Práctica 8: Probabilidad de Error en PCM - Modulación por Pulsos Codificados

Tyrone Miguel Novillo Bravo
Facultad de Ingeniería
Universidad de Cuenca
tyrone.novillo@ucuenca.edu.ec

Bryam Alexander Misacango
Facultad de Ingeniería
Universidad de Cuenca
bryam.misacango@ucuenca.edu.ec

Luis Miguel Corte
Facultad de Ingeniería
Universidad de Cuenca
luism.corte@ucuenca.edu.ec

Pablo Andrés Calle González
Facultad de Ingeniería
Universidad de Cuenca
pablo.calle1404@ucuenca.edu.ec

Resumen

La siguiente práctica presenta una simulación del sistema de transmisión digital basado en modulación por pulsos codificados (PCM) utilizando codificación de línea Polar NRZ. El objetivo principal es analizar la probabilidad de error de bit (BER) en presencia de ruido aditivo blanco gaussiano (AWGN), comparando los resultados simulados con las expresiones teóricas correspondientes. Asimismo, se exploran variantes como Unipolar NRZ y Bipolar RZ y las modificaciones necesarias para su implementación.

Index Terms

PCM, BER, codificación de línea, Polar NRZ, Unipolar NRZ, Bipolar RZ, AWGN, SNR.

OBJETIVOS:

- Verificar la Probabilidad de Error en PCM a través de una simulación.

I. INTRODUCCIÓN

En este trabajo se realiza una simulación de un sistema de transmisión digital basado en modulación por pulsos codificados (PCM). El objetivo principal es analizar cómo afecta el ruido al proceso de transmisión de bits, observando la probabilidad de error (BER) que se genera al recibir los datos.

Para esto, se construye un sistema que incluye un transmisor, un canal con ruido y un receptor. Se utiliza la codificación de línea Polar NRZ para representar los bits como señales, y se simula el canal con ruido blanco gaussiano (AWGN). Luego se mide la cantidad de errores que ocurren al comparar los bits enviados con los recibidos.

Además, se comparan diferentes tipos de codificación (Unipolar NRZ y Bipolar RZ) para ver cómo cambian los resultados. Finalmente, se analizan los gráficos obtenidos y se discute cuál codificación ofrece mejores resultados frente al ruido.

II. MARCO TEÓRICO

El sistema de modulación por pulsos codificados (PCM) es una técnica utilizada en comunicaciones digitales para transmitir datos binarios a través de un canal. Su funcionamiento se puede dividir en tres bloques principales: el transmisor, el canal y el receptor, como se muestra en la Figura 1.

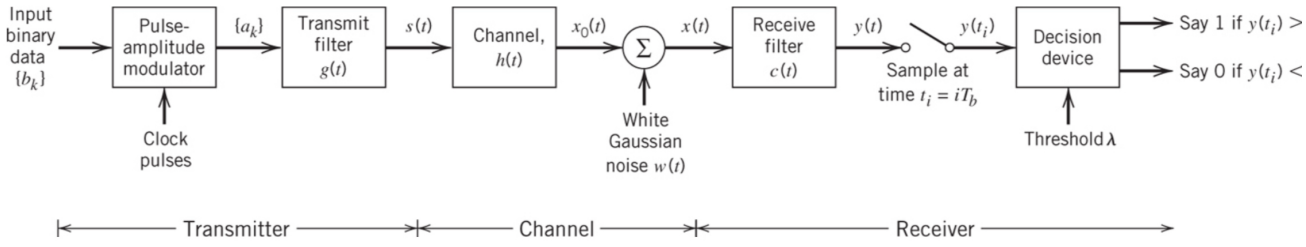

Figura 1: Esquema de un sistema PCM con canal ruidoso y detección por umbral.

En este esquema, los datos binarios de entrada se convierten en niveles de amplitud mediante un modulador, y luego se filtran para formar la señal transmitida $s(t)$. Esta señal pasa por el canal $h(t)$, el cual en esta práctica se considera ideal ($h(t) = \delta(t)$), pero con la adición de ruido blanco gaussiano $w(t)$.

La señal ruidosa resultante $x(t)$ es procesada por un filtro receptor $c(t)$, también llamado filtro acoplado o *matched filter*, el cual tiene como objetivo maximizar la relación señal a ruido (SNR) en la salida. La señal filtrada $y(t)$ se muestrea en instantes definidos por el periodo de bit T_b, y finalmente se toma una decisión comparando cada muestra contra un umbral λ.

Probabilidad de error para distintas codificaciones

La probabilidad de error de bit (BER) representa la probabilidad de que un bit recibido difiera del bit transmitido. Esta probabilidad depende de la relación señal a ruido $\frac{E_b}{N_0}$, donde E_b es la energía por bit y N_0 la densidad espectral de potencia del ruido.

Para canales afectados por ruido blanco gaussiano aditivo (AWGN), las fórmulas teóricas de la probabilidad de error varían según el tipo de codificación utilizada:

1. **Polar NRZ:** utiliza niveles $\pm A$, con energía por bit $E_b = A^2 T_b$. La probabilidad de error se calcula como:

$$P_e = Q\left(\sqrt{2 \cdot \frac{E_b}{N_0}}\right) \tag{1}$$

2. **Unipolar NRZ:** utiliza niveles 0 y $+A$, con energía promedio por bit reducida: $E_b = \frac{A^2 T_b}{2}$. La fórmula de error es:

$$P_e = Q\left(\sqrt{\frac{E_b}{N_0}}\right) \tag{2}$$

3. **Bipolar RZ:** alterna pulsos $+A$ y $-A$ durante $T_b/2$, con retorno a cero, y energía por bit $E_b = \frac{A^2 T_b}{2}$. Su probabilidad de error es similar a la de Unipolar NRZ:

$$P_e = Q\left(\sqrt{\frac{E_b}{N_0}}\right) \tag{3}$$

Donde la función $Q(x)$ es la cola de la distribución normal, y puede expresarse en términos de la función complementaria del error:

$$Q(x) = \frac{1}{2} \cdot \text{erfc}\left(\frac{x}{\sqrt{2}}\right) \tag{4}$$

Estas fórmulas permiten estimar de manera analítica el comportamiento ideal del sistema frente al ruido, y son utilizadas como referencia para validar los resultados obtenidos por simulación. [1]

III. Desarrollo

1. Generación del Flujo de Bits

La generación del flujo de bits binarios aleatorios constituye el punto de partida del sistema PCM. En esta práctica se definió una longitud de $N = 10000$ bits, con igual probabilidad de ocurrencia para los símbolos 0 y 1, es decir, $P_0 = P_1 = 0{,}5$. Esta decisión garantiza una distribución balanceada, ideal para evaluar de manera imparcial el comportamiento del sistema frente a ruido.

La generación se realizó mediante la función generar_bits, que utiliza el método ‘np.random.choice‘ para seleccionar aleatoriamente entre los valores 0 y 1 con las probabilidades indicadas. El código correspondiente es el siguiente:

Listing 1: Generación de bits aleatorios

```
def generar_bits(N, P0=0.5):

    Genera una secuencia aleatoria de bits con probabilidades P0 y P1.

    Parametros:
    -----------
    N : int
        Numero de bits a generar
    P0 : float
        Probabilidad del bit 0 (default: 0.5)

    Retorna:
```

```
    --------
    bits : array
        Array de bits (0 y 1)

    return np.random.choice([0, 1], size=N, p=[P0, 1-P0])
```

Esta función es invocada en la sección principal del código para generar la secuencia base sobre la cual se aplican todos los procesos subsiguientes del sistema PCM. A continuación se muestra la línea donde se llama a la función, pasando como argumento la cantidad de bits definida previamente:

Listing 2: Llamada a la función para generar los bits

```
bits_originales = generar_bits(Nbits, P0)
```

La variable 'bits_originales' contiene entonces el flujo de bits aleatorio que será codificado, transmitido y analizado a lo largo de la simulación. Cabe destacar que se utiliza una semilla ('np.random.seed(42)') para garantizar la reproducibilidad de los resultados.

2. Codificación de Línea: Polar NRZ

Una vez generado el flujo de bits binarios, el siguiente paso en el sistema PCM es aplicar una codificación de línea que permita representar digitalmente estos bits como una señal continua. En este caso, se ha optado por la codificación Polar NRZ (Non-Return to Zero), la cual asocia a cada bit un nivel de amplitud constante durante el tiempo de bit T_b, sin retorno a cero entre símbolos consecutivos.

En particular, para esta codificación se define el siguiente mapeo:

- Bit 1 $\rightarrow +A$
- Bit 0 $\rightarrow -A$

Este tipo de codificación es eficiente en términos de ancho de banda y energía, y es ampliamente utilizada en sistemas digitales. La función que implementa esta lógica es `codificar_polar_nrz`, y su código se presenta a continuación:

Listing 3: Codificación de bits con Polar NRZ

```
def codificar_polar_nrz(bits, A):
    # Codifica bits usando Polar NRZ
    # Bit 1 -> +A, Bit 0 -> -A
    # bits: secuencia de bits PCM
    # A: amplitud del pulso
    # Retorna: senal codificada (un valor por bit)
    senal = np.where(bits == 1, A, -A)
    return senal
```

La función `np.where` realiza la asignación directa de valores de amplitud en función del valor binario del bit. El resultado es una señal digital de amplitud constante por bit, alternando entre $+A$ y $-A$ según corresponda.

La función es utilizada inmediatamente después de generar los bits, como se muestra en la siguiente línea:

Listing 4: Codificación de la secuencia de bits

```
senal\_tx = codificar_polar_nrz(bits_originales, A)
```

El vector `senal_tx` representa la señal codificada que será transmitida por el canal. En este punto, cada bit ya ha sido transformado en un símbolo con amplitud definida y listo para ser afectado por el canal de comunicaciones.

3. Simulación del Canal con Ruido Blanco Gaussiano

El canal de comunicaciones es modelado como un canal con ruido aditivo blanco gaussiano (AWGN), una suposición común para simular entornos ideales de transmisión digital. En este caso, se ignoran efectos de atenuación o dispersión, y se considera que la única distorsión proviene del ruido.

La función que implementa este canal es `canal_awgn`, donde se agrega ruido gaussiano a la señal transmitida de tal forma que se mantenga una relación señal a ruido (SNR) específica, definida en decibelios.

Listing 5: Simulación del canal con ruido AWGN

```
def canal_awgn(senal, SNR_dB, A, Tb):
    # Agrega ruido AWGN al canal con un SNR especifico
```

```
SNR_linear = 10**(SNR_dB / 10)
Eb = A**2 * Tb
N0 = Eb / SNR_linear
sigma2 = N0 / (2 * Tb)
sigma = np.sqrt(sigma2)
ruido = np.random.normal(0, sigma, len(señal))
señal_ruidosa = señal + ruido
return señal_ruidosa, ruido
```

En esta función, se convierte el valor de SNR en decibelios a su forma lineal utilizando la siguiente expresión:

$$SNR_{linear} = 10^{\frac{SNR_{dB}}{10}} \tag{5}$$

Posteriormente, se calcula la energía por bit E_b para la codificación Polar NRZ, que se define como el producto entre el cuadrado de la amplitud del pulso y el tiempo de bit:

$$E_b = A^2 \cdot T_b \tag{6}$$

A partir de esta energía y del valor de SNR, se obtiene la densidad espectral de potencia del ruido N_0 mediante:

$$N_0 = \frac{E_b}{SNR_{linear}} \tag{7}$$

Ya que se trabaja con un valor por bit, la varianza del ruido por símbolo es:

$$\sigma^2 = \frac{N_0}{2T_b} \tag{8}$$

Y la desviación estándar correspondiente es:

$$\sigma = \sqrt{\sigma^2} \tag{9}$$

Finalmente, se genera un vector de ruido gaussiano con media cero y desviación estándar σ, el cual se suma a la señal transmitida para obtener la señal ruidosa que será procesada por el receptor.

Este procedimiento asegura que la señal recibida tenga un SNR igual al especificado.

4. Recepción y Filtro Acoplado

Una vez que la señal ha atravesado el canal ruidoso, el receptor debe procesarla para recuperar la secuencia de bits original. Para ello, se implementa un filtro acoplado (*matched filter*), el cual maximiza la relación señal a ruido (SNR) en la salida, siempre que la forma del pulso transmitido sea conocida.

En el caso de la codificación Polar NRZ, donde el pulso tiene amplitud constante A durante un intervalo T_b, el filtro acoplado ideal es otro pulso rectangular, simétrico en el tiempo respecto al transmitido:

$$h(t) = p(T_b - t) \tag{10}$$

Dado que en esta simulación se trabaja directamente con valores ya muestreados (un valor por bit), la implementación del filtro acoplado se simplifica considerablemente. En vez de una convolución completa, se realiza una multiplicación escalar por T_b, con el fin de mantener la energía correcta de cada símbolo:

$$y[n] = r[n] \cdot T_b \tag{11}$$

Esto equivale a integrar el valor de la señal recibida a lo largo del tiempo de bit, lo cual es consistente con la teoría del filtro acoplado. El código correspondiente es:

Listing 6: Implementación del filtro acoplado

```
def filtro_acoplado(senal, A, Tb):
    # Implementa el filtro acoplado para Polar NRZ
    # La salida se obtiene multiplicando cada muestra por Tb
    salida_filtro = senal * Tb
    return salida_filtro
```

Una vez obtenida la salida del filtro, se realiza una decisión binaria utilizando un umbral. Para codificación Polar NRZ con probabilidades $P_0 = P_1 = 0{,}5$, el umbral óptimo es 0. Si la muestra filtrada es mayor a cero, se decide un 1; en caso contrario, un 0:

$$\text{bit}_{\text{decidido}} = \begin{cases} 1 & \text{si } y[n] > 0 \\ 0 & \text{si } y[n] \leq 0 \end{cases} \tag{12}$$

La decisión se implementa mediante:

Listing 7: Decisión de bits a partir del filtro

```
def decisor(salida_filtro, umbral=0):
    # Decide los bits comparando la salida del filtro con un umbral
    bits_decididos = (salida_filtro > umbral).astype(int)
    return bits_decididos
```

La Figura 2 se muestra un ejemplo con los primeros 50 bits transmitidos bajo un SNR de 6 dB. Se observan claramente las tres etapas: señal transmitida codificada en niveles $\pm A$, la señal recibida con el efecto del ruido AWGN, y la salida del filtro acoplado con las muestras alineadas con el centro de cada bit.

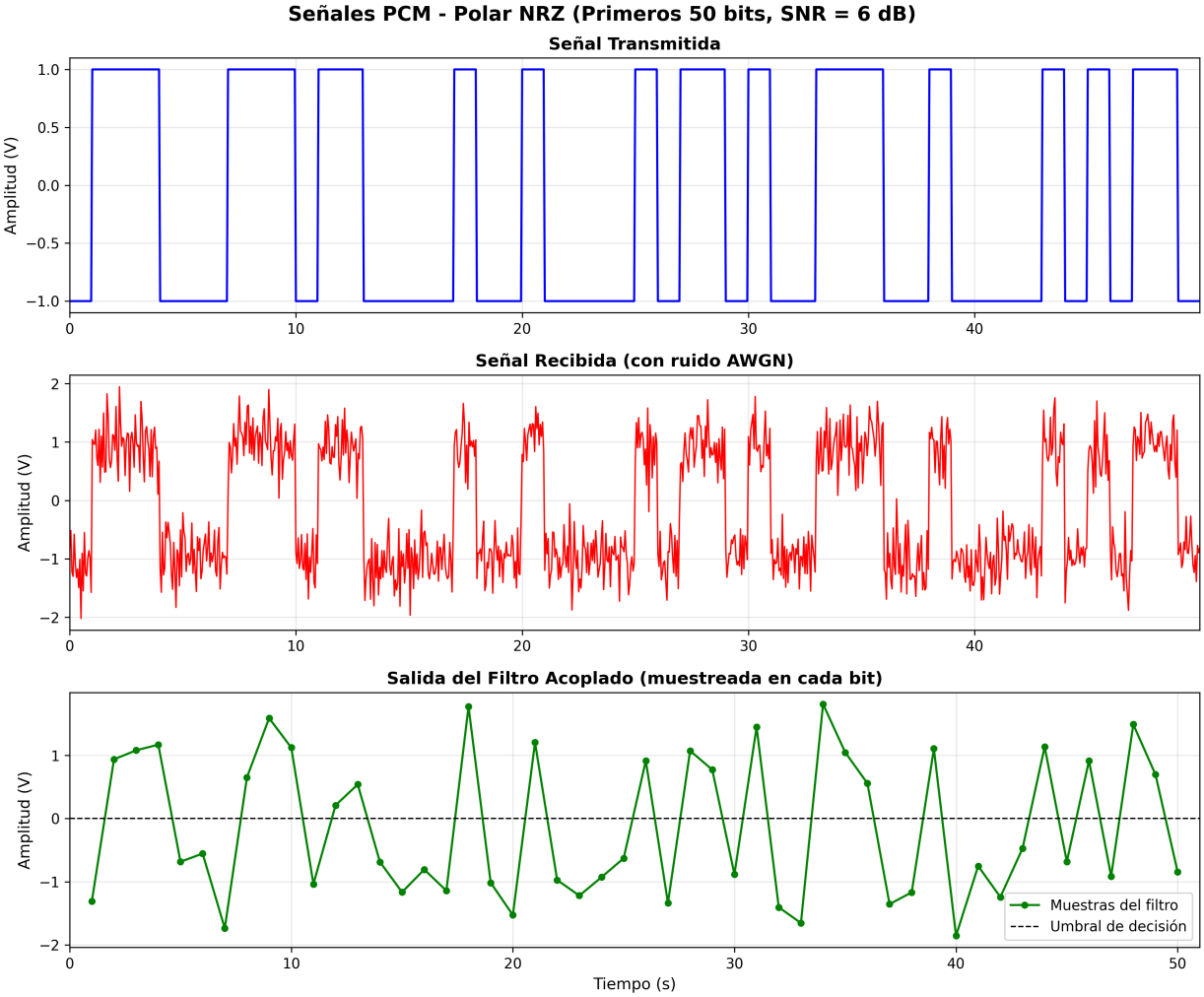

Figura 2: Señales PCM para los primeros 50 bits. Se observa la señal transmitida (arriba), la señal recibida con ruido (medio), y la salida del filtro acoplado (abajo), con el umbral de decisión marcado.

5. *Cálculo de la Tasa de Error (BER)*

Una vez que los bits han sido recibidos y detectados, se debe evaluar el rendimiento del sistema digital mediante la estimación de la tasa de error de bits o *Bit Error Rate* (BER). Este parámetro cuantifica la proporción de bits transmitidos que fueron erróneamente recibidos debido al efecto del ruido en el canal.

La expresión matemática para calcular la probabilidad de error simulada es:

$$P_e = \frac{N_{\text{errores}}}{N_{\text{total}}} \tag{13}$$

Donde:

- N_{errores} es el número de bits que difieren entre la secuencia transmitida y la secuencia recibida.
- N_{total} es el número total de bits transmitidos.

Este cálculo se implementa en la función `calcular_ber`, que compara ambas secuencias y devuelve tanto la tasa de error como la cantidad de errores absolutos:

Listing 8: Cálculo de la tasa de error de bits

```
def calcular_ber(bits_originales, bits_recibidos):
    # Calcula la tasa de error de bits (BER)
    errores = np.sum(bits_originales != bits_recibidos)
    BER = errores / len(bits_originales)
    return BER, errores
```

Durante la simulación, este proceso se repite para diferentes valores de SNR, generando así una curva que muestra cómo varía el BER en función del ruido presente en el canal. Esto permite analizar el comportamiento del sistema en distintos escenarios de interferencia.

Para cada iteración, también se calcula la tasa de error teórica correspondiente a la codificación Polar NRZ, usando la siguiente expresión:

$$P_{e,teo} = Q\left(\sqrt{2 \cdot \frac{E_b}{N_0}}\right) \tag{14}$$

Donde $Q(x)$ es la función Q de la distribución normal, que puede expresarse con la función complementaria del error:

$$Q(x) = \frac{1}{2} \cdot \text{erfc}\left(\frac{x}{\sqrt{2}}\right) \tag{15}$$

Este valor se calcula en la función `ber_teorica_polar_nrz`:

Listing 9: Cálculo de BER teórica para Polar NRZ

```
def ber_teorica_polar_nrz(SNR_dB):
    SNR_linear = 10**(SNR_dB / 10)
    argumento = np.sqrt(2 * SNR_linear)
    BER = 0.5 * special.erfc(argumento / np.sqrt(2))
    return BER
```

6. *Comparación con la Probabilidad de Error Teórica*

Para validar la simulación realizada, se comparan los resultados obtenidos para la tasa de error de bits (BER) con los valores teóricos correspondientes a la codificación Polar NRZ. En la Figura 3, se representa gráficamente la probabilidad de error en función del SNR, tanto para la simulación como para la expresión teórica:

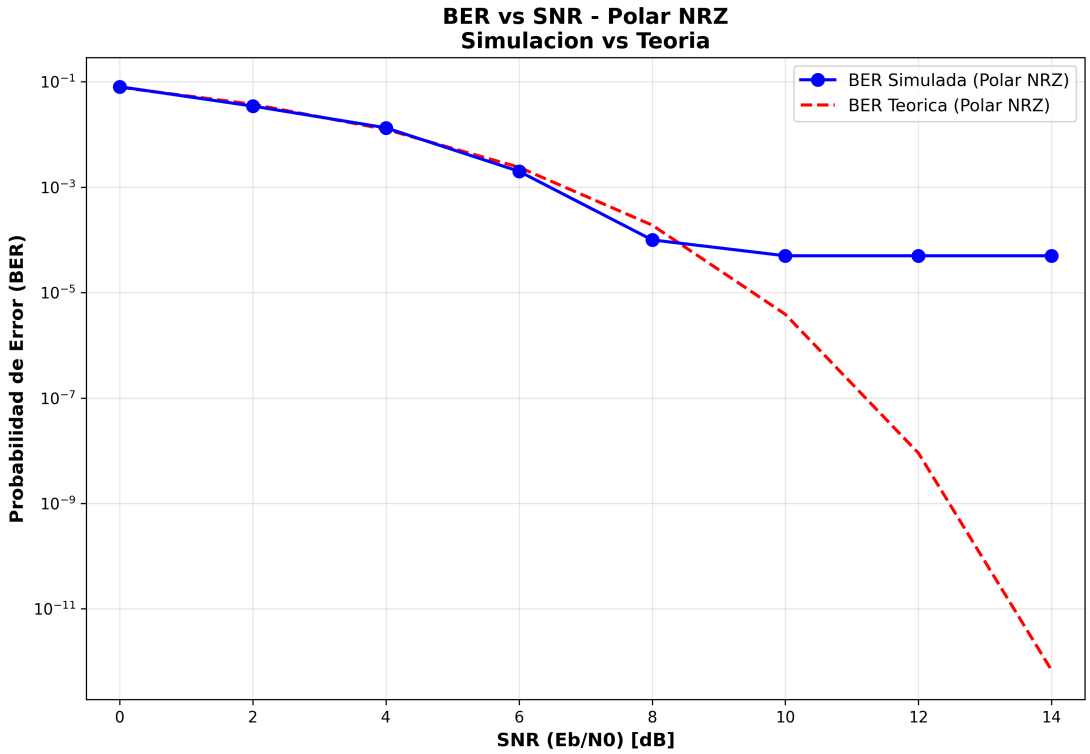

Figura 3: Comparación entre la BER simulada y la BER teórica para codificación Polar NRZ. Se observa una buena concordancia hasta aproximadamente 8 dB.

Como se puede observar, existe una muy buena coincidencia entre ambas curvas hasta aproximadamente SNR $= 8$ dB. A partir de este punto, la probabilidad de error real se vuelve tan pequeña que resulta difícil de estimar de forma precisa utilizando solamente 10,000 bits en la simulación. Esto se debe a que, con tan pocos errores observables, la estimación se vuelve ruidosa o incluso puede dar como resultado una tasa de error nula, lo cual es estadísticamente inexacto para un análisis de desempeño a largo plazo.

Para poder estimar con mayor precisión la BER en altos valores de SNR, sería necesario utilizar secuencias de mayor longitud, por ejemplo 10^6 o incluso 10^7 bits, de manera que se obtenga un número suficiente de errores para realizar una estimación estadísticamente confiable.

Es importante resaltar que durante toda la simulación se trabajó con un mismo flujo de bits fijo para todos los valores de SNR. Esto garantiza que las variaciones en la BER simulada se deban exclusivamente al efecto del ruido y no a cambios en los datos transmitidos, asegurando una comparación justa.

7. Análisis Comparativo entre esquemas de Codificación

Además de la codificación Polar NRZ, existen otras variantes de codificación de línea que pueden emplearse en un sistema PCM como la codificación Unipolar NRZ y Bipolar RZ. Cada una tiene diferencias en cuanto a su inmunidad al ruido y energía.

La Figura 4 muestra una comparación entre las curvas teóricas de BER para los tres esquemas mencionados:

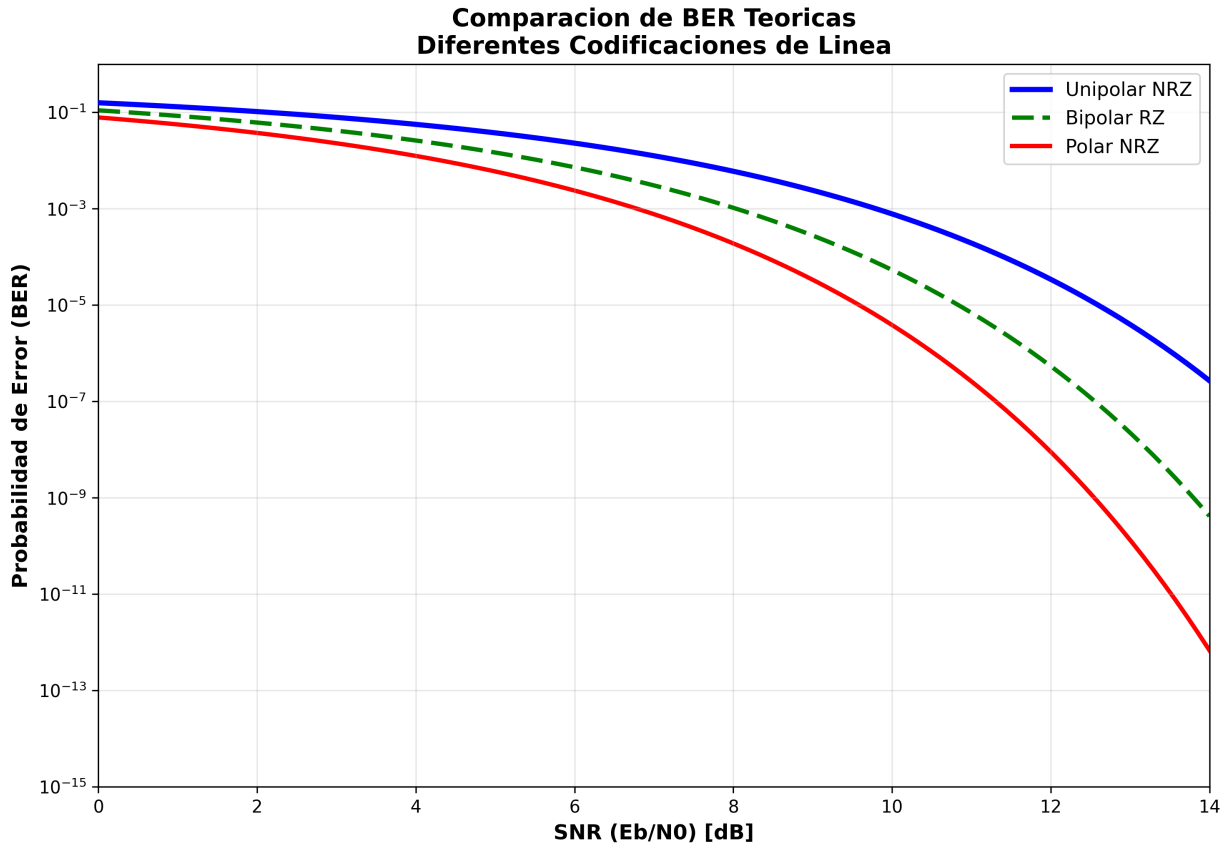

Figura 4: Comparación de las curvas teóricas de BER para las codificaciones Unipolar NRZ, Bipolar RZ y Polar NRZ. Se observa que Polar NRZ presenta el mejor desempeño, seguido por Bipolar RZ.

Como puede observarse, la codificación Polar NRZ ofrece la mejor tasa de error para un mismo valor de $\frac{E_b}{N_0}$, ya que aprovecha toda la amplitud disponible en ambas direcciones ($\pm A$), maximizando así la distancia entre símbolos. La codificación Bipolar RZ presenta un rendimiento intermedio, mientras que Unipolar NRZ es la menos eficiente, ya que utiliza únicamente un nivel positivo (0 y $+A$) y por tanto reduce la energía promedio disponible por bit.

Para implementar Unipolar NRZ, se deben considerar los siguientes cambios:

1. **Codificador:** En lugar de mapear los bits como $\pm A$, el bit 1 se codifica como $+A$, y el bit 0 como 0.

2. **Filtro acoplado:** El filtro sigue siendo un integrador del pulso, por lo tanto se puede seguir multiplicando por T_b. Sin embargo, al tener el bit 0 representado como 0 V, la energía promedio por bit disminuye a la mitad:

$$E_b = \frac{A^2 T_b}{2} \tag{16}$$

3. **Decisor:** El umbral óptimo de decisión ya no es cero, sino la mitad de la amplitud:

$$\text{umbral} = \frac{A}{2} \tag{17}$$

En el caso de Bipolar RZ, se codifica el bit 1 con un pulso positivo de media duración $T_b/2$, y el bit 0 con un pulso negativo del mismo ancho. En la segunda mitad del intervalo T_b, la señal retorna a cero. Esto implica:

1. **Codificador:** Se deben generar dos niveles ($+A$, $-A$) por $T_b/2$, seguidos de ceros (más complejo).
2. **Filtro acoplado:** El filtro debe adaptarse a la nueva forma del pulso, realizando una correlación con un pulso de A o $-A$ activo solo en la primera mitad del bit.
3. **Energía del bit:** Debido a que el pulso tiene la mitad de duración, la energía por bit también se reduce:

$$E_b = \frac{A^2 T_b}{2} \tag{18}$$

4. **Decisor:** El umbral de decisión permanece en cero.

IV. Conclusiones

- En la simulación, se evidenció que a partir de un SNR de 8 dB, la estimación de la BER simulada comienza a separarse de la curva teórica debido a la baja cantidad de errores detectados. Para obtener resultados reales se necesita transmitir un mayor número de bits.
- **Polar NRZ** ofrece el mejor desempeño en términos de probabilidad de error de bit (BER), ya que aprovecha toda la amplitud disponible para maximizar la distancia entre símbolos. Además, su implementación es sencilla y eficiente en cuanto a energía.
- **Unipolar NRZ**, aunque más simple debido a que utiliza solo niveles positivos (0 y $+A$), presenta un rendimiento inferior en BER. Esta desventaja se debe a que la energía promedio transmitida por bit es menor, lo cual afecta negativamente la capacidad del sistema para resistir el ruido.
- **Bipolar RZ** facilita la sincronización entre transmisor y receptor por el retorno a cero. Sin embargo, viene acompañado de una reducción en la energía por bit y una mayor complejidad en la generación de la señal, afectando su eficiencia.
- La codificación de línea y el nivel de ruido influyen en el desempeño de un sistema de comunicación digital. Además, es importante diseñar un receptor adecuado, en este caso, filtro acoplo yun dispositivo de decisión, basado en características del transmisor.

Referencias

[1] S. Haykin and M. Moher, *Communication Systems*. John Wiley & Sons, 2009. [Online]. Available: https://Communication-Systems-5-Simon-Haykin/dp/0470169966

Repositorio del Código

El código fuente utilizado para la simulación de este informe se encuentra disponible en el siguiente repositorio de GitHub: https://github.com/thyron001/pcm