Teoría de la Información y Codificación

Trabajo Práctico de Simulación

Nihany Santiago (03012/3)
santi.nihany@gmail.com

Índice

1 Comuniación digital 1
1.1 Introducción 1
1.2 Marco teórico 1
1.2.1 Ruido AWGN y modulación BPSK 1
1.2.2 Códigos de bloque lineales 1
1.2.3 Ganancia asintótica de codificación 2
1.2.4 Demodulación 2
1.3 Discusión 3
1.3.1 Diseño del sistema 3
1.3.2 Simulación 3
1.4 Resultados 5
1.4.1 Análisis P_{eb} vs $\frac{E_b}{N_0}$ 5
1.4.2 Análisis P_{ep} vs $\frac{E_b}{N_0}$ 6
1.4.3 Ganancia de codificación G_c vs $\frac{E_b}{N_0}$ 6
1.4.4 Tablas de resultados de ejecución 7
2 Codificación de fuente 8
2.1 Introducción 8
2.2 Marco teórico 8
2.2.1 Compresión 8
2.2.2 Codificación de Huffman 8
2.2.3 Extensión de fuente 8
2.3 Procedimiento 8
2.3.1 Procesamiento de la imagen 8
2.3.2 Cálculo de probabilidades 8
2.3.3 Extensión de la fuente 8
2.3.4 Construcción y evaluación del código de Huffman 8
2.4 Resultados 9
2.5 Conclusiones 10

1 Comuniación digital

1.1 Introducción

El propósito de este trabajo ha sido investigar cómo un esquema de codificación de canal mejora la confiabilidad de un enlace digital en presencia de ruido aditivo gaussiano blanco (AWGN). Para ello se ha seleccionado un código de bloque lineal sistemático (n,k) == (10,14) combinándolo con una modulación BPSK de decisión dura. Se analizó el comportamiento de la comunicación utilizando un decodificador como corrector y detector. Además, se realizó una cuantificación de la ganancia real que aporta el código y una comparación con las fórmulas teóricas.

1.2 Marco teórico

1.2.1 Ruido AWGN y modulación BPSK

El **canal AWGN** (Additive White Gaussian Noise) se caracteriza por añadir a cada muestra de la señal transmitida un valor de ruido gaussiano con media cero y densidad espectral de potencia constante $\frac{N_0}{2}$. Matemáticamente, si $s(t)$ es la señal enviada, la recibida es:

$$r(t) = s(t) + n(t), n(t) \sim \mathbb{N}\Big(0, \frac{N_0}{2}\Big)$$

La modulación **BPSK** (Binary Phase Shift Keying) representa los bits mediante dos símbolos antipodales:

$$0 \to +A, 1 \to -A$$

donde la probabilidad de error de bit teórica del canal es:

$$P_{\text{eb}} = Q\left(\sqrt{\frac{2E_b}{N_0}}\right)$$

1.2.2 Códigos de bloque lineales

Un **código de bloque lineal** (n, k) traduce **palabras de fuente** de longitud k a **palabras código** de longitud n.

Para realizar la codificación, se utiliza una **matriz generadora $\boldsymbol{G}$** que convierte una palabra de fuente u en palabra código $v = u \cdot G$. Las filas de G son palabras de código y las palabras de código son combinación lineal de las filas de G. Esta matriz se define:

$$G = [I_k \mid P]$$

de modo que los primeros k bits de v coincidan con u, y la submatriz P aporta la redundancia.

Para la decodificación es utilizada la **matriz de verificación de paridad $\boldsymbol{H^T}$**, que satisface $G \cdot H^T = 0$ y permite calcular el **síndrome** $s = v \cdot H^T$. En forma sistemática:

$$H = [P^T \mid I_{\text{n-k}}]$$

Cuando el cálculo del síndrome es cero, se puede afirmar que hubo error en la transmisión.

El **peso** (w_H) de una palabra de código es el número de sus elementos que son distintos de cero y la **distancia** entre dos palabras de código es la distancia de Hamming (d_H) entre ellos, es decir, el número de elementos en los que difieren.

Un código con $d_{\min}$ ($\text{mín}_{\text{i,j}}\{d_{\text{H}}(v_i, v_j)\}$) corrige hasta:

$$t_c = \left\lfloor \frac{d_{\min}-1}{2} \right\rfloor$$

errores arbitrarios por palabra, y detecta hasta:

$$t_d = d_{\min} - 1$$

La **cota de Hamming** establece la cantidad de bits de redundancia $(n-k)$ necesarios en para formar un código de bloque lineal que puede corregir hasta t_c errores. Se expresa como:

$$\sum_{i=0}^{t_c} \binom{n}{i} \leq 2^{n-k}$$

El miembro izquierdo cuenta el número de esquemas de error que se deben poder corregir y el miembro derecho representa la cantidad de síndromes posibles. Esto implica que, para que un código pueda corregir t_c errores, debe haber suficientes síndromes distintos para representar todos los errores posibles.

1.2.3 Ganancia asintótica de codificación

La **tasa** del código es $R = \frac{k}{n}$. Bajo decisión dura, la ganancia de codificación asintótica en dB se define como:

$$G_a = 10\log(R \cdot t_c) = 10\log\left(\tfrac{k}{n} \cdot \left\lfloor \tfrac{d_{\min}-1}{2} \right\rfloor\right)$$

Este valor indica el ahorro teórico de energía por bit de información al operar en la región asintótica de bajas tasas de error.

1.2.4 Demodulación

- **Decisión dura**: el demodulador obtiene bits discretos $\{0, 1\}$ y el decodificador trabaja directamente sobre ellos. Es simple y rápido, pero desprecia la información de confiabilidad de cada bit.
- **Decisión blanda**: se emplean los valores continuos recibidos (por ejemplo, log-likelihood ratios) para ponderar las probabilidades de cada bit, mejorando el rendimiento en aproximadamente 2–3 dB respecto a la decisión dura, a costa de mayor complejidad computacional y de memoria.

En este trabajo, se modelará el sistema utilizando decisiones duras.

1.3 Discusión

1.3.1 Diseño del sistema

Para nuestro código de bloque trabajamos con palabras de longitud total $n = 14$, de las cuales $k = 10$ son bits de información y $n - k = 4$ bits de paridad. Aplicando la cota de Hamming,

$$\sum_{i=0}^{t_c} \binom{n}{i} \leq 2^{n-k} = 16$$

se comprueba que para $t_c = 1$ se cumple $\binom{14}{0} + \binom{14}{1} = 15 \leq 16$, mientras que para $t_c = 2$ ya supera el límite. De ello se deduce:

- Corrección de errores: $t_c = 1$
- Distancia mínima: $d_{\min} = 2t_c + 1 = 3$
- Detección de errores: $t_d = d_{\min} - 1 = 2$

Este código puede corregir automáticamente un error por palabra y detectar hasta dos sin riesgo de mal corrección.

Se adopta la forma sistemática

$$G = [I_{10} \mid P]$$

con

$$P = \begin{pmatrix} 1100 \\ 0110 \\ 0011 \\ 1001 \\ 1010 \\ 0101 \\ 1101 \\ 1110 \\ 0111 \\ 1011 \end{pmatrix}$$

de modo que las filas de P sean distintas y con peso $w_H \geq d_{\min} - 1$.

La matriz de paridad se construye como:

$$H = [P^T \mid I_4]$$

1.3.2 Simulación

El rango de relación señal-ruido por bit empleado es $\frac{E_b}{N_0} = [1..7\text{dB}]$ (paso 0.25 dB). Se decidió utilizar este rango de valores porque, a partir de 7dB, el tiempo de ejecución de la simulación aumentaba considerablemente.

1.3.2.1 Generación de secuencias de prueba

Para cada valor de $\frac{E_b}{N_0}$ se determina un número de palabras M suficientemente grande — al menos 10^6 o adaptado a la probabilidad teórica de error de bit— y se construye la matriz U con palabras de fuente aleatorias de largo k.

1.3.2.2 Codificación y modulación BPSK

Cada fila de U se convierte en palabra código multiplicando la matriz por la matriz generadora y obteniendo $V = U \cdot G$. Luego, el mapeo BPSK transforma ceros en +1 y unos en −1. El canal AWGN añade ruido gaussiano de varianza $\frac{N_0}{2}$ a cada símbolo.

1.3.2.3 Decodificación

A partir de la salida R, se calcula la matriz de síndromes:

$$S = R \cdot H^T$$

Utilizando las matrices R y S podemos obtener identificar los errores en la trasmisión y construir la matriz V_e con las palabras de código originales. Como definimos anteriormente, las filas de S no nulas indican que hubo un error de transmisión en la palabra de código asociada a esa fila. Podemos emplear dos modos para construir V_e:

1. **Modo corrector**: Cada síndrome s_i no nulo se compara con las filas de H^T. Si el síndrome concuerde con la fila número j, se invierte el bit j en la palabra recibida.

2. **Modo detector**: Se eliminan de R las palabras correspondientes a los síndromes no nulos.

1.3.2.4 Cálculo de métricas

Para calcular los errores de bit: obtenemos una matriz de error E comparando la matriz de bits estimados U_e (primeras k columnas de V_e) con los transmitidos U:

$$E = U \oplus U_e$$

La cantidad de **filas no nulas** de E será el número de **palabras erradas**. Se estima la P_{ep} dividiendo por la cantidad total de filas M.

La cantidad de **elementos no nulos** de E será el número de **bits errados**. Se estima la P_{eb} dividiendo por la cantidad total de elementos $M \times k$.

1.3.2.5 Cálculo de ganancia

La ganancia de código se mide para una P_{eb} dada:

$$G_c = \frac{\frac{E_b}{N_0}\ \text{sin cod.}}{\frac{E_b}{N_0}\ \text{con cod.}}\Big|_{@P_{\text{eb}}}$$

Calculamos $\frac{E_b}{N_0}$ con cod. para cada iteración utilizando la función inversa de $Q(.)$ y la probabilidad de error de bit asociada. Obtenemos G_c restando estos valores por los $\frac{E_b}{N_0}$ sin cod. asociados a cada iteración.

1.4 Resultados

1.4.1 Análisis P_{eb} vs $\frac{E_b}{N_0}$

En la **Figura 2.1** (modo corrector) se observa una mejora en la probabilidad de error de bit al aplicar codificación. La curva simulada con código decrece más rápido que la del sistema sin codificar y se acerca al límite teórico con decisión dura. A 7 dB, el sistema codificado alcanza un $P_{\text{eb}} \approx 10^{-4}$, mientras que el sistema sin código queda en torno a 10^{-2}.

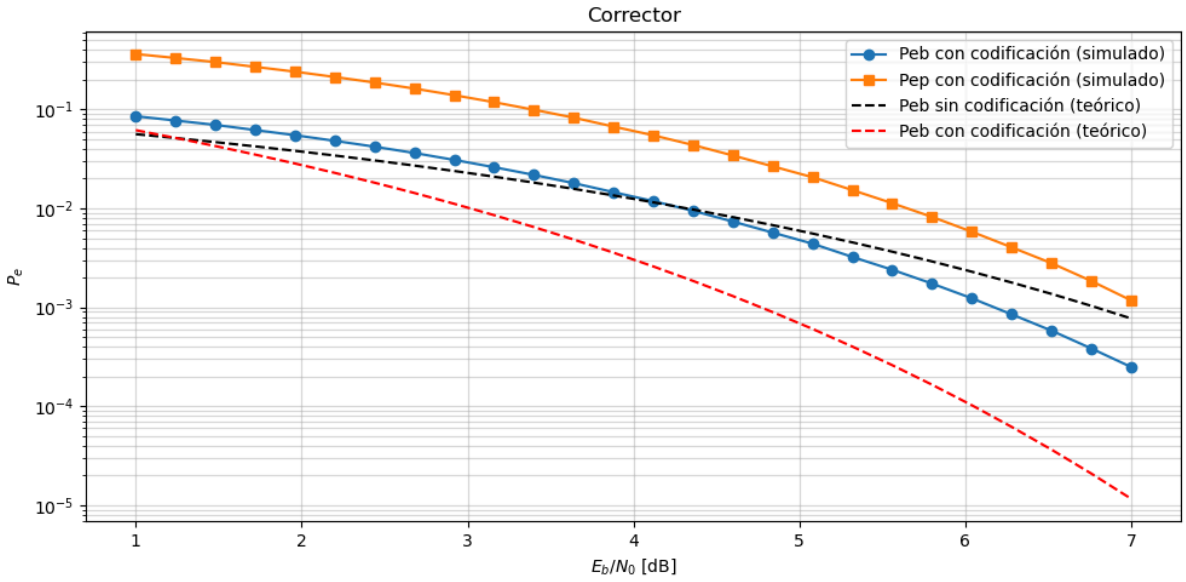

Figure 2.1: Probabilidad de error de bit y palabra en modo corrector

En la **Figura 2.2** (modo detector), la curva de P_{eb} simulada se sitúa por debajo del límite teórico, debido a que se descartan todas las palabras con errores detectados. Esto mejora el rendimiento de bit, a costa de un incremento en P_{ep}.

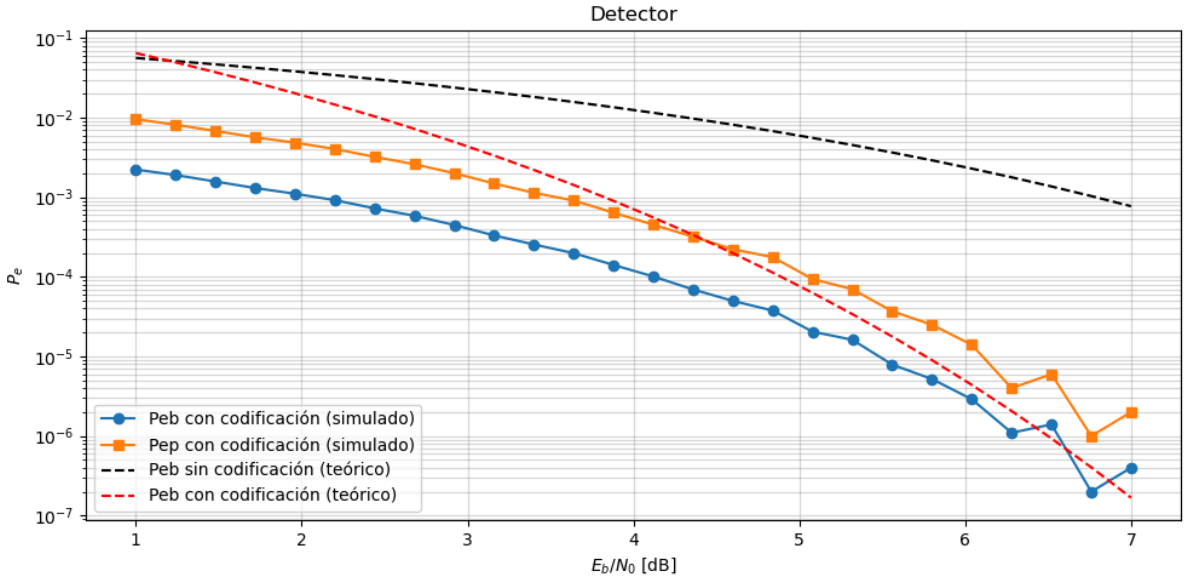

Figure 2.2: Probabilidad de error de bit y palabra en modo detector

Los saltos inesperados en algunas muestras son debido a que:

- Se realizó una única iteración por valor de $\frac{E_b}{N_0}$.
- En algunos casos quizás sea necesario aumentar la cantidad de palabras por valor de $\frac{E_b}{N_0}$.

Estas decisiones fueron para lograr mayor simplicidad y menor complejidad del código. Para realizar una muestra estadística representativa, se debería aumentar la cantidad de palabras y realizar múltiples iteraciones por valor de $\frac{E_b}{N_0}$.

1.4.2 Análisis P_{ep} vs $\frac{E_b}{N_0}$

En ambos modos, la probabilidad de error de palabra refleja la tasa de bloques con al menos un bit erróneo. En el **modo corrector**, la curva de P_{ep} desciende suavemente, reflejando la capacidad del código para corregir errores individuales. La diferencia entre P_{ep} y P_{eb} se amplía a medida que mejora el SNR. En el **modo detector**, se mantiene elevada incluso para valores altos de $\frac{E_b}{N_0}$, ya que el sistema descarta todas las palabras con errores detectados, sin intentar su recuperación.

1.4.3 Ganancia de codificación G_c vs $\frac{E_b}{N_0}$

En la **Figura 2.3** (modo corrector), la ganancia de codificación simulada G_c parte de valores bajos en SNR moderado y crece con el aumento de $\frac{E_b}{N_0}$, estabilizándose en torno a **1.5 dB** a partir de 6 dB, en acuerdo con la ganancia asintótica teórica G_a.

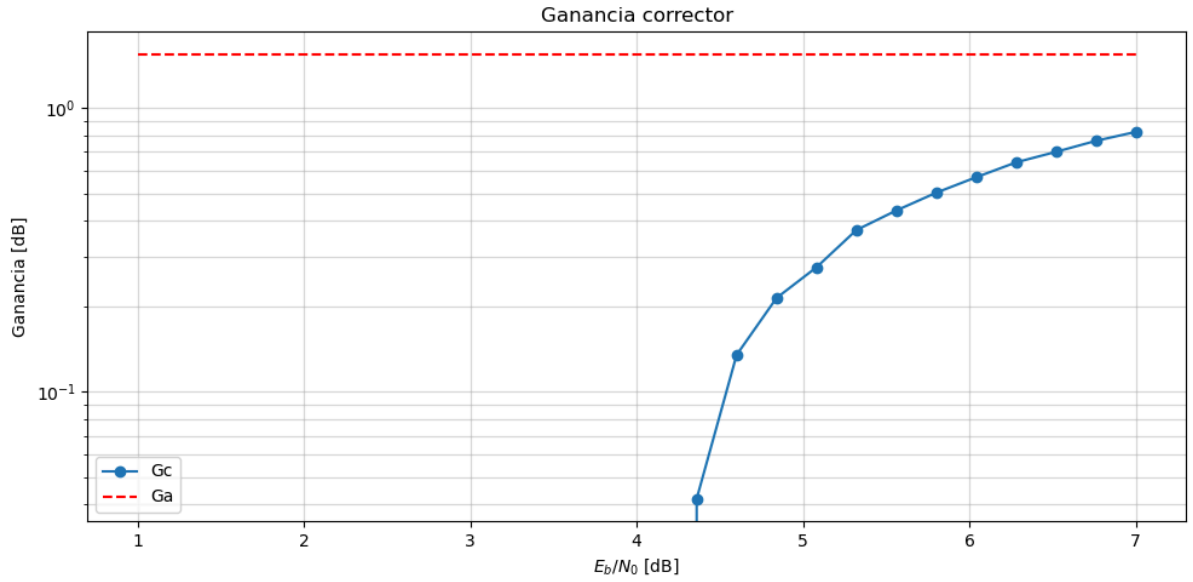

Figure 2.3: Ganancia de codificación modo corrector

En la **Figura 2.4** (modo detector), en cambio, G_c se mantiene alrededor de **4.0–5.0 dB** en todo el rango, muy por encima de G_a. Esta ganancia aparente se debe al sesgo introducido por el descarte de bloques erróneos, que reduce P_{eb} al eliminar las contribuciones de errores más probables.

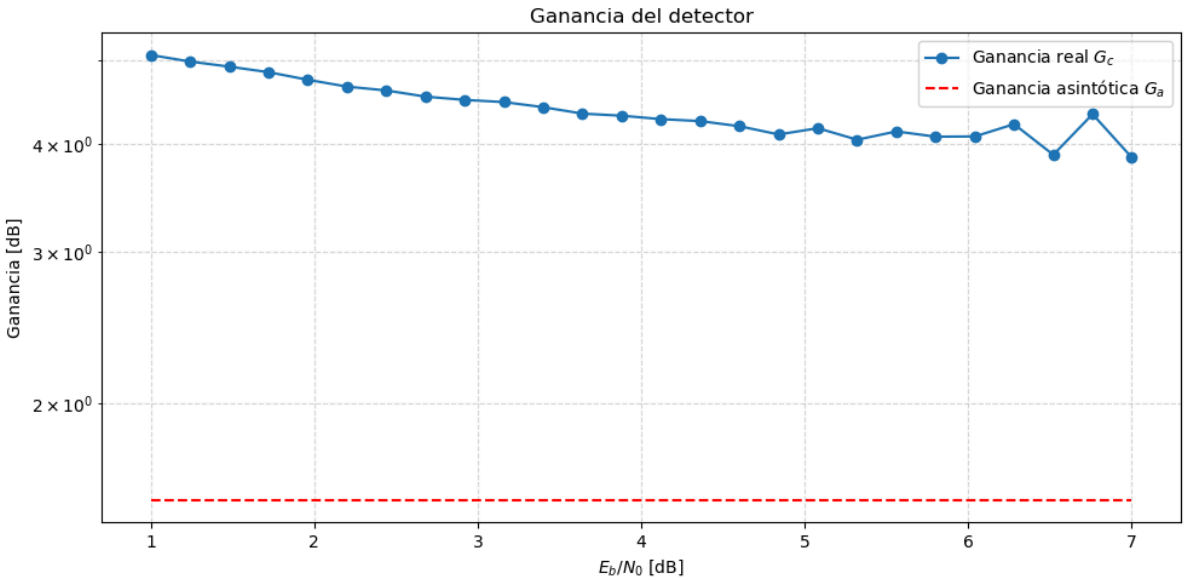

Figure 2.4: Ganancia de codificación modo detector

1.4.4 Tablas de resultados de ejecución

Las tablas los resultados de ejecución se encuentran los archivos `resultados/detector.csv` y `resultados/corrector.csv` para cada modo.

2 Codificación de fuente

2.1 Introducción

Este trabajo tiene como objetivo reducir la cantidad de bits necesarios para representar la imagen `logo_FI.tif`. Para ello, se calcula la frecuencia de aparición de los símbolos de la fuente utilizando distintas longitudes $n = 1, 2, 3$, y se construyen fuentes extendidas de orden $m = 2$ y $m = 3$ utilizando codificación de Huffman. Finalmente, se analizan los resultados comparando entropía, largo promedio y tasa de compresión entre los distintos modelos.

2.2 Marco teórico

2.2.1 Compresión

- **Entropía**: $H = -\sum p_i * \log_2(p_i)$
- **Largo promedio**: $L = \sum p_i * l_i$
- **Tasa de compresión**: $TC = \frac{n*m}{L}$

2.2.2 Codificación de Huffman

Es un algoritmo que sistematiza la idea de asignar las longitudes de las palabras en forma inversa a la probabilidad de cada símbolo y garantiza el mínimo largo promedio. Para realizar la codificaión, se necesita conocer a priori las probabilidades de los símbolos.

2.2.3 Extensión de fuente

A partir de una fuente discreta S, definimos una extensión de la misma, S^n como el conjunto de todas las n-tuplas que se pueden formar con elementos de S.

- Por ej., si S = {0, 1} $\Longrightarrow$ S^2 = {00, 01, 10, 11}.
- Puede demostrarse que $H(S^n) = nH(S)$.

2.3 Procedimiento

2.3.1 Procesamiento de la imagen

La imagen se convierte en escala de grises y se aplica un umbral de valor 128 para binarizar cada píxel como 0 (blanco) o 1 (negro). El resultado es una secuencia lineal de bits correspondiente a la cada fila.

2.3.2 Cálculo de probabilidades

Sobre la secuencia de bits se recorre con una ventana deslizante de longitud n para contar la frecuencia de cada patrón. La probabilidad de un patrón de longitud n se calcula como su frecuencia relativa dentro de la secuencia total.

2.3.3 Extensión de la fuente

Para capturar dependencias de orden superior, se generan las combinaciones de patrones unitarios por producto cartesiano, formando bloques de longitud n multiplicado por el orden m (m=2,3). Asumiendo independencia entre los bits del patrón básico, la probabilidad de cada bloque extendido se obtiene multiplicando las probabilidades individuales de sus componentes.

2.3.4 Construcción y evaluación del código de Huffman

Con la lista de bloques extendidos y sus probabilidades, se construye el árbol de Huffman seleccionando iterativamente los dos símbolos de menor probabilidad. Cada rama izquierda añade un bit 0 y cada rama derecha un bit 1. A continuación, se calculan: H, L y TC.

2.4 Resultados

Para n = 1 se obtuvieron las siguientes probabilidades para cada símbolo:

- 0: 0.514614
- 1: 0.485386

Las probabilidades son muy similares. Por esta razón se puede observar en la **Tabla 2.1** que no es posible comprimir la fuente aún cuando se extiende la fuente a órdenes 2 y 3.

Orden m	Largo promedio [bits]	Entropía [bits/símbolo]	Tasa compresión
2	2.000000	1.998767	1.000000
3	3.000000	2.998151	1.000000

Tabla 2.1: Métricas para $n = 1$

Por esta razón, se decidió realizar procesar la imagen tomando simbolos de tamaño $n = 2$ y $n = 3$. En la **Tabla 2.2** y **2.3** podemos observar la efectiva compresión de la fuente al aumentar la longitud de los símbolos y luego codificar la fuentes extendidas de orden 2 y 3. Esto es debido a que las probabilidades de ocurrencia en patrones de pixeles todos blancos o todos negros es mucho mayor a la de pixeles mixtos.

Por ejemplo las probabilidades para $n = 2$:

- 00: 0.503640
- ...
- 11: 0.474417

y para $n = 3$:

- 000: 0.492799
- ...
- 111: 0.463507

Orden m	Largo promedio [bits]	Entropía [bits/símbolo]	Tasa compresión
2	2.405338	2.303169	1.662968
3	3.480551	3.454753	1.723865

Tabla 2.2: Métricas para $n = 2$

Orden m	Largo promedio [bits]	Entropía [bits/símbolo]	Tasa compresión
2	2.659967	2.607184	2.255667
3	3.957027	3.910777	2.274435

Tabla 2.3: Métricas para $n = 3$

2.5 Conclusiones

La compresión utilizando codificación de Huffman resultó efectiva al considerar símbolos de mayor longitud. Cuando se utilizó la fuente original con símbolos de un solo bit ($n = 1$), no se logró ningún tipo de compresión, ya que las probabilidades de los símbolos eran muy similares.

Sin embargo, al redefinir los símbolos como bloques de 2 y 3 bits ($n = 2$ y $n = 3$) y aplicar la extensión de fuente (órdenes $m = 2$ y $m = 3$), se logró una reducción significativa en el largo promedio, y por lo tanto una mejora en la tasa de compresión. Esto se debe a que la imagen presenta patrones con distinta frecuencia de aparición, predominando secuencias homogéneas de píxeles blancos o negros. Aprovechando esta distribución de probabilidad, estas secuencias son codificadas con longitud más corta.

Finalmente, se observa que la mejora en la compresión comienza a estabilizarse a partir de cierto punto, dado que la entropía de la fuente impone un límite teórico a la eficiencia alcanzable. Esto se refleja en que, al incrementar el orden de extensión de la fuente, las ganancias en la tasa de compresión se vuelven progresivamente menores. Por lo tanto, es importante encontrar un equilibrio entre el orden elegido y la complejidad computacional para obtenerlo.