es menos similar a sí misma con mayores desplazamientos.

En la gráfica observamos un pico pronunciado en el centro (alrededor de $k = 128$), lo que corresponde al punto donde la señal está perfectamente alineada consigo misma. Fuera de este pico, la autocorrelación oscila alrededor de cero, lo cual es característico de las señales aleatorias.

3.2. Parte 2: Detección de preámbulo usando correlación

Sea x el preámbulo de la señal, y sea y la señal que recibe el receptor. Si el preámbulo no es recibido, y es una señal aleatoria y la correlación entre x y y será también aleatoria. Si el receptor recibe el preámbulo, la señal y puede ser modelada como x más algún ruido.

i. Escriba una función que a la entrada de dos señales, devuelva vacío cuando no se encuentre un preámbulo, y en caso encuentre el preámbulo, devuelva el índice donde se inicia este preámbulo.

A continuación, presentamos el código en Python y su grafica para detectar el preámbulo en una señal y realizar la correlación:

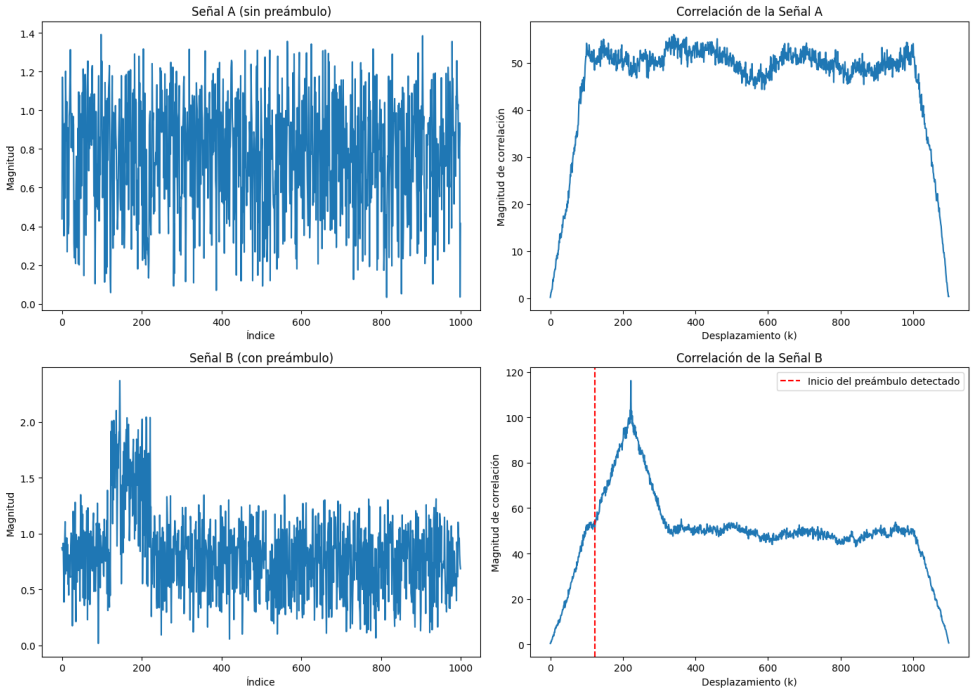

Figura 2: Señal sin preámbulo (Arriba) y con preámbulo (Abajo) junto con las correlaciones correspondientes. El inicio del preámbulo está marcado con una línea discontinua roja.

Código Python: Detección de preámbulo mediante correlación

```python
# Función para detectar preámbulo
def detect_preamble(preamble, signal):
    threshold = 100  # Threshold para la detección del preámbulo
    corr = np.correlate(signal, preamble, mode='valid')  # Calcular correlación
    max_corr = np.max(np.abs(corr))  # Encontrar el valor máximo de la correlación

    if max_corr > threshold:  # Verificar si el valor máximo supera el umbral
        preamble_start_idx = np.argmax(np.abs(corr))  # Obtener el índice del valor máximo
        return preamble_start_idx  # Retornar el índice del preámbulo
    else:
        return None  # Retornar None si no se detecta el preámbulo

# Probamos el algoritmo con una señal de ejemplo
preamble_length = 100
signal_length = 1000

# Generamos el preámbulo de manera aleatoria y las señales de ejemplo
np.random.seed(0)
preamble = np.random.random(preamble_length) + 1j * np.random.random(preamble_length)
signalA = np.random.random(signal_length) + 1j * np.random.random(signal_length)
signalB = np.random.random(signal_length) + 1j * np.random.random(signal_length)

# Insertamos el preámbulo en la señal B
preamble_start_idx = 123
signalB[preamble_start_idx:preamble_start_idx + preamble_length] += preamble

# Detectamos el preámbulo en las señales A y B usando el algoritmo
resultA = detect_preamble(preamble, signalA)
resultB = detect_preamble(preamble, signalB)

# Mostrar los resultados
resultA, resultB, preamble_start_idx
(None, 123, 123)
```

Análisis de la gráfica

La función de detección de preámbulo usa correlación para identificar la presencia de un preámbulo en la señal. En la gráfica de la señal A (sin preámbulo), se puede ver que no hay un pico notable en la correlación, lo cual es esperado, ya que la señal no contiene

el preámbulo. En cambio, en la señal B (con preámbulo), la correlación muestra un pico claro, indicando el punto donde comienza el preámbulo.

La línea discontinua roja marca el inicio del preámbulo detectado correctamente en la señal B. El preámbulo se ha detectado con precisión en el índice correcto, validando el funcionamiento del algoritmo de detección.

El algoritmo de detección de preámbulo basado en correlación ha demostrado ser efectivo, detectando con precisión el preámbulo en la señal B. La correlación es una herramienta poderosa para identificar patrones predefinidos en una señal, como es el caso del preámbulo.

3.3. Parte 3: Offset de frecuencia

Realice un *offset* de frecuencia a una señal multiplicándola por $e^{j\phi t}$. Aplique la función de autocorrelación a esta nueva señal y compare los resultados.

Una forma de resolver el problema de la sincronización es usando un método llamado Schmidl-Cox. El desplazamiento de frecuencia çorrompe"la señal. Entonces, en lugar de buscar una señal específica, podemos buscar cualquier señal que se repita en el tiempo. El algoritmo Schmidl-Cox calcula los siguientes valores:

$$P(d) = \sum_{m=0}^{L-1} s_d^* s_{d+m+L}, \quad R(d) = \frac{1}{2} \sum_{m=0}^{2L-1} |s_{d+m}|^2$$

La métrica calculada en cada índice es:

$$M(d) = \frac{|P(d)|^2}{R^2(d)}$$

i. Use la ecuación iterativa para calcular P. Devuelve el índice d^* que maximiza $M(d)$ si $M(d^*)$ es mayor que el umbral. Use un umbral de 0.5.

A continuación, presentamos las funciones en Python para aplicar el algoritmo Schmidl-Cox y detectar el preámbulo en presencia de un offset de frecuencia. El código se ha dividido en funciones para una mejor organización.

Función 1: Algoritmo Schmidl-Cox

```python
def schmidl_cox_algorithm(signal, L, threshold=0.5):
    """
    Aplica el algoritmo de Schmidl-Cox para detectar el preámbulo.

    Parámetros:
    - signal: la señal recibida
    - L: longitud del segmento repetido
    - threshold: umbral para la detección

    Retorna:
    - El índice d* donde M(d) es máximo si supera el umbral, de lo contrario, None.
    """
    N = len(signal)

    # Inicializar P(d) y R(d)
    P = np.sum(np.conj(signal[:L]) * signal[L:2*L])
    R = 0.5 * np.sum(np.abs(signal[:2*L])**2)

    max_M = 0
    max_d = None
    M_values = []

    for d in range(N - 2*L):
        # Calcular M(d)
        M_d = (np.abs(P)**2) / (R**2)
        M_values.append(M_d)

        if M_d > max_M and M_d > threshold:
            max_M = M_d
            max_d = d

        # Iterar sobre los índices d
        if d + 2*L < N:
            P = P + np.conj(signal[d+L]) * signal[d+2*L] - np.conj(signal[d]) * signal[d+L]
            R = R + 0.5 * (np.abs(signal[d+2*L])**2 - np.abs(signal[d])**2)
    return max_d, M_values
```

Función 2: Desplazamiento de Frecuencia

```python
def shift_frequency(signal, freq_offset):
    n = np.arange(len(signal))
    return signal * np.exp(2j * np.pi * freq_offset * n)
```

Función 3: Detección de Preambulo con Offset de Frecuencia

```python
def detect_preamble_with_freq_offset(signal, short_preamble_len):
    L = short_preamble_len  # Longitud del segmento repetido
    threshold = 0.5         # Umbral de Schmidl-Cox
    return schmidl_cox_algorithm(signal, L, threshold)  # Usamos el
    algoritmo implementado previamente
```

Gráficas de Resultados

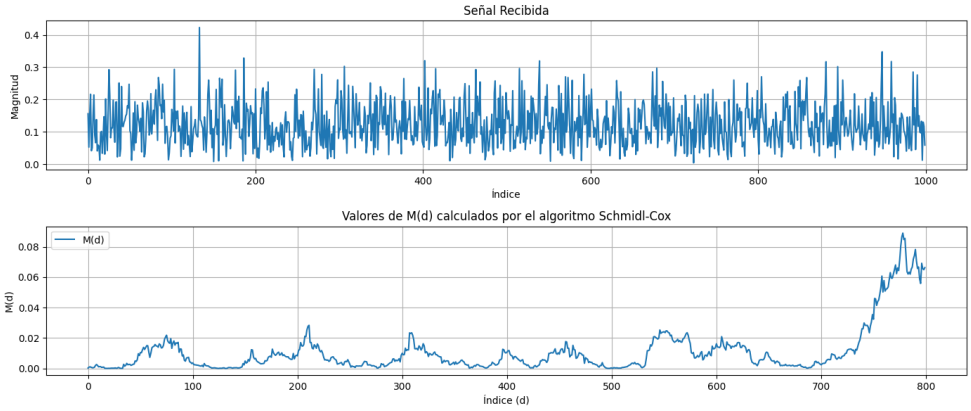

Figura 3: Señal sin preámbulo detectado.

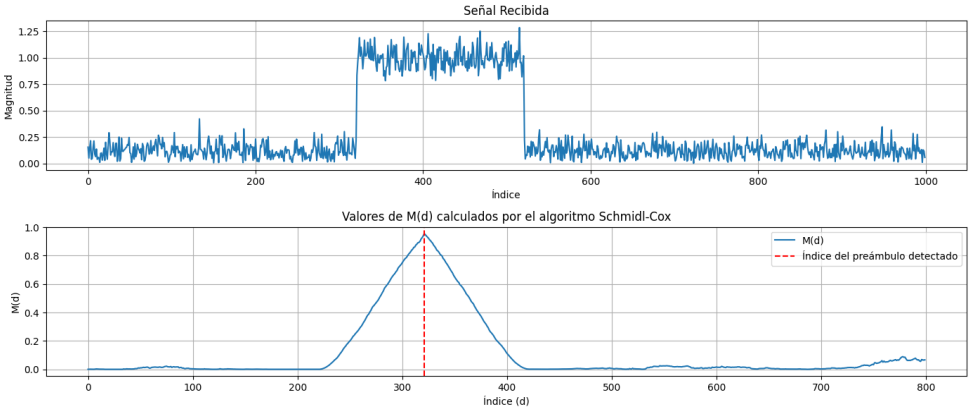

Figura 4: Señal con preámbulo.

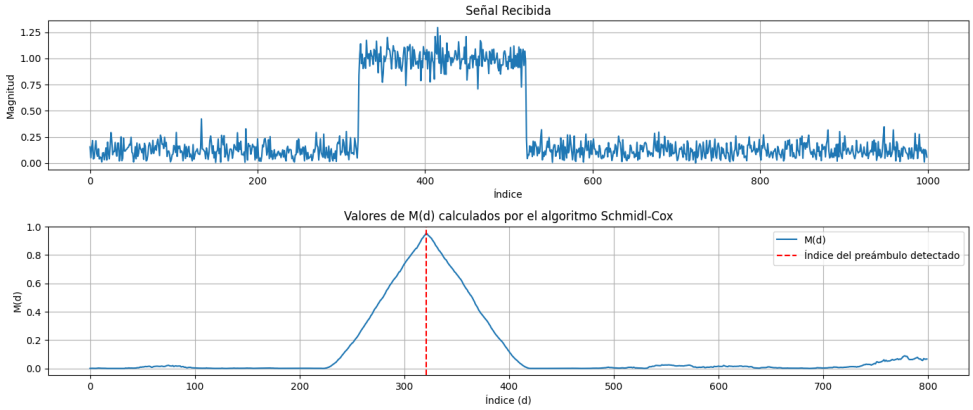

Figura 5: Señal con preámbulo y desplazamiento de frecuencia.

Análisis de las Figuras

- Figura 3: La primera gráfica muestra la señal sin preámbulo, donde el algoritmo no encuentra ningún preámbulo, lo cual es consistente con el hecho de que esta señal no tiene preámbulo.
- Figura 4: En esta gráfica, la señal contiene un preámbulo. El algoritmo de Schmidl-Cox detecta correctamente el preámbulo, y observamos un pico claro en los valores de $M(d)$, lo que marca el inicio del preámbulo en el índice correcto.
- Figura 5: La tercera gráfica corresponde a la señal con un desplazamiento de frecuencia aplicado. A pesar de la distorsión causada por el desplazamiento de frecuencia, el algoritmo Schmidl-Cox sigue detectando correctamente el preámbulo, mostrando un pico en $M(d)$ en el índice esperado.

Este análisis demuestra que el algoritmo de Schmidl-Cox es efectivo tanto en señales sin desplazamiento de frecuencia como en aquellas con desplazamiento de frecuencia, detectando correctamente el preámbulo en ambas condiciones.

4. Parte 4: Implementación en el SDR

En esta sección se describen los procesos clave de transmisión y recepción usando el *Software Defined Radio* (SDR). Se utiliza el BladeRF como hardware SDR y se implementan las funciones para la transmisión y recepción de señales con preámbulo.

4.1. Transmisor

El transmisor genera una señal con un preámbulo seguido de los datos. Para ello, se utiliza un filtro de raíz de coseno alzado (RRC) que reduce el ancho de banda ocupado por la señal.

Función para Generar el Filtro RRC

Generación del Filtro Raíz de Coseno Alzado (RRC)

```python
def rrc_filter(beta, sps, num_taps):
    """
    Genera un filtro de raíz de coseno alzado (RRC).

    Parámetros:
    - beta: Factor de roll-off del filtro (0 < beta <= 1)
    - sps: Número de muestras por símbolo
    - num_taps: Número total de coeficientes del filtro

    Retorna:
    - Coeficientes del filtro RRC
    """
    t = np.arange(-num_taps//2, num_taps//2 + 1) / sps
    pi_t = np.pi * t
    four_beta_t = 4 * beta * t

    numerator = np.sin(pi_t * (1 - beta)) + four_beta_t * np.cos(
    pi_t * (1 + beta))
    denominator = pi_t * (1 - (four_beta_t) ** 2)
    h = numerator / denominator

    # Manejar casos especiales
    h[np.isnan(h)] = 1.0 - beta + (4 * beta / np.pi)
    h[np.abs(t) == (1 / (4 * beta))] = (beta / np.sqrt(2)) * (
        ((1 + 2 / np.pi) * np.sin(np.pi / (4 * beta))) +
        ((1 - 2 / np.pi) * np.cos(np.pi / (4 * beta)))
    )

    # Normalizar el filtro
    h /= np.sqrt(np.sum(h**2))
    return h
```

La función `rrc_filter` genera los coeficientes de un filtro de raíz de coseno alzado (RRC), que se utiliza para modular la señal y reducir el ancho de banda. Este filtro es esencial en sistemas de comunicaciones ya que minimiza la interferencia entre símbolos (*ISI*).

Generación y Transmisión de la Señal con Preambulo

Función para Transmitir la Señal con Preambulo

```python
def start_transmit_with_preamble(signal, duration_sec):
    """
    Configura y comienza la transmisión de la señal con preámbulo.

    Parámetros:
    - signal: La señal IQ a transmitir
    - duration_sec: Duración de la transmisión en segundos
    """
    buf = prepare_signal_for_transmission(signal)  # Prepara la señal para transmisión
    num_samples = len(signal)

    # Configurar el stream síncrono para la transmisión
    sdr.sync_config(layout=_bladerf.ChannelLayout.TX_X1,
                    fmt=_bladerf.Format.SC16_Q11,  # Formato de datos int16
                    num_buffers=16,
                    buffer_size=8192,
                    num_transfers=8,
                    stream_timeout=3500)

    # Iniciar la transmisión
    print("Iniciando transmisión...")
    tx_ch.enable = True  # Habilitar el canal de transmisión

    start_time = time.time()
    while time.time() - start_time < duration_sec:
        sdr.sync_tx(buf, num_samples)  # Escribir el buffer a BladeRF

    # Finalizar la transmisión
    print("Deteniendo transmisión")
    tx_ch.enable = False  # Deshabilitar el canal de transmisión
```

Esta función toma una señal modulada con preámbulo, la convierte al formato adecuado para el BladeRF y la transmite durante un período especificado. Utiliza un sistema de transmisión síncrona que asegura una comunicación fluida y continua.

4.2. Transmisor: Configuración y Generación de la Señal

El código siguiente presenta la configuración del transmisor para generar y transmitir una señal que incluye un preámbulo seguido de datos modulados.

Parámetros de Configuración

```
# Parámetros de configuración
sample_rate = 5e6           # Tasa de muestreo: 5 MHz
center_freq = 940e6         # Frecuencia central: 940 MHz
gain = 60                   # Ganancia: 60 dB
samples_per_symbol = 8      # Número de muestras por símbolo
duration_sec = 40           # Duración de la transmisión en segundos
beta = 0.2                  # Factor de rodadura del filtro RRC
num_taps = 151              # Número de coeficientes del filtro RRC
```

- **Tasa de muestreo**: La frecuencia a la que se capturan muestras de la señal (5 MHz).
- **Frecuencia central**: La frecuencia de portadora para la transmisión (940 MHz).
- **Ganancia**: La potencia de la señal transmitida, ajustada a 60 dB.
- **Muestras por símbolo**: Define cuántas muestras representan un símbolo en la señal digital.
- **Factor de rodadura (beta)**: Usado para generar el filtro de raíz de coseno alzado (RRC), ajusta el ancho de banda de la señal.
- **Número de coeficientes (num_taps)**: Define la longitud del filtro RRC que se usará para el filtrado.

Generación del Preambulo

Generación del Preambulo

```
# Generación del preámbulo
preamble_bits = np.array([0, 1] * 100)  # Secuencia alternada de 0 y 1, longitud 200 bits
preamble_symbols = 2 * preamble_bits - 1  # Mapear bits a símbolos BPSK (-1, +1)
```

- **Generación de preámbulo**: El preámbulo está formado por una secuencia alternada de bits 0 y 1. Estos se mapean a símbolos BPSK, donde 0 corresponde a -1 y 1 corresponde a +1.

Generación de Datos Aleatorios

Generación de Datos Aleatorios

```
# Generación de datos aleatorios
data_bits = np.random.randint(0, 2, size=1000)  # 1000 bits de datos
data_symbols = 2 * data_bits - 1  # Mapear bits a símbolos BPSK (-1, +1)

# Concatenar preámbulo y datos
symbols = np.concatenate((preamble_symbols, data_symbols))
```

- **Generación de datos**: Se generan 1000 bits aleatorios que representan los datos a transmitir.
- **Mapeo a BPSK**: Los bits son mapeados a símbolos BPSK, lo que implica que los bits 0 se convierten en -1 y los bits 1 en +1.
- **Concatenación**: El preámbulo y los datos se combinan en una sola secuencia de símbolos que será transmitida.

Sobremuestreo de Símbolos y Filtrado

Sobremuestreo y Filtrado de Símbolos

```
# Sobremuestreo de los símbolos
symbols_upsampled = np.zeros(len(symbols) * samples_per_symbol)
symbols_upsampled[::samples_per_symbol] = symbols  # Insertar símbolos con ceros entre ellos

# Generar el filtro RRC
rrc_coef = rrc_filter(beta, samples_per_symbol, num_taps)

# Filtrar la señal
signal_filtered = np.convolve(symbols_upsampled, rrc_coef, mode='same')
```

- **Sobremuestreo**: Se introducen ceros entre los símbolos para incrementar la tasa de muestreo, lo cual es necesario para la transmisión.
- **Filtrado**: Se aplica un filtro de raíz de coseno alzado (RRC) para suavizar la señal y minimizar la interferencia entre símbolos (*ISI*).

Normalización y Conversión a Formato Complejo

Normalización y Conversión a Formato Complejo

```
# Normalizar la amplitud de la señal
signal_filtered /= np.max(np.abs(signal_filtered))

# Convertir a formato complejo (BladeRF espera muestras complejas)
signal_complex = signal_filtered.astype(np.complex64)

np.save('transmitted_signal.npy', signal_complex)
```

- **Normalización**: La señal filtrada es normalizada para asegurar que los valores de la señal estén dentro del rango permitido para la transmisión.
- **Formato complejo**: BladeRF requiere señales en formato de números complejos (I/Q), por lo que la señal se convierte a tipo `complex64`.

Transmisión de la Señal

Transmisión de la Señal

```
# Instanciar el radio y transmitir la señal
radio_tx = BladeRFRadio(sample_rate, center_freq, gain)
radio_tx.start_transmit_with_preamble(signal_complex, duration_sec=
    duration_sec)
```

- **Instanciación del radio**: Se inicializa el dispositivo BladeRF con los parámetros configurados, como la tasa de muestreo y la frecuencia central.
- **Transmisión**: La señal compleja generada se transmite durante el tiempo especificado a través del dispositivo BladeRF.

4.3. Receptor

El receptor recibe las muestras IQ, las filtra y detecta el preámbulo utilizando el algoritmo Schmidl-Cox. Luego, podemos visualizar el espectro de la señal recibida y verificar si el preámbulo ha sido correctamente detectado.

Procesamiento de la Señal Recibida y Detección de Preambulo

Función para Procesar la Señal Recibida y Detectar el Preambulo

```python
def process_preamble(samples):
    """
    Procesa las muestras recibidas para detectar el preámbulo
    usando Schmidl-Cox.

    Parámetros:
    - samples: Array de muestras recibidas

    Retorna:
    - Índice donde se detectó el preámbulo y la métrica M(d)
    """
    rrc_coef = rrc_filter(beta, samples_per_symbol, num_taps)
    samples_filtered = np.convolve(samples, rrc_coef, mode='same')

    # Longitud L para el algoritmo Schmidl-Cox
    L = (preamble_len * samples_per_symbol) // 2
    d_max, M = schmidl_cox_algorithm(samples_filtered, L)
    return d_max, M
```

Esta función filtra las muestras recibidas usando un filtro RRC y luego aplica el algoritmo de Schmidl-Cox para detectar el preámbulo. La métrica $M(d)$ es calculada para determinar si se ha encontrado el preámbulo.

4.4. Análisis

- **Transmisión de la Señal (Transmisor)**: Se utiliza un filtro RRC para suavizar la señal transmitida, lo que reduce el ancho de banda y minimiza la interferencia entre símbolos (ISI). El preámbulo se transmite al principio de la señal, permitiendo al receptor sincronizarse.
- **Detección del Preámbulo (Receptor)**: El receptor filtra la señal recibida con un filtro RRC y utiliza el algoritmo Schmidl-Cox para detectar el preámbulo. El pico en la métrica $M(d)$ indica el inicio del preámbulo.

Resultados de la Recepción y Detección del Preambulo

A continuación, se presentan los resultados obtenidos al aplicar el algoritmo de detección de preámbulo en diferentes escenarios.

i. La figura 6 muestra el espectro de frecuencia, la detección del preámbulo mediante el algoritmo Schmidl-Cox y la señal en el dominio del tiempo cuando **no** se recibe ninguna señal válida. Como se observa en la gráfica, los valores de $M(d)$ indican solo ruido, y no se detecta ningún preámbulo.

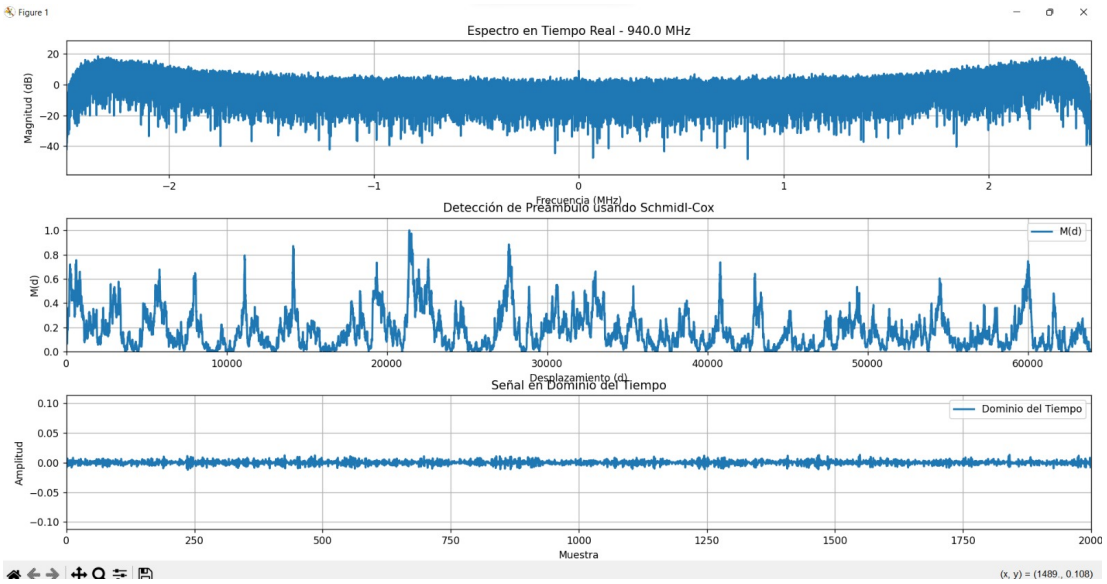

Figura 6: 1) Espectro de Frecuencia, 2) Detección de preámbulo con Schmidl-Cox (se observa ruido), 3) Señal en el dominio del tiempo.

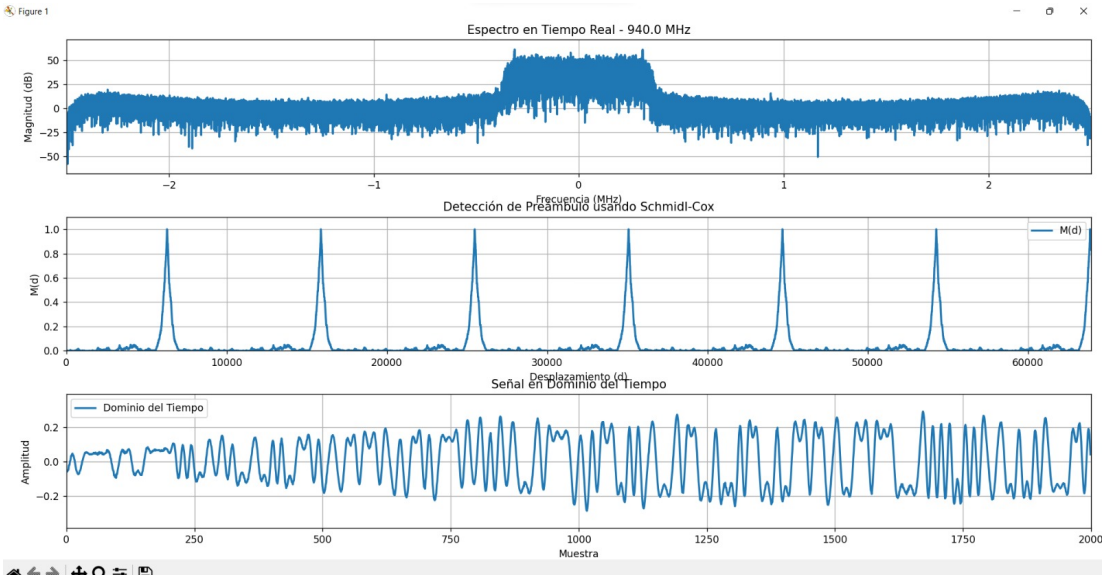

Figura 7: 1) Espectro de Frecuencia, 2) Detección del preámbulo con Schmidl-Cox (El preámbulo se repite cada 10,000 bits aproximadamente), 3) Señal en el dominio del tiempo.

ii. La figura 7 muestra el mismo análisis cuando se transmite y recibe una señal que contiene un preámbulo. En este caso, el preámbulo es claramente visible en el gráfico de $M(d)$ donde se observa un patrón repetitivo cada 10,000 muestras aproximadamente, lo que corresponde a la estructura del preámbulo que se inserta en la señal.

5. Conclusiones

i. **Efectividad del Algoritmo Schmidl-Cox**: El algoritmo de Schmidl-Cox ha demostrado ser efectivo para la detección de preámbulos en señales transmitidas por SDR. A pesar de la presencia de ruido o desplazamientos de frecuencia, la técnica basada en la correlación ha logrado identificar correctamente el inicio del preámbulo en diversas condiciones.

ii. **Impacto del Offset de Frecuencia**: La introducción de un desplazamiento de frecuencia afecta la calidad de la señal recibida, sin embargo, el uso del algoritmo Schmidl-Cox y el filtrado con el filtro de raíz de coseno alzado (RRC) permiten mitigar estos efectos, logrando una detección confiable del preámbulo.

iii. **Importancia del Filtro RRC**: El filtro RRC aplicado tanto en la transmisión como en la recepción ha permitido reducir la interferencia entre símbolos, lo que ha mejorado la precisión en la detección del preámbulo. Su uso es fundamental en la modulación de señales digitales para optimizar el ancho de banda y reducir la distorsión en la señal transmitida.

iv. **Robustez de la Detección en Condiciones de Ruido**: Las pruebas realizadas con señales que no contienen un preámbulo han mostrado que el algoritmo es robusto, al no producir falsos positivos en presencia de ruido. Esto demuestra que el sistema de detección es confiable para identificar únicamente señales válidas con preámbulos.

v. **Transmisión y Recepción en SDR**: El uso del BladeRF como plataforma SDR ha permitido implementar un sistema completo de transmisión y recepción de señales con preámbulo, brindando flexibilidad y control en los parámetros de la señal, como la frecuencia de transmisión, ganancia y el uso de filtros digitales. Esto abre un campo amplio para el diseño y pruebas de sistemas de comunicación inalámbrica.

6. Presentación del informe

- Cuando el docente lo autorice, puede mandar un correo electrónico a: ahilariot@unsa.edu.pe con las siguientes especificaciones:
  - Asunto: Redes Inalámbricas: C - Informe de práctica 2, Equipo 3.
  - El correo electrónico debe contener: Informe en pdf (Escrito en LaTeX), y un enlace para un Drive personal donde estén los programas escritos en Matlab.
  - https://github.com/jorgechvz/redes-inalambricas/tree/main/practica-2