Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Análisis y Optimización de Algoritmos de Esteganografía en Imágenes Basado en Teoría de la Información

Autor: Alejandro Náger Fernández-Calvo
Tutor: Jesús Martínez Mateo

Madrid, Junio 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática

Título: Análisis y Optimización de Algoritmos de Esteganografía en Imágenes Basado en Teoría de la Información

Junio 2025

Autor: Alejandro Náger Fernández-Calvo
Tutor: Jesús Martínez Mateo
Departamento de Matemática Aplicada a las Tecnologías de la Información y las Comunicaciones
Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

Resumen

En un mundo donde la protección de la información es cada vez más relevante, la esteganografía se presenta como una técnica complementaria a la criptografía, permitiendo ocultar la existencia misma de los datos. En particular, la ocultación de información en imágenes digitales ha cobrado gran importancia debido a la amplia disponibilidad de este tipo de archivos, su fácil difusión y su facilidad para disimular alteraciones sin afectar perceptiblemente la calidad visual.

Este trabajo se centra en el análsis y la optimización de distintos algoritmos de esteganografía en imágenes desde una perspectiva basada en la teoría de la información. En concreto, se parte del algoritmo clásico de inserción en el bit menos significativo, o LSB, por sus siglas en inglés. Este algortimo es ampliamente utilizado por su simplicidad, pero limitado en términos de la cantidad de información capaz de ocultar y su resistencia frente a análisis perceptuales.

El objetivo principal es mejorar este algoritmo utilizando medidas de entropía local para seleccionar las regiones más adecuadas para la ocultación, priorizando aquellas con mayor complejidad visual. Para ello, se ha implementado una herramienta en Python que permite comparar el rendimiento del algoritmo clásico frente a su versión optimizada. Evaluando métricas como la capacidad de ocultación de información y la calidad visual mediante PSNR y SSIM.

Los resultados obtenidos permitirán valorar si el uso de información estructural de la imagen contribuye a una ocultación más eficiente y menos detectable, aportando así avance práctico y medible en el campo de la esteganografía digital.

Abstract

In a world where information protection is becoming increasingly relevant, steganography emerges as a technique that complements cryptography by hiding the very existence of the data. In particular, hiding information within digital images has gained significant importance due to the widespread availability of such files, their ease of distribution, and their ability to conceal alterations without noticeably affecting visual quality.

This work focuses on the analysis and optimization of various image steganography algorithms from an information-theoretic perspective. Specifically, it builds on the classic Least Significant Bit (LSB) insertion algorithm. This algorithm is widely used due to its simplicity but is limited in terms of the amount of information it can hide and its resistance to perceptual analysis.

The main goal is to improve this algorithm by using local entropy measures to select the most suitable regions for data hiding, prioritizing areas with higher visual complexity. To achieve this, a Python tool has been developed to compare the performance of the classic algorithm against its optimized version, evaluating metrics such as data hiding capacity and visual quality using PSNR and SSIM.

The results obtained will help assess whether leveraging structural information from the image contributes to more efficient and less detectable data hiding, thus offering a practical and measurable advancement in the field of digital steganography.

Tabla de contenidos

Capítulo 1

Introducción

1.1. Motivación y necesidad

En la actualidad, el crecimiento exponencial de la información digital y su circulación constante a través de redes públicas hace que la protección de los datos se haya convertido en una prioridad fundamental. Aunque la criptografía garantiza la confidencialidad del contenido, no impide que la existencia del mensaje sea evidente. En este contexto, la esteganografía, cuyo objetivo es ocultar la propia presencia de la información dentro de soportes aparentemente inocuos, adquiere una relevancia cada vez mayor.

Las imágenes digitales, debido a su gran disponibilidad y a su capacidad para tolerar pequeñas modificaciones sin que resulten perceptibles al ojo humano, representan un canal ideal para ocultar datos de forma discreta. Sin embargo, muchos de los algoritmos clásicos, como el método del bit menos significativo (LSB), aunque simples y fáciles de implementar, presentan limitaciones importantes en cuanto a la cantidad de datos que pueden ocultar y la calidad visual resultante.

Este trabajo nace de la necesidad de explorar mejoras sobre estos métodos clásicos utilizando herramientas teóricas y métricas objetivas, con el fin de maximizar la capacidad de ocultación sin comprometer la calidad visual de las imágenes modificadas. Además, se busca ofrecer una solución práctica que pueda servir de base para futuros desarrollos en el campo de la esteganografía digital.

1.2. Objetivos y alcance del proyecto

El objetivo general de este proyecto es analizar y optimizar un algoritmo de esteganografía en imágenes, concretamente el método LSB, para incrementar la cantidad de datos ocultos sin afectar perceptiblemente la calidad de la imagen. Este proyecto se centra exclusivamente en imágenes estáticas en formato estándar (como PNG o BMP), dejando fuera otros medios como vídeo o audio. Asimismo, no se aborda con profundidad el análisis forense ni la detección de

esteganografía (steganalysis), ya que el objetivo se limita a mejorar la calidad y eficiencia de la ocultación, no su resistencia ante ataques.

1.3. Estructura de la memoria

Este documento consta de 5 capítulos incluyendo la introducción. De forma ordenada, a continuación se explicarán brevemente cada uno de los siguientes capítulos.

2. **Estado del arte:** en este capítulo se presenta el marco teórico del trabajo, introduciendo los conceptos fundamentales de la esteganografía digital, la teoría de la información y las métricas utilizadas para evaluar calidad y capacidad del ocultamiento de información en imágenes.
3. **Desarrollo:** aquí se describe en detalle la metodología empleada, incluyendo el diseño e implementación de los algoritmos, así como los criterios de evaluación utilizados.
4. **Análisis de resultados:** en esta parte del documento se exponen los resultados obtenidos en los experimentos realizados, acompañados de un análisis comparativo entre el algoritmo clásico y las propuestas optimizadas.
5. **Conclusiones y trabajo futuro:** se presentan las conclusiones del proyecto, así como posibles líneas de mejora y trabajo futuro.
6. **Análisis de impacto:** finalmente se describen las utilidades y aportaciones del proyecto tanto para la comunidad como para el autor.

Capítulo 2

Estado del arte

2.1. Fundamentos de la esteganografía

2.1.1. Definición y propósito

La esteganografía es la disciplina que estudia las técnicas para ocultar información dentro de un medio aparentemente inocuo, de forma que se mantenga en secreto no solo el contenido del mensaje, sino incluso su existencia. El término proviene del griego steganos (cubierto) y graphein (escribir), y su uso se remonta a la antigüedad, con ejemplos como la escritura oculta con tinta invisible o mensajes grabados bajo capas de cera [1].

A diferencia de la criptografía, que busca proteger el contenido del mensaje ante posibles interceptores, la esteganografía persigue la invisibilidad del canal de comunicación. En este contexto, el objetivo no es tanto impedir la lectura del mensaje como evitar levantar sospechas de su existencia.

En la era digital, la esteganografía se ha convertido en una herramienta poderosa en campos como la ciberseguridad, la protección de derechos de autor, por ejemplo, mediante marcas de agua digitales [2], la comunicación encubierta y, lamentablemente, también en actividades ilícitas. Esto ha despertado el interés tanto en el diseño de técnicas esteganográficas como en su análisis y detección.

2.1.2. Principios de diseño de sistemas esteganográficos

Si analizamos que cualidades debe cumplir un sistema esteganográfico para que sea robusto, podemos deducir que debe cumplir las siguientes cualidades:

- **Imperceptibilidad:** Se refiere a que los cambios introducidos en el medio portador (como una imagen) para ocultar el mensaje no deben ser reconocibles, al menos para el ojo humano. Si el mensaje oculto altera visiblemente la imagen, el objetivo de pasar desapercibido se pierde. Este principio se evalúa comúnmente con métricas objetivas como el PSNR o el SSIM.
- **Capacidad de ocultación:** Es la cantidad de información que puede ocultarse en el medio sin comprometer la imperceptibilidad. Existe una relación

inversa entre capacidad y calidad: a mayor cantidad de información oculta, mayor es el riesgo de introducir distorsiones o artefactos en el medio. Un diseño eficiente busca el equilibrio óptimo entre ambos.

- **Robustez:** Hace referencia a la resistencia del mensaje oculto frente a manipulaciones accidentales o intencionadas del medio (como compresión, recortes, cambios de formato, etc.). Aunque en este trabajo no se abordará la robustez como objetivo principal, su mención es esencial para contextualizar el diseño general de sistemas esteganográficos.

A estos principios, algunos autores como C.Cachin [3] añaden la seguridad esteganográfica, que implica que aunque el atacante conociese el método de ocultación del mensaje, fuese incapaz de distinguir una imagen con información oculta y una sin ella.

2.1.3. Clasificación de técnicas esteganográficas

Las técnicas esteganográficas pueden clasificarse desde distintos enfoques, dependiendo del tipo de medio, del dominio de trabajo, o del método de ocultación. A continuación, se describen las principales clasificaciones:

a) Según el tipo de medio portador

- Texto
- Imágenes
- Audio
- Vídeo
- Tráfico de red (esteganografía en protocolos)
- Otros

Este trabajo se centra exclusivamente en la esteganografía en imágenes, por ser un medio altamente redundante y con alta tolerancia visual, lo que lo hace ideal para ocultar información.

b) Según el dominio de ocultación

- **Dominio espacial:** El mensaje se oculta directamente en los píxeles de la imagen. Técnicas como el algoritmo LSB (Least Significant Bit) son representativas de este enfoque.
- **Dominio transformado:** Se trabaja sobre una representación transformada de la imagen (por ejemplo, mediante DCT, DWT o FFT). Este enfoque es más robusto ante transformaciones, aunque suele ser más complejo de implementar.

c) Según la técnica de ocultación

- **Sustitución directa:** Como en el método LSB, se sustituyen bits menos significativos por los bits del mensaje.

- **Dispersión o propagación:** El mensaje se dispersa a lo largo del medio para aumentar la seguridad y dificultar la detección.
- **Codificación adaptativa:** El medio se analiza para determinar las regiones óptimas donde ocultar información, como las zonas con alta entropía o ruido visual.

d) Según la disponibilidad del mensaje original

- **Esteganografía pura:** No se necesita clave ni información adicional, solo el medio portador.
- **Esteganografía con clave secreta:** Se requiere una clave compartida para ocultar y recuperar el mensaje.
- **Esteganografía con clave pública:** Similar al cifrado asimétrico, aunque menos común en la práctica.

En este trabajo nos enfocaremos en técnicas de sustitución directa en el dominio espacial, particularmente en el algoritmo LSB y en dos versiones mejoradas que utilizan análisis de entropía local para seleccionar regiones óptimas de ocultación.

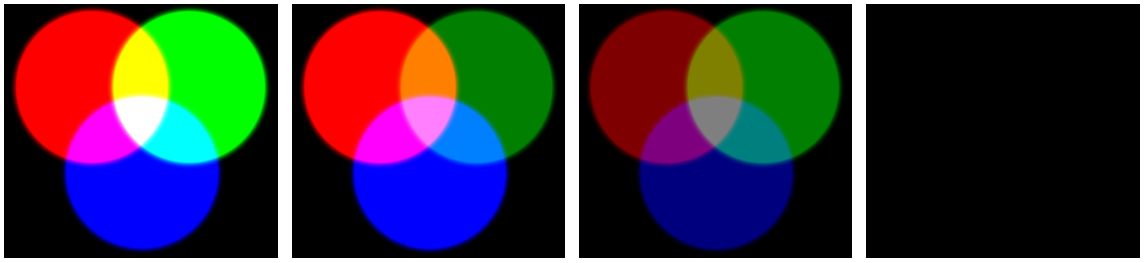
2.2. Esteganografía en imágenes digitales

La esteganografía en imágenes digitales consiste en ocultar información dentro de archivos de imagen. Evitando en medida de lo posible la detección de la información añadida al contenido original. A continuación, se abordan tres aspectos fundamentales que permiten comprender mejor el uso de imágenes como medios portadores en sistemas esteganográficos: las propiedades técnicas de las imágenes, las técnicas clásicas de ocultación, y los criterios para evaluar la calidad visual.

2.2.1. Propiedades de las imágenes digitales

Una imagen digital puede entenderse como una matriz de píxeles, donde cada píxel representa un color. En imágenes en escala de grises, un píxel se representa por un único valor de intensidad (generalmente de 8 bits, entre 0 y 255). En imágenes a color, cada píxel se compone de tres canales: Rojo (R), Verde (G) y Azul (B), también de 8 bits cada uno, lo que da lugar a 24 bits por píxel. Es la combinación de estos 3 canales y sus respectivas intensidades (0-255) lo que permite formar el resto de colores. Por ejemplo, si observamos la Figura 2.1[1] podremos ver como la combinación de diferentes intensidades en estos 3 canales permiten mostrar otros colores.

[1]Imágenes producidas con la herramienta RGB Color Addition de P. Classroom [4]



Blanco Rosa Gris Negro

Figura 2.1: Comparación de la apariencia del píxel según la intensidad de los canales RGB

Las propiedades más relevantes de las imágenes para la esteganografía son:

- **Redundancia espacial:** Las imágenes contienen una gran cantidad de información visual que no es perceptiblemente afectada por pequeñas modificaciones. Esta redundancia es lo que permite ocultar datos sin ser detectados.
- **Representación binaria:** Dado que los valores de los píxeles están codificados en binario, resulta sencillo intervenir directamente en los bits menos significativos de cada componente sin alterar de forma apreciable la imagen.
- **Formato de almacenamiento:** Los algoritmos de esteganografía deben adaptarse al formato de la imagen (BMP, PNG, JPEG, etc.). Algunos formatos como BMP o PNG son más adecuados para la ocultación sin pérdidas, mientras que formatos comprimidos con pérdida como JPEG introducen desafíos adicionales debido a las transformaciones aplicadas (por ejemplo, DCT) [5].

Estas propiedades hacen de las imágenes digitales un entorno ideal para desarrollar sistemas de ocultación simples y eficientes, especialmente en el dominio espacial.

2.2.2. Técnicas clásicas de ocultación en imágenes

Entre las técnicas más utilizadas de esteganografía en imágenes, y en las que se centra este documento, son aquellas basadas en la modificación de los bits menos significativos (LSB, por sus siglas en inglés). Como ya se ha mencionado anteriormente, esta técnica opera en el dominio espacial y es tan popular por su simplicidad y eficacia.

a) Técnica LSB (Least Significant Bit)
El método LSB consiste en reemplazar el bit menos significativo de uno o varios canales de color de cada píxel por los bits del mensaje secreto. Dado que el bit menos significativo tiene un impacto mínimo en el valor total del píxel, la alteración visual es casi imperceptible.

Por ejemplo, un píxel con un valor de intensidad 10110100 (180 en decimal)

podría modificarse a 10110101 (181) para codificar un '1'. Esta modificación representa una variación mínima en la imagen, imperceptible para el ojo humano.

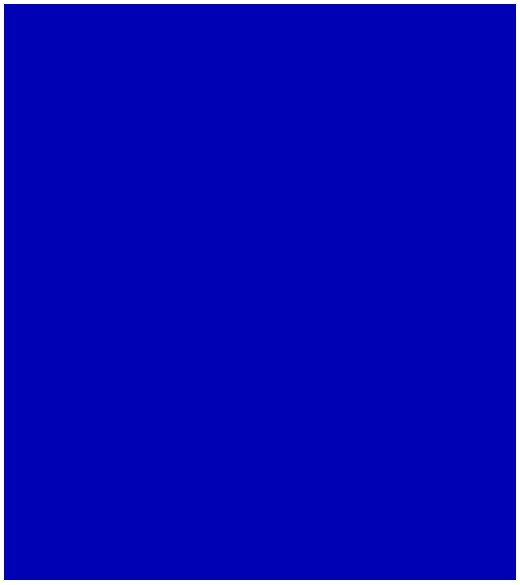


Figura 2.2: Canal azul (B) del RGB con valor 180

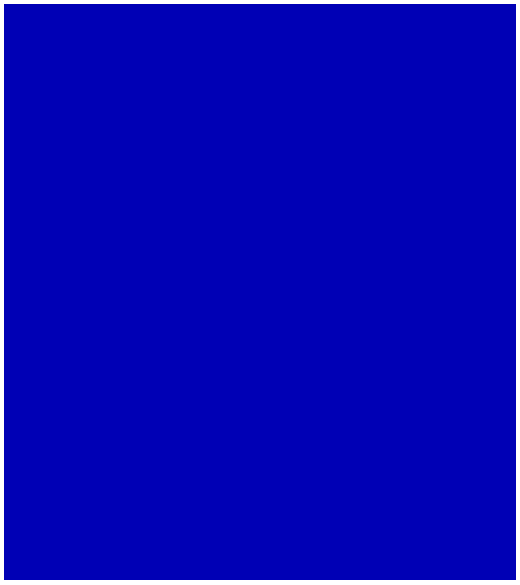


Figura 2.3: Canal azul (B) del RGB con valor 181

Algunas variantes del método LSB incluyen:

- LSB en un solo canal (e.g., azul): se oculta el mensaje solo en el canal azul, que es el menos perceptible para el ojo humano.
- LSB en todos los canales RGB: se reparte la información oculta entre los tres canales para maximizar la capacidad de ocultación del mensaje en la imagen.
- LSB con desplazamiento pseudoaleatorio: la posición de los píxeles modificados se determina mediante una clave secreta, aumentando la seguridad.

A pesar de su eficacia, la técnica LSB es vulnerable a ataques estadísticos, compresión con pérdida, y transformaciones sobre la imagen.

b) Técnicas en el dominio transformado
Aunque este trabajo se centra en el dominio espacial, cabe mencionar las técnicas en el dominio transformado como las que utilizan la Transformada Discreta del Coseno (DCT) o la Transformada Wavelet (DWT). Estas técnicas ocultan la información en los coeficientes de la transformación y son más robustas frente a compresión y manipulación de imágenes, pero requieren mayor complejidad de implementación.

2.3. Fundamentos de teoría de la información aplicados

La teoría de la información, introducida por Claude Shannon en 1948, proporciona un marco teórico para cuantificar la información, la incertidumbre y la

redundancia en sistemas de comunicación [6]. Estos conceptos resultan especialmente útiles cuando se aplican al análisis de imágenes digitales, ya que permiten entender y modelar la distribución de la información visual y, por tanto, optimizar técnicas de ocultación como la esteganografía.

A continuación, se describen tres conceptos clave: la entropía de Shannon, la entropía local y la redundancia en imágenes digitales.

2.3.1. Entropía de Shannon

La entropía de Shannon es una medida de la incertidumbre o impredecibilidad de una fuente de información. En el contexto de imágenes digitales, puede interpretarse como una medida de cuán variada o caótica es la distribución de los niveles de gris o colores en una imagen.

Formalmente, para una variable aleatoria discreta X con posibles valores $x_1, x_2, ..., x_n$ y probabilidades asocaidas $p(x_i)$, la entropía de Shannon se define como:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i) \tag{2.1}$$

En imágenes, X representa los valores de intensidad de los píxeles, y $p(x_i)$ es la probabilidad de que un pixel tenga un valor específico x_i. Por ejemplo, una imagen con una distribución uniforme de niveles de gris tendrá mayor entropía que una imagen con áreas homogéneas.

Una mayor entropía indica mayor cantidad de información, pero también mayor dificultad para detectar alteraciones sutiles. Por esta razón, las regiones con alta entropía son más adecuadas para ocultar información, ya que los cambios en los bits menos significativos son menos detectables.

2.3.2. Entropía local

La entropía local es una extensión del concepto anterior, donde se evalúa la entropía en regiones específicas de la imagen en lugar de considerar toda la imagen como una única distribución. Esto permite identificar zonas donde los píxeles son más impredecibles, como bordes, texturas o detalles.

Se calcula aplicando la fórmula de Shannon a una ventana móvil (por ejemplo, 3x3 o 5x5 píxeles) alrededor de cada píxel. El resultado es un mapa de entropía, donde cada valor refleja la variabilidad de su vecindario.

Este mapa puede utilizarse para guiar algoritmos esteganográficos ya que se puede priorizar la inserción de información oculta en aquellas regiones con mayor entropía local, minimizando la probabilidad de que se perciban artefactos o alteraciones visuales.

Ejemplo: una región de cielo azul (poca variación entre píxeles) tendrá baja entropía local, mientras que una zona con vegetación o textura detallada tendrá alta entropía local.

2.3.3. Información redundante en imágenes

La redundancia es la parte de la información que no aporta novedad, y está presente de forma repetida o predecible en los datos. En imágenes, esto se traduce en regiones homogéneas, patrones repetitivos, o correlación entre píxeles adyacentes.

Desde la teoría de la información, una fuente con alta redundancia tiene baja entropía, y por tanto menor capacidad de ocultación. No obstante, esta redundancia es precisamente lo que permite técnicas de compresión.

En esteganografía, la información redundante puede considerarse una oportunidad para ocultar datos, pero con cuidado: ocultar información en zonas de baja entropía puede generar artefactos visibles. Por ello, los algoritmos que evalúan la redundancia y adaptan dinámicamente dónde insertan los datos ocultos (por ejemplo, según la entropía local) son más eficaces y robustos.

2.4. Métricas de evaluación

La evaluación objetiva del rendimiento de un algoritmo esteganográfico requiere el uso de métricas que permitan analizar principalmente dos factores fundamentales. Por un lado, la cantidad de información que puede ocultarse en una imagen sin ser perceptible, y por otro, el grado de alteración visual que dicha ocultación produce.

En esta sección se describen las principales métricas utilizadas en el campo de la esteganografía, organizadas en torno a los conceptos de capacidad de ocultación y calidad visual percibida.

2.4.1. Capacidad de ocultación

La capacidad de ocultación se refiere a la cantidad de datos que pueden ser insertados en una imagen sin comprometer su calidad visual ni facilitar su detección. Se mide habitualmente en bits por píxel (bpp) o bits totales ocultos.

$$\textit{Capacidad (bpp)} = \frac{\textit{Número total de bits ocultos}}{\textit{Número total de píxeles}} \tag{2.2}$$

Un valor de 1 bpp implica que, en promedio, se ha ocultado 1 bit por cada píxel. Sin embargo, en la práctica, se busca un equilibrio entre capacidad y calidad. Una mayor capacidad de ocultación tiende a generar mayores distorsiones perceptibles.

En el caso del algoritmo LSB clásico, es habitual modificar solo el bit menos significativo de cada canal (en imágenes RGB, por ejemplo), lo que permite una capacidad teórica de hasta 3 bpp si se modifican los tres canales de color. No obstante, las versiones más prudentes emplean 1 bpp o menos para mantener la imperceptibilidad.

El objetivo de este trabajo es optimizar el algoritmo LSB para lograr una mayor capacidad sin afectar negativamente la calidad visual, mediante el uso de métricas como la entropía local para elegir zonas de ocultación óptimas.

2.4.2. Calidad visual percibida

La calidad visual percibida mide cuán perceptible es la distorsión introducida por el proceso de ocultación. Una buena técnica de esteganografía debe garantizar que la imagen modificada (imagen estego) sea visualmente indistinguible de la original.

Las métricas más comunes para evaluar la calidad visual son:

a) PSNR (Peak Signal-to-Noise Ratio)
El PSNR es una medida basada en la diferencia entre los valores de los píxeles de la imagen original y la modificada. Se expresa en decibelios (dB) y cuanto mayor sea el PSNR, menor es la distorsión [7], por tanto, a mayor PSNR mayor será el parecido con la imagen original.

Podemos calcular el valor del PSNR mediante la siguiente fórmula:

$$PSNR = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right) \tag{2.3}$$

donde MAX_I^2 es el valor máximo posible del píxel (por ejemplo, 255 para imágenes de 8 bits) y MSE es el error cuadrático medio:

$$MSE = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} ||I(i,j) - K(i,j)||^2 \tag{2.4}$$

donde I es la imagen original y K la imágen alterada [8].

b) SSIM (Structural Similarity Index Measure)
El SSIM evalúa la similitud estructural entre la imagen original y la esteganografiada. A diferencia del PSNR, tiene en cuenta factores como el contraste, la luminancia y la estructura. Un SSIM con valor de 1 indica una coincidencia perfecta [9].

Haciendo uso de la formula a continuación podemos calcular su valor:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{2.5}$$

donde:

- μ_x, μ_y son las medias de x e y,
- σ_x^2, σ_y^2 son las varianzas,
- σ_{xy} es la covarianza entre x e y,

- $C_1 = (k_1 L)^2$, $C_2 = (k_2 L)^2$ son constantes pequeñas para evitar divisiones por cero,
- L es el rango dinámico de los valores de los píxeles (noramlmente $L = (2^{\textit{bits por píxel}} - 1)$),
- $k_1 = 0{,}01$ y $k_2 = 0{,}03$ por defecto [10].

2.4.3. Conclusiones de evaluación

Para este trabajo se emplearán tanto PSNR como SSIM como métricas complementarias para evaluar la calidad visual de las imágenes modificadas, y la capacidad de ocultación (bpp) para medir la eficiencia del algoritmo. El objetivo es lograr una alta capacidad con mínima pérdida de calidad perceptible, lo cual se considera el ideal en esteganografía.

2.5. Revisión de trabajos relacionados

A la hora de recabar información relevante para este proyecto se han investigado diversas fuentes bibliográficas. En este apartado veremos fuentes académicas más completas, haciendo un breve resumen de cada una para explicar que han aportado a este trabajo.

2.5.1. Trabajos basados en esteganografía digital

2.5.1.1. Esteganografía, Disciplina para ocultar información

El trabajo de la Universidad de Buenos Aires titulado «Esteganografía, Disciplina para ocultar información» por Juan Miguel Sánchez Arteaga [11] explora el mundo de la esteganografía, definiendo cuál es su propósito y se aclarando los términos clave asociados. Se hace un recorrido por su evolución histórica y se presentan distintas formas de clasificación. También se analizan las características de los algoritmos esteganográficos, las distintas técnicas que existen para ocultar datos y los requisitos necesarios para hacerlo. Además, se mencionan algunas herramientas que permiten insertar mensajes ocultos en otros archivos, y se aborda el uso actual de la esteganografía en contextos tanto legales como ilegales. Finalmente, se introduce el concepto de estegoanálisis, que se encarga de detectar si se ha aplicado esteganografía en un archivo. Este trabajo ha sido de valor para conocer conceptos clave y también debido a que trata, aunque de forma breve, temas como esteganografía en imágenes, algoritmos estenográficos y técnicas de dominio espacial y frecuencial con transformadas como DFT, DCT y DWT.

2.5.1.2. Esteganografía en archivos digitales - Digital File Steganography

El trabajo de la Universidad Complutense de Madrid titulado «Esteganografía en archivos digitales - Digital File Steganography» por Alberto Germán Arias Del Cerro [5] se centra enla creación de una aplicación que permite ocultar y extraer mensajes de texto en imágenes utilizando distintos métodos esteganográficos

basados en la modificación del bit menos significativo (LSB). Entre las opciones disponibles se incluyen: el LSB simple con una variante de codificación Elias delta, la técnica de diferencia de valor de píxel (PVD), la modificación de dirección (EMD) y una combinación de los tres métodos (LSB+PVD+EMD). Para cada una de estas técnicas, el autor realiza un ánalisis con el fin de determinar que algorimto genera una mejor ocultación de la información. Gracias a este trabajo se ha indagado en temas relevantes como son los distintos algoritmos y técnicas de análisis de los mismos.

2.5.1.3. Esteganografía en imágenes digitales

El trabajo de la Universidad Politécnica de Madrid (ETSISI) titulado «Esteganografía en imágenes digitales» por Alberto Pampliega Bolaños [12] es semejante a los dos anteriores en el sentido en el que indaga tanto en métodos actuales de esteganografía como en los dominios espaciales y de transformadas, por lo que podríamos considerarlo un punto intermedio entre ambos. Algunas de las aportaciones que este trabajo incluye y los otros no es su manera de comprobar la fidelidad de los algoritmos estenográficos. El trabajo revisa diferentes herramientas con las cuales atacar el medio con la información o mensaje, de modo que se pone a prueba la resistencia de los distintos algoritmos. Este trabajo ha ayudado a consolidar conceptos y ha sido de gran utilidad para la parte práctica o de desarrollo de la siguiente sección 3.

2.5.2. Trabajos basados en teoría de la información

2.5.2.1. Teoría de la Información y la Codificación

La información recogida para la parte de teoría de la información de este trabajo viene mayoritariamente del temario impartido en la asignatura «Teoría de la Información y la Codificación» de la Universidad Politécnica de Madrid (ETSIINF) impartido por el profesor Jesús Martinez Mateo [13]. De esta asignatura quiero destacar la relevancia para este trabajo temas como las nociones básicas de la información y técinicas de transmisión de información en canales sin ruido. Es gracias a estos conocimientos que en la sección de desarollo 3 de este trabajo lograremos una comprensión óptima de nuestro mensaje, ocupando menos espacio a la hora de ocultar la información y produciendo menos distorsiones respecto a la imagen original.

2.6. Consideraciones de implementación

2.6.1. Lenguaje y entorno de desarrollo

Para la implementación del sistema se ha utilizado el lenguaje de programación Python, debido a su simplicidad, amplia comunidad, y las múltiples bibliotecas disponibles para el tratamiento de imágenes y análisis matemático. Entre los paquetes utilizados destacan:

- **Pillow** para carga y manipulación de imágenes.

- **NumPy** para operaciones matriciales y cálculos de entropía.
- **scikit-image** para el cálculo de métricas como SSIM.
- **matplotlib** para visualización de resultados.
- **Tkinter** para la creación de una interfaz gráfica simple.

El desarrollo se ha realizado sobre un entorno VS Code [14] y scripts en Python 3.13 [15] en un sistema operativo Ubuntu 22.04 [16].

2.6.2. Estructura general del sistema

El sistema desarrollado se divide en tres módulos principales:

- **Módulo de ocultación:** permite insertar un mensaje en una imagen utilizando el algoritmo LSB clásico o la versión optimizada.
- **Módulo de extracción:** recupera el mensaje oculto desde una imagen modificada.
- **Módulo de evaluación:** calcula las métricas de capacidad y calidad visual para comparar el rendimiento entre algoritmos.

2.6.3. Representación de imágenes y manipulación de píxeles

Las imágenes utilizadas se convierten internamente a matrices para facilitar el acceso a los valores RGB de cada píxel. Para evitar pérdida de información, se trabajan imágenes en formato PNG o BMP (sin compresión con pérdida).

En el caso del algoritmo LSB, se modifican los bits menos significativos del uno o varios canales para insertar los bits del mensaje.

2.6.4. Implementación del algoritmo LSB clásico

El algoritmo LSB se ha implementado como una función que toma una imagen y un mensaje como entrada, convierte el mensaje a binario, y va sustituyendo los bits menos significativos de los píxeles de la imagen original.

El proceso inverso de extracción recupera los bits de los píxeles modificados y reconstruye el mensaje original.

2.6.5. Mejora basada en entropía local

La mejora propuesta se basa en el cálculo de la entropía local en ventanas deslizantes sobre la imagen. Para cada región, se estima su variabilidad local utilizando la fórmula de entropía de Shannon.

Las regiones con mayor entropía son seleccionadas como candidatas prioritarias para ocultar datos, ya que las alteraciones en estas zonas tienden a ser menos perceptibles al ojo humano.

Esto permite aumentar la cantidad de datos ocultos manteniendo la calidad visual más alta, especialmente en imágenes con texturas o ruido visual.

2.6.6. Interfaz de usuario y pruebas

Se ha desarrollado una interfaz simple (CLI o GUI) para facilitar el uso del sistema. El usuario puede seleccionar una imagen, introducir un mensaje, elegir el método de ocultación (clásico u optimizado) y visualizar los resultados de calidad y capacidad tras la inserción.

Para validar el funcionamiento, se han realizado pruebas con imágenes de distintas resoluciones y características visuales, evaluando los resultados mediante PSNR, SSIM y la capacidad en bits por píxel.

2.6.7. Limitaciones y decisiones técnicas

- El sistema está optimizado para imágenes en escala de grises o RGB sin compresión.
- No se ha implementado cifrado del mensaje oculto, aunque podría añadirse como mejora ya la encriptación de un mensaje es independiente de su ocultación.
- La eficiencia en tiempo no ha sido prioritaria, dado que se trabaja con imágenes de tamaño moderado.
- Se ha evitado el uso de bibliotecas externas para esteganografía para garantizar el control total sobre la lógica del algoritmo.

Capítulo 3

Desarrollo

A lo largo de este capítulo se explicará tanto el entorno de desarrollo como las técnicas aplicadas para la implementación de los distintos algoritmos estenográficos. El código de las implemenrtaciones se encuentra en un repositorio github para mejorar su control de versiones [17].

3.1. Análisis de requisitos

En esta sección se detallan los requisitos funcionales y no funcionales que guían el desarrollo del proyecto. El objetivo principal es implementar y evaluar un sistema de esteganografía en imágenes, optimizado mediante técnicas basadas en la teoría de la información.

3.1.1. Requisitos funcionales

Los requisitos funcionales definen las funcionalidades clave que debe proporcionar el sistema. Entre ellos se encuentran:

- Permitir la codificación de mensajes de texto dentro de imágenes utilizando el método LSB clásico.
- Implementar una versión optimizada del algoritmo LSB basada en entropía local, adaptando dinámicamente el número de bits ocultos por píxel.
- Soportar la decodificación exacta de los mensajes ocultos, asegurando su integridad.
- Evaluar cuantitativamente la calidad visual de las imágenes esteganografiadas mediante métricas como PSNR y SSIM.
- Automatizar pruebas sobre múltiples imágenes y mensajes, generando estadísticas y visualizaciones comparativas.
- Ofrecer una interfaz principal (CLI) para codificar, decodificar y evaluar imágenes de forma sencilla.

3.1.2. Requisitos no funcionales

Los requisitos no funcionales definen propiedades generales del sistema y restricciones del entorno de desarrollo:

- El sistema debe estar implementado en el lenguaje Python, por su simplicidad y amplio soporte de bibliotecas científicas.
- La ejecución debe ser reproducible en distintos entornos operativos (Linux, Windows) mediante un entorno virtual y un archivo `requirements.txt`.
- La estructura del proyecto debe ser modular, separando claramente los algoritmos, métricas, interfaz y utilidades.
- El sistema debe permitir procesar imágenes con resoluciones de al menos 512×512 píxeles sin tiempos de espera significativos.
- Todo el código debe estar cubierto por pruebas automáticas unitarias para garantizar la robustez de la implementación.
- El código debe seguir buenas prácticas de estilo, incluyendo documentación en funciones clave.

3.2. Tecnologías y herramientas empleadas

Para el desarrollo del sistema de esteganografía propuesto se han empleado diversas tecnologías, herramientas y bibliotecas, tanto a nivel de programación como de análisis de imágenes y gestión del proyecto. A continuación se describen las más relevantes.

- **Python 3**: Lenguaje de programación principal del proyecto, elegido por su claridad sintáctica, versatilidad y la disponibilidad de bibliotecas especializadas para el procesamiento de imágenes y análisis numérico.
- **NumPy**: Biblioteca fundamental para el cálculo numérico con matrices. Se ha utilizado para representar y manipular las imágenes como arrays, facilitando la implementación de los algoritmos de codificación y cálculo de entropía.
- **Pillow (PIL)**: Biblioteca para la manipulación de imágenes. Ha permitido cargar, modificar y guardar imágenes en distintos formatos.
- **scikit-image**: Biblioteca especializada en procesamiento de imágenes. Se ha utilizado en el cálculo de métricas de calidad como SSIM y en operaciones de entropía local.
- **Matplotlib y Seaborn**: Herramientas de visualización empleadas para generar gráficos comparativos, mapas de calor y figuras utilizadas en la memoria del proyecto.
- **PyTest**: Framework de pruebas unitarias utilizado para verificar el correcto funcionamiento de los algoritmos, asegurar la cobertura de casos límite y validar los resultados durante el desarrollo.

- **Git y GitHub**: Sistema de control de versiones empleado para gestionar la evolución del código, coordinar cambios y mantener un historial completo del desarrollo.
- **VSCode + extensiones**: Entorno de desarrollo integrado empleado para escribir, depurar y organizar el código. Las extensiones de Python, Markdown, y Git han sido claves para una integración fluida.

Estas herramientas han permitido un flujo de trabajo eficiente, facilitando la implementación modular del sistema, la validación automatizada y la generación de resultados reproducibles.

3.3. Estructura del proyecto

El proyecto ha sido desarrollado siguiendo una estructura modular, facilitando su mantenimiento, comprensión y escalabilidad. A continuación se describe el árbol de directorios y la función de cada uno de sus componentes principales:

```
stegano_project/
├── stegano/
│   ├── lsb.py
│   ├── lsb_optimized.py
│   ├── lsb_adaptive.py
│   ├── entropy.py
│   ├── metrics.py
│   └── utils.py
├── data/
│   ├── input/
│   │   ├── image1.png
│   │   └── ...
│   ├── messages/
│   │   ├── msg1.txt
│   │   └── ...
│   ├── output/
│   │   ├── lsb_image1.png
│   │   ├── lsbopt_image1.png
│   │   ├── lsbadapt_image1.png
│   │   └── ...
│   └── results/
│       ├── results.csv
│       └── ...
├── tests/
│   ├── test_lsb.py
│   └── ...
├── main.py
├── requirements.txt
└── README.md
```

A continuación, se resumen los principales directorios y ficheros:

- **stegano**: Contiene todos los módulos fuente del proyecto. Cada archivo implementa una parte fundamental del sistema:
  - lsb.py: Implementación de la codificación y decodificación LSB clásica.
  - lsb_optimized.py: Versión optimizada del algoritmo LSB basada en entropía local (mejor estenograficamente).
  - lsb_adaptive.py: Versión adaptativa del algoritmo LSB basada en entropía local (mejor compresión del mensaje).
  - entropy.py: Cálculo de la entropía local para un conjunto de píxeles de la imagen.
  - metrics.py: Funciones para el cálculo y generación de métricas de evaluación.
  - utils.py: Utilidades comunes para conversión de texto a bits, bits a texto, delimitadores y carga de archivos.
- **data**: Carpeta de datos dividida en:
  - input: Imágenes originales utilizadas para realizar pruebas.
  - messages: Directorio con archivos de texto que contienen los mensajes a ocultar.
  - output: Imágenes generadas tras la esteganografía de los algoritmos, usadas para evaluación.
  - results: Resultados cuantitativos tras el análisis y comparación de cada imagen generada con su original (incluye figuras como mapas de diferencias y gráficas comparativas).
- **tests**: Contiene pruebas automatizadas para verificar el correcto funcionamiento de cada componente del sistema.
- **main.py**: Script principal para ejecutar el sistema desde línea de comandos (CLI), permitiendo codificar, decodificar y evaluar imágenes de forma sencilla.
- **requirements.txt**: Lista de dependencias necesarias para ejecutar el proyecto en un entorno virtual.
- **README.md**: Documento con instrucciones generales sobre el uso y ejecución del proyecto.

Esta organización permite una separación clara entre algoritmos, datos, pruebas, visualización y lógica de ejecución.

3.4. Implementación de los algoritmos

Esta sección explica de forma detallada los algoritmos estenográficos implementados. De forma razonada se irán complementando los aspectos más teóricos con su correspondiente implementación en código. Antes de introducir el primer algoritmo es conveniente entender aspectos comunes a todos ellos, por tanto, a continuación en el subapartado 3.4.1 se exponen los principios seguidos para el procesamiento de las imágenes en lo referente a este trabajo.

3.4.1. Procesamiento de imágenes

El procesamiento de imágenes es una etapa fundamental en este proyecto, en este apartado se explican los principios empleados para lograr las técnicas de esteganografía sobre imágenes digitales. Una imagen digital en color está formada por una matriz tridimensional de píxeles, donde cada píxel contiene tres valores enteros correspondientes a los canales rojo (R), verde (G) y azul (B). Estos valores se representan comunmente con 8 bits por canal, lo que permite un rango de intensidad de 0 a 255 para cada color.

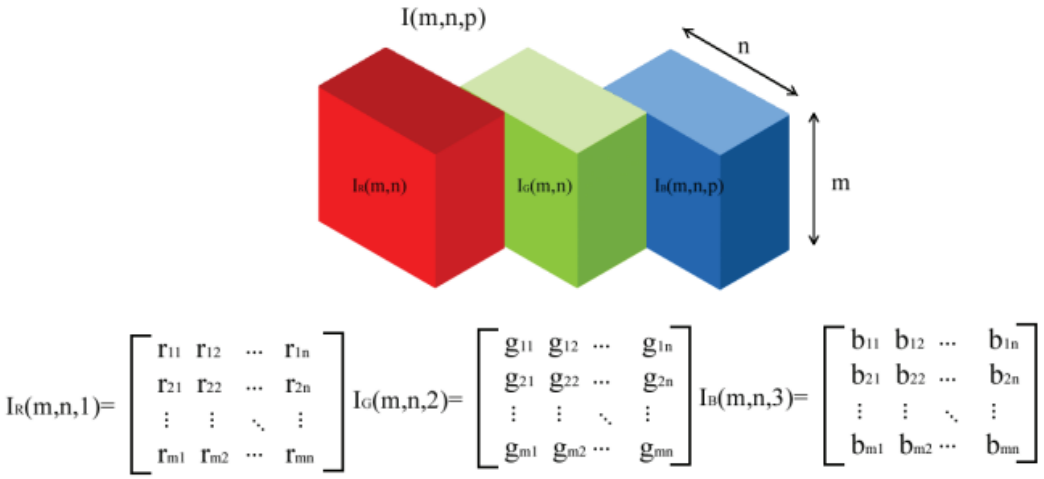


Figura 3.1: Representación matricial canales RGB

Antes de aplicar cualquier algoritmo de esteganografía, las imágenes son convertidas al espacio de color RGB y transformadas en una matriz. Esta conversión permite acceder y modificar individualmente los bits de cada canal de color, lo cual es esencial para la técnica LSB (Least Significant Bit).

Para los métodos optimizados basados en entropía local, se realiza un procesamiento adicional sobre la imagen en escala de grises. Con esto se busca estimar la cantidad de información o variabilidad en una región de la imagen, lo que resulta útil para determinar las zonas más adecuadas para ocultar datos sin comprometer la perceptibilidad visual.

3.4.2. Codificación LSB clásica

El método *Least Significant Bit* (LSB) es una de las técnicas más sencillas de esteganografía en la cual se sustituyen los bits menos significativos de cada canal de color de los píxeles por los bits del mensaje. Dado que el LSB sólo altera ligeramente el valor numérico del canal, la distorsión visual es prácticamente imperceptible.

Fundamento teórico. Sea un píxel con componentes de color R, G, B, cada una en el rango $[0, 255]$. En binario, cada componente tiene 8 bits:

$$R = r_7 r_6 \dots r_1 r_0, \quad G = g_7 g_6 \dots g_1 g_0, \quad B = b_7 b_6 \dots b_1 b_0,$$

donde r_0, g_0, b_0 son los bits menos significativos (LSB). Para ocultar un bit de mensaje $m \in \{0, 1\}$ en, por ejemplo, el canal R, se aplica:

$$R' = (R \wedge 1111\,1110_2) \vee (m),$$

es decir, se limpia el LSB con `AND` y luego se inserta m con `OR`.

Repitiendo el proceso para los bits del mensaje y los canales (R, G, B) de cada píxel, podemos ocultar hasta 3 bits por píxel.

Implementación en Python. A continuación se muestra la función `encode_lsb` que implementa exactamente este procedimiento, y su contraparte `decode_lsb` para recuperar el mensaje.

Listing 3.1: Codificación LSB clásica (`stegano/lsb.py`)

```python
from PIL import Image
from stegano.utils import text_to_bits, bits_to_text,
 add_delimiter, remove_delimiter

def encode_lsb(image_path: str, message: str, output_path: str,
 delimiter: str = "#####") -> None:
  """Oculta un mensaje en la imagen utilizando LSB y guarda el
 resultado."""
  # 1. Cargar y convertir a RGB
  image = Image.open(image_path).convert("RGB")
  pixels = list(image.getdata())

  # 2. Transformar mensaje a bits y añadir delimitador
  binary_message = text_to_bits(add_delimiter(message, delimiter)
 )
  message_len = len(binary_message)

  # 3. Comprobar capacidad: 3 bits por píxel
  if message_len > len(pixels) * 3:
    raise ValueError("El mensaje es demasiado largo para esta
 imagen.")

  # 4. Inserción bit a bit en los LSB de R, G y B
  new_pixels = []
```

```
    bit_index = 0
    for r, g, b in pixels:
      if bit_index < message_len:
        r = (r & ~1) | int(binary_message[bit_index])
        bit_index += 1
      if bit_index < message_len:
        g = (g & ~1) | int(binary_message[bit_index])
        bit_index += 1
      if bit_index < message_len:
        b = (b & ~1) | int(binary_message[bit_index])
        bit_index += 1
      new_pixels.append((r, g, b))

    # 5. Guardar imagen estego
    image.putdata(new_pixels)
    image.save(output_path)
```

Listing 3.2: Decodificación LSB clásica (`stegano/lsb.py`)

```
def decode_lsb(image_path: str, delimiter: str = "#####") -> str:
    """Extrae un mensaje oculto de la imagen utilizando LSB."""
    image = Image.open(image_path).convert("RGB")
    pixels = list(image.getdata())

    # 1. Leer los LSB de cada canal
    bits = ""
    for r, g, b in pixels:
      bits += str(r & 1) + str(g & 1) + str(b & 1)

    # 2. Reconstruir texto y eliminar delimitador
    message = bits_to_text(bits)
    return remove_delimiter(message, delimiter)
```

Comentario de la implementación:

- El código emplea las funciones auxiliares `text_to_bits` y `bits_to_text` para transformar el mensaje entre texto y una secuencia de bits, facilitando la manipulación bit a bit dentro de los píxeles, Pueden consultarse en A.2.
- Para señalar el final del mensaje incrustado, se añade un delimitador fijo (`"#####"`). Esto permite a la función de decodificación identificar con precisión cuándo debe dejar de leer los bits, sin necesidad de almacenar la longitud del mensaje.
- La operación `& ~1` limpia (pone a 0) el bit menos significativo (LSB) del canal correspondiente, y luego se inserta el nuevo bit mediante `| bit`.
- La complejidad del algoritmo es lineal respecto al número de píxeles, es decir, $O(N)$, ya que se recorren una vez para codificar o decodificar.
- Este enfoque no distingue entre regiones texturizadas u homogéneas, mo-

difica los píxeles de forma uniforme en toda la imagen. Esto puede generar artefactos visibles si se ocultan demasiados datos en zonas planas.

Esta técnica, aunque simple, sienta las bases de las versiones optimizadas basadas en entropía local. Este algoritmo modifica los píxeles de forma uniforme en toda la imagen lo que puede generar artefactos visibles si se ocultan demasiados datos en zonas planas, lo que justifica el uso de variantes optimizadas que consideren la entropía local.

3.4.3. Codificación LSB optimizada por entropía

Como ya se ha explicado, mediante LSB clásico se inserta información en el bit menos significativo de cada canal RGB de todos los píxeles, sin distinguir zonas homogéneas de texturizadas. Esto puede generar artefactos visibles en regiones planas. Para mitigarlo, se ha aprovechado la *"entropía local"* de la imagen, que evalua la variabilidad de un conjunto de píxeles. Al ocultar datos solo donde la entropía es alta, se mejora la imperceptibilidad ya que las alteraciones de color se producen en zonas de la imagen donde ya había un alto contraste.

En este trabajo se han desarrollado dos algoritmos LSB basados en la entropía local de una imagen:

- **Selectiva**: inserta 1 bit por canal solo en píxeles con alta entropía.
- **Adaptativa**: asigna 0, 1 o 2 bits por canal según tres rangos de entropía.

3.4.3.1. Cálculo de entropía local

Convertimos la imagen a escala de grises y, para cada píxel, calculamos la entropía de Shannon en una ventana $w \times w$. Esto devuelve un mapa de valores en [0, $\log_2(w^2)$], que luego normalizamos a $[0, 1]$.

Listing 3.3: Cálculo y normalización de entropía local (`stegano/entropy.py`)

```python
from skimage.filters.rank import entropy as sk_entropy
from skimage.morphology import square
import numpy as np
from PIL import Image

def compute_entropy_map(image: Image.Image, window_size: int = 3)
    -> np.ndarray:
    gray = image.convert("L")
    arr  = np.array(gray)
    ent  = sk_entropy(arr, footprint=square(window_size))
    return ent.astype(np.float32)

def normalize_entropy_map(entropy_map: np.ndarray) -> np.ndarray:
    mn, mx = entropy_map.min(), entropy_map.max()
    if mx == mn:
    return np.zeros_like(entropy_map)
    return (entropy_map - mn) / (mx - mn)
```

3.4.3.2. Codificación LSB optimizada para esteganografía

Este algoritmo analizará la entropía de la imagen en ventanas de 3×3 píxeles, creando en el proceso un mapa de calor. Este mapa de calor señala las regiones de la imágen donde hay mayor diferencia de intensidad, permitiendonos seleccionar esas zonas y modificarlas, reduciendo así la aparición de aparatos o cambios bruscos en zonas más perceptibles de la imagen. Para seleccionar estas zonas de alta entropía en el mapa de calor se aplica un umbral τ, de modo que sólo quellos píxeles con entropía normalizada $\geq \tau$ guardan bits. El siguiente código inserta, en orden, 1 bit por canal hasta agotar el mensaje o los píxeles disponibles.

Listing 3.4: Codificación LSB optimizada (`stegano/lsb_optimized.py`)

```
def encode_lsb_optimized(image_path: str,
             message: str,
             output_path: str,
             window_size: int = 3,
             threshold: float = 0.5,
             delimiter: str = "#####") -> None:

  # 1. Carga y preparación
  img = Image.open(image_path).convert("RGB")
  arr = np.array(img)
  ent_norm = normalize_entropy_map(compute_entropy_map(img,
 window_size))
  mask_pos = np.argwhere(ent_norm >= threshold)

  # 2. Comprobar capacidad
  bits = text_to_bits(add_delimiter(message, delimiter))
  total_bits = len(bits)
  if total_bits > len(mask_positions) * 3:
    raise ValueError("El mensaje es demasiado largo para la
 capacidad disponible.")

  # 3. Embedding LSB en píxeles seleccionados
  stego = arr.copy()
  bit_idx = 0
  for y, x in mask_positions:
    if bit_idx >= total_bits: break
    for c in range(3):
      if bit_idx < total_bits:
        stego[y, x, c] = (stego[y, x, c] & 0xFE) | int(bits[
 bit_idx])
        bit_idx += 1

  # 4. Guardar imagen estego
  Image.fromarray(stego).save(output_path)
```

Y su contraparte a continuación, se encarga de extraer el mensaje codificado de una imágen ya modificada:

Listing 3.5: Decodificación LSB optimizada (`stegano/lsb_optimized.py`)

```python
def decode_lsb_optimized(image_path: str,
                         window_size: int = 3,
                         threshold: float = 0.5,
                         delimiter: str = "#####") -> str:

  # 1. Cargar imagenes estego y original
  stego_img = Image.open(image_path).convert("RGB")
  stego_arr = np.array(stego_img)

  # 2. Derivar ruta de la imagen original a partir del nombre de archivo
  base = os.path.basename(image_path)
  if base.startswith("lsbopt_"):
    orig_name = base[len("lsbopt_"):]
  elif base.startswith("lsb_"):
    orig_name = base[len("lsb_"):]
  else:
    orig_name = base
  orig_path = os.path.join("data", "input", orig_name)
  orig_img = Image.open(orig_path).convert("RGB")

  # 3. Reconstrucción de la máscara de entropía
  ent_norm = normalize_entropy_map(compute_entropy_map(orig_img, window_size))
  mask_positions = np.argwhere(ent_norm >= threshold)

  # 4. Extracción bit a bit con parada en delimitador
  bits = ""
  message = ""
  for y, x in mask_positions:
    for c in range(3):
      bits += str(int(stego_arr[y, x, c]) & 1)
      if len(bits) >= 8:
        byte = bits[:8]
        bits = bits[8:]
        char = chr(int(byte, 2))
        message += char
        if message.endswith(delimiter):
          return message[:-len(delimiter)]

  # 5. Si nunca vimos el delimitador, devolvemos todo lo leído
  return message
```

El código mantiene el planteamiento de LSB clásico, con la diferencia que esta vez somos más cuidadosos con los píxeles modificados, disminuyendo así la detectabilidad de cambios en la imagen. Sin embargo, esto conlleva una deficiencia clara frente al algorítmo clásico. Al priorizar la detectabilidad, se ha reducido la longitud máxima posible del mensaje a ocultar, pues recordemos que ahora ocultamos bits del mensaje tan sólo en aquellos píxeles que cumplan $\geq \tau$.

3.4.3.3. Codificación LSB adaptativa

El algoritmo anterior, sacrifica la longitus máxima del mensaje a olcultar a cambio de reducir la detectabildad. Para sufrir esta limitación y maximizar la capacidad de ocultación sin aumentar en gran medida la distorsión, asignamos $\{0, 1, 2\}$ bits por canal según tres niveles de entropía:

$$\begin{cases} < t_0 : & 0 \text{ bits/canal} \\ t_0 \leq e < t_1 : & 1 \text{ bit/canal} \\ \geq t_1 : & 2 \text{ bits/canal} \end{cases}$$

De esta manera en aquellos píxeles con mayor entropía se introducirán más bits modificados.

Listing 3.6: Codificación LSB adaptativa (`stegano/lsb_adaptive.py`)

```
def encode_lsb_adaptive(image_path: str,
            message: str,
            output_path: str,
            window_size: int = 3,
            thresholds: tuple[float,float] = (0.5, 0.75),
            bits_per_channel: tuple[int,int,int] = (0,1,2),
            delimiter: str = "#####") -> None:

  # 1. Cargar la imagenes
  img = Image.open(image_path).convert("RGB")
  arr = np.array(img)
  h, w, _ = arr.shape

  # 2. Calc. y normalizar mapa de entropía local sobre imagen original
  ent_norm = normalize_entropy_map(compute_entropy_map(img, window_size))

  # 3. Extraer los umbrales y configuración de bits por canal
  t0, t1 = thresholds
  b0, b1, b2 = bits_per_channel

  # 4. Generar mapa de bits por canal
  chan_slots = np.zeros((h, w), dtype=int)
  chan_slots[ent_norm < t0] = b0
  chan_slots[(ent_norm >= t0) & (ent_norm < t1)] = b1
  chan_slots[ent_norm >= t1] = b2

  # 5. Preparar el mensaje para su inserción
  bits = text_to_bits(add_delimiter(message, delimiter))
  total_bits = len(bits)
  capacity = int(np.sum(chan_slots) * 3) # Capacidad total de bits (3 canales por canal)

  # 6. Comprobar capacidad
  if total_bits > capacity:
```

```
34        raise ValueError(f"Mensaje de {total_bits} bits supera
      capacidad de {capacity} bits.")
35
36      # 7. Crear copia de la imagen para modificar
37      stego = arr.copy()
38      bit_idx = 0
39
40      # 8. Inserción de bits canal por canal
41      ys, xs = np.nonzero(chan_slots > 0)
42      positions = sorted(zip(ys, xs), key=lambda p: chan_slots[p[0],
      p[1]], reverse=True)
43
44      for y, x in positions:
45        bpc = chan_slots[y, x]   # Bits por canal en este píxel
46        for channel in range(3): # Iterar sobre canales R, G, B
47          for k in range(bpc):   # Insertar bits menos significativos
48            if bit_idx >= total_bits:
49              break
50            # Limpiza el bit k
51            mask = (~(1 << k)) & 0xFF
52            orig_val = int(stego[y, x, channel])
53            # Construir nuevo bit
54            new_bit = (int(bits[bit_idx]) & 1) << k
55            # Sustituir bit k
56            stego[y, x, channel] = (orig_val & mask) | new_bit
57            bit_idx += 1
58          if bit_idx >= total_bits:
59            break
60        if bit_idx >= total_bits:
61          break
62
63      # 9. Guardar imagen estego
64      Image.fromarray(stego.astype(np.uint8)).save(output_path)
```

Teóricamente este algoritmo es el que mayor capacidad de ocultación tiene de los 3 algoritmos presentados en este trabajo, ya que en caso de que todos los píxeles de una imagen superen el primer umbral t_1, este algorítmo sería capaz de ocultar hasta el doble de información que el segundo alortimo con mayor capacidad de ocultación, siendo el LSB clásico.

A continuación y de forma similar al algoritmo optimizado 3.4.3.2 se decodifica el mensaje a partir de la imagen modificada:

Listing 3.7: Decodificación LSB adaptativa (`stegano/lsb_adaptive.py`)

```
1   def decode_lsb_adaptive(image_path: str,
2               original_path: str,
3               window_size: int = 3,
4               thresholds: tuple[float,float] = (0.5, 0.75),
5               bits_per_channel: tuple[int,int,int] = (0,1,2),
6               delimiter: str = "#####") -> str:
7
```

```
    # 1. Cargar imagenes estego y original
    stego_img = Image.open(image_path).convert("RGB")
    stego = np.array(stego_img)
    orig_img = Image.open(original_path).convert("RGB")

    # 2. Calc. y normalizar mapa de entropía local sobre imagen original
    ent_norm = normalize_entropy_map(compute_entropy_map(orig_img, window_size))

    # 3. Extraer los umbrales y configuración de bits por canal
    t0, t1 = thresholds
    b0, b1, b2 = bits_per_channel

    # 4. Mapa indicando n bits por canal extraer en cada píxel
    h, w, _ = stego.shape
    chan_slots = np.zeros((h, w), dtype=int)

    # 5. Asignar bits por canal según el valor de entropía local
    chan_slots[ent_norm < t0] = b0
    chan_slots[(ent_norm >= t0) & (ent_norm < t1)] = b1
    chan_slots[ent_norm >= t1] = b2

    # 6. Coordenadas de píxeles con al menos 1 bit disponible
    ys, xs = np.nonzero(chan_slots > 0)

    # 7. Ordenar las posiciones por cantidad de bits disponibles (de mayor a menor)
    positions = sorted(zip(ys, xs), key=lambda p: chan_slots[p[0], p[1]], reverse=True)

    # 8. Extraer bits de la imagen estego
    bits = ""
    message = ""
    for y, x in positions:
      bpc = chan_slots[y, x]   # Bits por canal en este píxel
      for channel in range(3): # Iterar sobre canales R, G, B
        for k in range(bpc):   # Extraer bits menos significativos
          bits += str((stego[y, x, channel] >> k) & 1)
          # Cuando se acumulan 8 bits, convertirlos a un carácter
          if len(bits) >= 8:
            byte, bits = bits[:8], bits[8:]
            char = chr(int(byte, 2))
            message += char
            if message.endswith(delimiter):
              return message[:-len(delimiter)]

    # 9. Devolver mensaje completo si no se encontró el delimitador
    return message
```

Este algoritmo permite una mayor capacidad de ocultación frente a los otros dos algoritmos presentados, sin embargo, no sólo modifica el bit menos significatico

sino los dos últimos. Aunque estas modificaciones se realicen en las zonas de mayor entropía de la imagen siguen, es importante aclarar que este comportamiento conlleva un ligero aumento en la detectabilidad.

3.5. Métricas y evaluación

Para comparar la calidad de las imágenes estego frente a las originales y cuantificar la efectividad de cada algoritmo, utilizamos cinco métricas:

1. **PSNR** (Peak Signal-to-Noise Ratio)
2. **SSIM** (Structural Similarity Index)
3. **Capacidad de ocultación** (bits por píxel)
4. **Proporción de píxeles modificados**
5. **Proporción de bits modificados**

Si recapitulando a la sección 2.4 podremos consultar los aspectos más teóricos de estas métricas, por tanto, en esta sección nos centraremos en la implementación de las mismas.

3.5.1. PSNR

El PSNR mide la relación máxima posible de la señal (imagen original) al ruido introducido (error de estego), en decibelios.

Listing 3.8: Métricas de evaluación (`stegano/metrics.py`)

```
# PSNR: relación señal/ruido pico
def calculate_psnr(original: Image.Image, stego: Image.Image) ->
 float:
  orig_np  = np.array(original)
  stego_np = np.array(stego)
  return peak_signal_noise_ratio(orig_np, stego_np, data_range
 =255)
```

Interpretación:

- Valores > 40, se consideran cambios imperceptibles.
- Entre $30-40$, los cambios son difíciles de notar.
- Para valores < 30, ya hay una degradación visible.

3.5.2. SSIM

El SSIM compara luminancia, contraste y estructura locales entre dos imágenes, generando un índice en $[0, 1]$ donde 1 indica imagen idéntica:

Listing 3.9: Métricas de evaluación (stegano/metrics.py)

```
# SSIM: similitud estructural
def calculate_ssim(original: Image.Image, stego: Image.Image) ->
 float:
  orig_np  = np.array(original)
  stego_np = np.array(stego)
  return structural_similarity(orig_np, stego_np, channel_axis
 =-1)
```

Interpretación:

- Un valor de $1{,}0$ indica que las imágenes son idénticas.
- Cuando SSIM es $> 0{,}95$, las diferencias son casi imperceptibles.
- Si el resultado es $< 0{,}90$, existen diferencias visibles.

3.5.3. Capacidad de ocultación

Se define como el número de bits del mensaje dividido por el número total de píxeles:

Listing 3.10: Métricas de evaluación (stegano/metrics.py)

```
# Capacidad: bits ocultos por píxel
def calculate_capacity(message: str, image: Image.Image) -> float
 :
  total_bits   = len(text_to_bits(message))
  width, height= image.size
  return total_bits / (width * height)
```

Interpretación:

- Valor es 3 (RGB, 3 canales), se está usando toda la capacidad posible (1 bit por canal por pixel).
- Valor alto, buena eficiencia, se aprovecha el espacio.
- Valor cercano a 0, el mensaje ocultado es muy pequeñoen comparación al tamaño de la imagen.

3.5.4. Proporción de píxeles modificados

Dado un mapa que indica cuántos bits por píxel puede contener cada ubicación, la proporción de píxeles modificados es

$$\textit{Proporción píxeles modificados} = \frac{\textit{Número total de píxeles modificados}}{\textit{Número total de píxeles}} \tag{3.1}$$

donde los píxeles usados se calculan en orden descendente de capacidad para cubrir todos los bits del mensaje.

Listing 3.11: Métricas de evaluación (`stegano/metrics.py`)

```
# Mapa de calor de las diferencias absolutas medios por píxel
def generate_heatmap(original_path, stego_path, output_path=None, title=None):
    original = np.array(Image.open(original_path).convert("RGB"), dtype=np.int16)
    stego = np.array(Image.open(stego_path).convert("RGB"), dtype=np.int16)

    diff = np.abs(original - stego)    # diferencias por canal
    diff_gray = np.mean(diff, axis=2) # diferencia media por píxel

    plt.figure(figsize=(8, 6))
    plt.imshow(diff_gray, cmap='hot', interpolation='nearest')
    plt.axis('off')
    if title:
        plt.title(title)

    if output_path:
        os.makedirs(os.path.dirname(output_path), exist_ok=True)
        plt.savefig(output_path, bbox_inches='tight')
        plt.close()
    else:
        plt.show()

# Proporción de píxeles modificados
def calculate_pixels_modified_ratio(message, slots_map):
    total_bits = len(message) * 8 # bits a ocultar
    total_capacity = slots_map.sum()

    if total_capacity < total_bits:
        raise ValueError("No hay suficiente capacidad para ocultar el mensaje")

    # 1. Num píxeles modificados = total_bits / bits_por_píxel
    modified_pixels = 0
    bits_remaining = total_bits

    # 2. Plano por plano (prioriza zonas con más bits disponibles)
    for bits in [3, 2, 1]:
        count = np.sum(slots_map == bits)
        usable_bits = count * bits
        if bits_remaining <= usable_bits:
            modified_pixels += (bits_remaining + bits - 1) // bits
            bits_remaining = 0
            break
        else:
            modified_pixels += count
            bits_remaining -= usable_bits
```

```
45
46      h, w = slots_map.shape
47      return modified_pixels / (h * w)
```

Interpretación:

- Valor bajo, por ejemplo $0{,}05$, sólo el $0{,}05\,\%$ de los píxeles fueron modificados. El algoritmo concentró la inserción del mensaje en una pequeña parte de la imagen. Minimiza el impacto visual.
- Valor alto, por ejemplo $0{,}90$, el $90\,\%$ de los píxeles fueron modificados. Puede aumentar el riesgo de detección y la aparición de artefactos.

3.5.5. Proporción de bits modificados

Se mide con una operación XOR bit a bit entre la imagen original y la estego, contando el número de bits distintos y dividiéndolo entre el total de bits ($H \times W \times 3 \times 8$).

Listing 3.12: Métricas de evaluación (`stegano/metrics.py`)

```
1   # Proporción de bits modificados
2   def calculate_bits_modified_ratio(original: Image.Image,
3     # Convertir a array
4     orig_np = np.array(original, dtype=np.uint8)
5     stego_np = np.array(stego, dtype=np.uint8)
6
7     # XOR para obtener bits cambiados en cada canal
8     diff = np.bitwise_xor(orig_np, stego_np) # forma (H, W, 3)
9
10    # Convertir cada canal a bits
11    bits = np.unpackbits(diff, axis=-1)      # forma (H, W, 3, 8)
12    total_bits = bits.size
13    modified_bits = bits.sum()
14
15    return modified_bits / total_bits
```

Interpretación:

- Valor bajo, por ejemplo $0{,}01$, sólo el $0{,}01\,\%$ de los bits fueron modificados. El algoritmo logra ocultar la información alterando un mínimo de bits. Minimiza el impacto visual.
- Valor alto, por ejemplo $0{,}15$, se ha alterado una fracción considerable de bits. Puede aumentar el riesgo de detección y la aparición de artefactos. Si se modifican demasiados bits no significativos el color del pixel se altera demasiado. Puede indicar que el mensaje es demasiado largo o el algoritmo es poco optimo.

3.6. Interfaz principal y automatización de pruebas

3.6.1. Interfaz por línea de comandos (CLI)

Para facilitar la interacción con los distintos algoritmos de esteganografía implementados, se ha desarrollado una interfaz por línea de comandos accesible mediante el script `main.py`. Esta CLI permite al usuario codificar y decodificar mensajes en imágenes de forma flexible, sin necesidad de modificar el código fuente. La implementación completa de este script se encuentra accesible en A.1.

La herramienta soporta los tres algoritmos desarrollados:

- **LSB clásico** (`lsb`)
- **LSB optimizado por entropía** (`lsb_opt`)
- **LSB adaptativo** (`lsb_adapt`)

El modo de uso se selecciona mediante el parámetro `-mode`, que puede tomar los valores `encode` o `decode`. A continuación se muestran ejemplos de ejecución para cada modo:

Listing 3.13: Codificaión desde línea de comandos

```
python3 main.py --mode encode --input data/input/img.png \
--output data/output/img_encoded.png \
--message data/messages/msg.txt \
--method lsb_adapt \
--thresholds 0.5 0.75 \
--bits-per-channel 0 1 2
```

Listing 3.14: Decodificaión desde línea de comandos

```
python3 main.py --mode decode --input data/output/img_encoded.png
   \
--original data/input/img.png \
--method lsb_adapt
```

Cada modo requiere parámetros distintos:

- En modo `encode` se debe indicar la imagen de entrada (`-input`), el archivo de mensaje (`-message`) y la imagen de salida (`-output`).
- En modo `decode`, si se usa el método `lsb_adapt`, también se requiere la imagen original sin codificar (`-original`) para comparar la capacidad y reconstruir correctamente el mensaje.

Además, se pueden personalizar:

- El delimitador de fin de mensaje (`-delimiter`).

- Los umbrales de entropía para el algoritmo adaptativo (`-thresholds`).
- La asignación de bits por canal según el nivel de entropía local (`-bits-per-channel`).

Gracias a esta interfaz, es posible realizar pruebas de forma ágil y flexible, además de integrarla fácilmente en scripts automatizados.

3.6.2. Automatización de pruebas

El sistema desarrollado se complementa con una batería de pruebas automatizadas que permiten:

- Verificar la correcta codificación y decodificación de mensajes.
- Evaluar cuantitativamente la calidad de la imagen esteganografiada mediante las métricas de PSNR y SSIM.
- Analizar la eficiencia de ocultación mediante la capacidad (bits por píxel), y la proporción de bits y píxeles modificados.
- Generar mapas de calor para representar visualmente el impacto de la modificación de píxeles.

Las pruebas se ejecutan sobre una serie de imágenes y mensajes de ejemplo ubicados en los directorios `data/input` y `data/messages`. Los resultados se almacenan en `results/`, incluyendo gráficos, métricas en formato `CSV` y comparativas entre métodos. Tanto estos tests como los resultados del fichero CSV pueden consultarse en C y D respectivamante.

Gracias a esta automatización, es posible realizar comparaciones objetivas entre los distintos algoritmos y validar su comportamiento de manera sistemática. Este sistema de evaluación también ha sido clave para ajustar parámetros como los umbrales de entropía y la asignación óptima de bits en el algoritmo adaptativo.

Proximamente, en el capítulo 4 se analizan y comparan los resultados de cada algoritmo, mostrando ejemplos gráficos de las distintas llamadas al programa.

3.6.3. Conclusión

La combinación de una interfaz clara y flexible con un sistema de pruebas completo permite evaluar el sistema de esteganografía en distintos contextos, facilitando tanto su validación como su uso por terceros. Además, estas herramientas han sido fundamentales durante el desarrollo iterativo para verificar que las modificaciones introducidas mantenían la calidad visual y aumentaban la eficiencia de ocultación.

Capítulo 4

Análisis de resultados

En este capítulo se realizan y muestran los resultados de distintos experimentos con imágenes, mostrando las capadidades de cada algoritmos al enfrentarse a distintos casos.

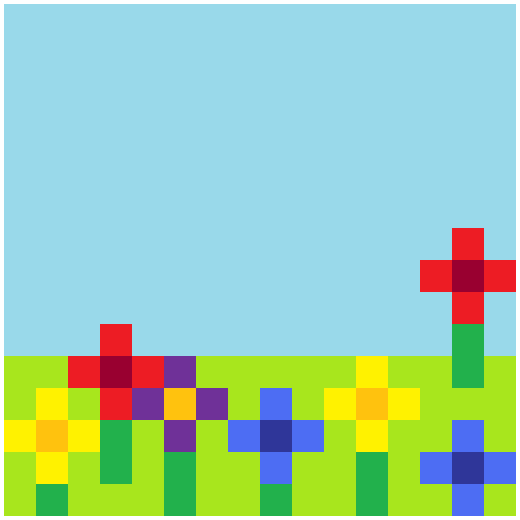
Se proporciona una muestra de datos con los que se trabajará a lo largo del capítulo. Esta muestra está formada por 4 imágenes representadas en la Figura 4.1 y 3 textos con carácterísticas reflejadas en la Tabla 4.1. El análisis se ha realizado sobre la combinación de todos los posibles elementos. Por tanto, existen hasta 36 resultados (4 imágenes $\times$ 3 mensajes $\times$ 3 algoritmos).

4.1. Datos de entrada

En primer lugar, se han seleccionado 4 imágenes del directorio `data/input/`. Las 4 imágenes seleccionadas poseen caráctirísticas diferenciales, las cuales servirán para mostrar el comportamiento de los algoritmos al trabajar sobre distintos entornos.

Respecto a la carácterística única de cada imágen seleccionada:

- La primera imagen tiene un tamaño de 16×16 píxeles, representa unas flores en un campo. Esta imagen es la más pequeña de la muestra y el objetivo es ver como se comportan los algoritmos con poco espacio para ocultar la información.
- La segunda imágen muestra una serpiente y tiene la peculiaridad de tener una única fuente de ruido visual, pues el fondo de la imagen es plano, siendo en su totalidad de un único color, en este caso negro.
- La tercera es una foto está completamente llena de ruido visual, representa un montón de flores ocupando la totalidad de la imagen.
- La peculiaridad de la cuarta imagen es que la mitad contiene elementos planos donde no hay practicamente ruido visual, en este caso el cielo, y la otra mitad está formada por zonas de alta entropía o ruido, en este caso una pradera con un río y un yak.



pixel_flowers.png

snake.png

yellow_flowers.png

yak.png

Figura 4.1: Imágenes originales

Por otro lado, disponemos también de distintos mensajes para ocultar en nuestras imágenes. Del directorio `data/messages/` se han escogido 3 textos de distintas longitudes con el objetivo de que los resultados sean variados. Las características de estos 3 mensajes son las siguientes:

Mensaje	**Palabras**	**Carácteres**	**Bits**
msg1	5	19	152
msg2	202	1.156	9.248
msg3	4.297	29.344	234.752

Cuadro 4.1: Estadísticas de los mensajes

4.2. Codificación y resultados visuales

Para cada combinación de imagen original y mensaje secreto se generaron imágenes esteganografiadas aplicando los tres algoritmos: LSB clásico, LSB optimizado y LSB adaptativo.

En el método clásico de LSB, el proceso de codificación consiste en recorrer los píxeles de la imagen en orden (escaneo por filas) y sustituir el bit menos significativo de cada componente de color con los bits del mensaje de forma secuencial (hasta agotar el mensaje).

La variante optimizada introduce un criterio de selección: antes de insertar, evalúa la complejidad local (entropía) de una ventana de píxeles en este caso de 3×3 y favorece la inserción en regiones de alta entropía (por ejemplo, bordes o texturas complejas) donde los cambios pasarían más desapercibidos. De este modo se reduce la distorsión visual en zonas planas, sacrificando ligeramente la capacidad promedio.

Por último, el algoritmo adaptativo ajusta dinámicamente el número de bits a modificar según la complejidad local del píxel o bloque. En zonas muy texturizadas puede modificar más de un bit por candal de un píxel, mientras que en áreas homogéneas se limita a un bit o incluso omite la inserción para minimizar artefactos. Esta estrategia de esteganografía adaptativa intenta maximizar el balance entre imperceptibilidad y capacidad.

En la práctica, los resultados visuales muestran que los tres métodos producen imágenes esteganografiadas muy similares a las originales. Por ejemplo, comparando las dos imágenes más dispares de nuestra batería de tests, obtenemos la Figura 4.2, que compara la imagen `snake.png` original con su versión modificada por el algoritmo LSB `lsb_snake.png` utilizando `msg3.txt`.

(a) Imagen original (b) Imagen con LSB

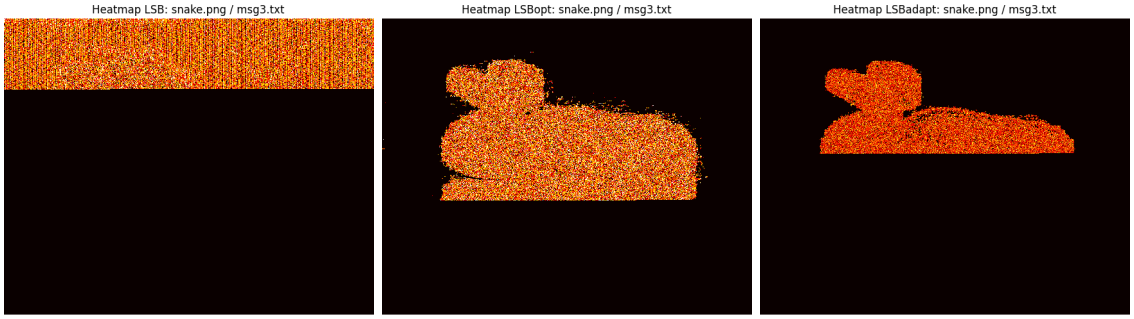
Figura 4.2: Comparación visual entre la imagen original y la imagen tras aplicar codificación LSB.

La figura anterior tiene el peor resultado en el índice SSIM con un valor de $0{,}99336$. Esto se debe a que hemos utilizado la imágen con mas zonas de color planas y el algoritmo clásico, que no es capaz de distinguir estas zonas y por tanto modifica los bits de esos píxeles, creando ruido. A pesar de todo esto, escoger la peor combinación posible de imágenes, algoritmos y mensajes, sigue sin ser suficiente para detectar cambios a simple vista.

4.3. Mapas de diferencia y heatmaps

Debido a que los cambios en todas las imágenes son casi imperceptibles, se tomó la decisión de resaltar las modificaciones introducidas generando mapas de diferencia y "heatmaps" que ilustran la magnitud del cambio pixel a pixel. Un mapa de diferencia usualmente representa la diferencia absoluta (o el error cuadrático) entre la imagen original y la estego, codificando en color las áreas con mayor discrepancia. Los heatmaps pueden mostrar, por ejemplo, la intensidad del error local (MSE) o la entropía asociada a la imagen esteganografiada. Estas visualizaciones facilitan comparar cómo se distribuye el mensaje oculto en cada algoritmo, donde una imagen se ve idéntica al original, gracias a los mapas de diferencias podemos ver donde cambian los bits y con los mapas de calor podmeos ver done y cuantos cambian.

Esto también es muy útil para comparar los cambios que cada algoritmo está realizando. En la Figura 4.3 se muestra un ejemplo de mapas de calor para los 3 algoritmos sobre la misma imagen de prueba. Se observa que LSB clásico modifica de forma secuencial la imagen mientras que las variantes con entropía evitan regiones planas. En el caso de LSB adaptativo observamos como es capaz de concentrar los cambios aún más que LSB optimizado.



(a) LSB clásico (b) LSB optimizado (c) LSB adaptativo

Figura 4.3: Comparación de mapas de calor de los tres algoritmos para `snake.png` con `msg3`.

Cabe aclarar que los mapas de diferencias que se encuentran en el anexo son idénticos a los mapas de calor con la excepción de que estos no muestran más que los píexeles modificados, sin la capacidad para indicar la cantidad de cambios como si lo hace el mapa de calor con sus tonalidades.

4.4. Métricas cuantitativas y comparativas

Más allá de las visualizaciones, las métricas cuantitativas permiten evaluar objetivamente la calidad de la esteganografía. En la Tabla D.1 se resumen los resultados calculados a partir del archivo `data/results/results.csv` para cada caso de prueba. Graficando esos datos podemos evaluar distintos aspectos, como por ejemplo que algoritmo ha logrado mayor simulitud con la imagen original. Si observamos la siguiente figura, que grafica los SSIM de todos los algoritmos, podemos verificar como el algoritmo optimizado logra la mayor fiabilidad, seguido de cerca por el algoritmo adaptativo, mientras que el clásico es con diferencia el menos fiable.

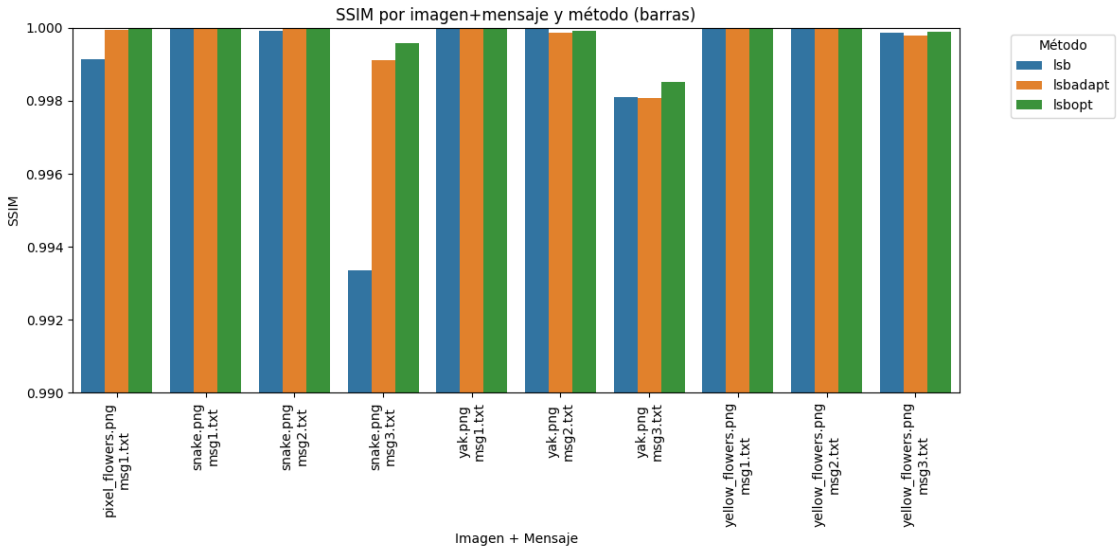


Figura 4.4: Comparación SSIM de todas las pruebas

Si observamos detenidamente podemos ver como por ejemplo no existen valores para `pixel_flowers.png` con `msg2.txt` o `msg3.txt`. Esto se debe a que los mensajes eran demasiado largos para ocultarlos en una imagen de tan poco tamaño.

Otra de las dudas que estos resultados pueden resolvernos es, ¿que algoritmo genera más artefactos o ruido visual? Esta pregunta se puede responder comparando los valores PSNR generados. El la Figura 4.5 podemos obeservar como los algoritmos clásico y optimizado optienen valores muy parecidos mientras que el adaptativo tiene un menor rendimiento. Esto se debe a que este algoritmo es capaz de introducir más cambios por canal, lo que inevitablemente lleva a un aumento en la aparición de ruido en la imagen.

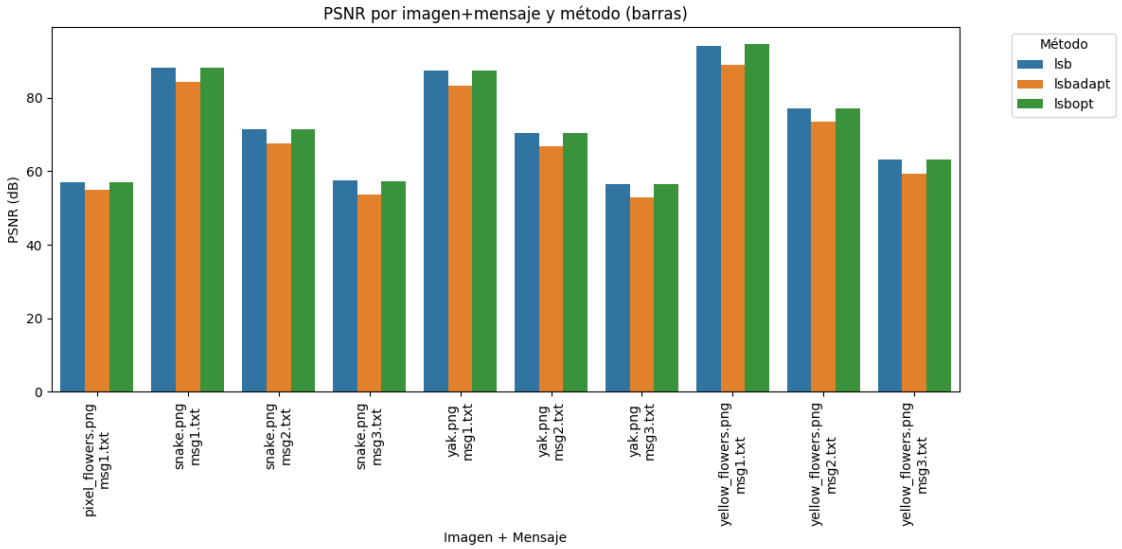


Figura 4.5: Comparación PSNR de todas las pruebas

Podríamos llegar a pensar que el algoritmo adaptativo no sobresale en nada, ya que el algortimo que produce la imagen más fiel a la original y el que menos ruido genera es el optimizado. Pero, ¿y si nos fijamos en la capacidad de ocultar un mensaje? Bien pues es en este aspecto en el que el algoritmo adaptativo se diferencia de los otros dos. Gracias a su capacidad de ocultar varios bits del mensaje por canal, la cantidad de píxeles modificados puede disminuir considerablemente. Observando la Figura 4.6 vemos como especialmente destaca para imágenes con areas planas como `snake.png` y ocultando mensajes gran longitud como `msg3.txt`.

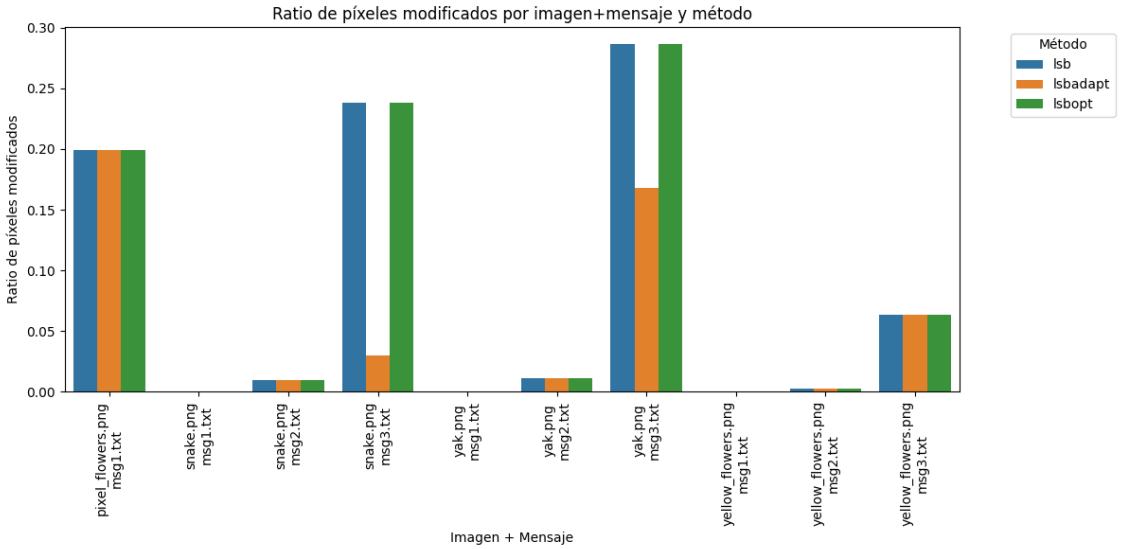


Figura 4.6: Comparación píxeles moficados de todas las pruebas

Si desviamos la atención un momento de los algoritmos, es indiscutible que tanto la imagen y el mensaje con el que operan tienen gran impacto en el resultado final de los mismos. Idealmente se escogerán una imagen y un mensaje acordes, de de tal manera que se aproveche la máxima capacidad de la imágen. Este pequeño detalle se ha medido con la capacidad de ocultación que tiene una imagen. Esta capacidad es independiente del algoritmo empleado ya que mide la relación entre los bits del mensaje y el número de píxeles, en otras palabras, que tan bien un mensaje aprovecha toda la imagen y cuanto se queda sin modificar.

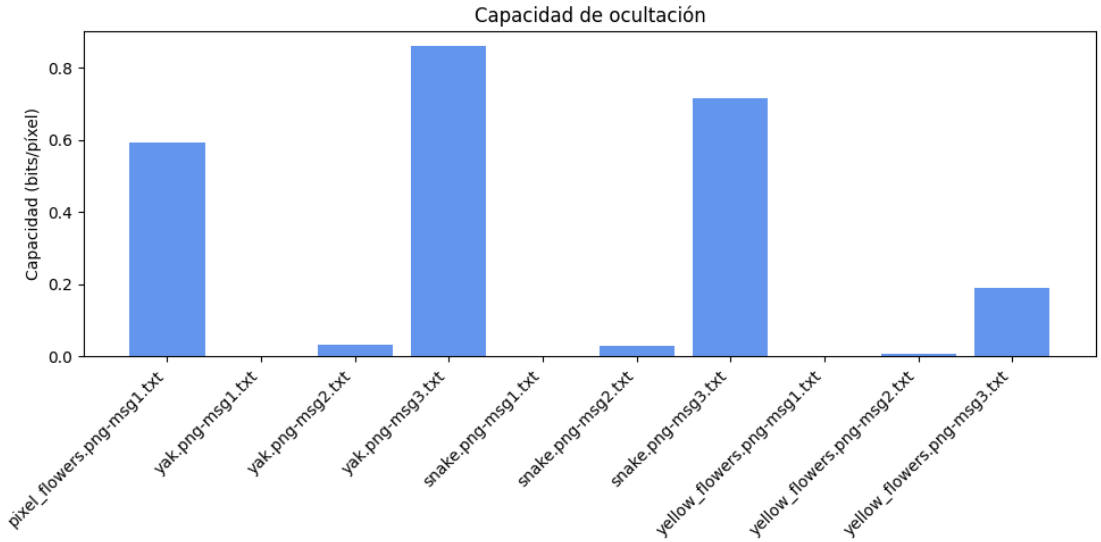


Figura 4.7: Comparación capacidad de ocultación de las imágenes respescto a los mensajes

4.5. Resumen comparativo final

Conociendo todas estas métricas, ahora sabemos que una correcta estegenografía se logra cuando todas las piezas colaboran entre si. Es tan importante escoger el algoritmo correcto como lo es una imágen y longitud del mensaje adecuados. Debemos tener en cuenta las limitaciones que cada uno de estos elementos pueden ocasionar para lograr el mejor resultado posible.

Finalmente, a modo de recomendación, se indican las situaciones de uso preferente de cada método:

- **LSB clásico:** Cuando la imagen no posee grandes superficies planes o el mensaje es de gran longitud.
- **LSB optimizado por entropía:** Cuando la ocultabilidad sea una prioridad clara y no necesitemos ocultar un mensaje demasiado largo.
- **LSB adaptativo:** Cuando la ocultabilidad no es el requisito principal sino la habilidad de ocultar un mensaje extenso.

Estas referencias visuales y cuantitativas completan el análisis comparativo de los resultados obtenidos con la muestra de pruebas. Cada algoritmo posee ventajas específicas según el escenario pero se ha demostrado que las implementaciones basadas en teoría de la información son superiores a la versión clásica. En cualquier caso, la decisión final dependerá de la aplicación concreta y del nivel de imperceptibilidad deseado.

Capítulo 5

Conclusiones y trabajo futuro

Este proyecto de investigación aborda la aplicación y análisis de métodos de ocultamiento de datos en imágenes digitales, centrándose especialmente en la técnica de codificación LSB y su mejora a través del uso de entropía local. Se ha establecido una base teórica sólida a lo largo del estudio, la cual ha sido complementada posteriormente mediante una implementación práctica detallada para finalizar examinando los resultados obtenidos en una comparativa que ha arrojado conclusiones significativas.

Uno de los principales hitos de esta investigación ha sido el desarrollo de una herramienta en Python que puede codificar y decodificar mensajes en imágenes y al mismo tiempo generar una serie de parámetros para evaluar su rendimiento usando métricas como PSNR, SSIM y capacidad de ocultación o mapas de calor. Se ha comprobado que la técnica clásica LSB es efectiva pero genera patrones detectables en áreas uniformes. Por otro lado, los enfoques basados en entropía local optimizan la ocultación al aprovechar las zonas de mayor variabilidad entre los píxeles. La efectividad de los algoritmos se ve notablemente afectada por los datos de entrada, lo que refuerza la importancia de adaptar la estrategia a las características del contenido. La realización de pruebas automatizadas ha simplificado las comparaciones consistentes entre diferentes enfoques, agilizando el proceso experimental y fortaleciendo la confiabilidad de los resultados obtenidos.

El desarrollo de esta herramienta abre nuevas líneas de investigación. Incorporar modelos perceptuales inspirados en el sistema visual humano podría mejorar aún más la selección de píxeles menos perceptibles, reduciendo la notoriedad de las modificaciones. Otro paso natural sería extender el sistema para explorar la esteganografía en otros formatos como vídeo o audio, lo que contribuiría a nuevas líneas de investigación. Desde un enfoque más orientado a la seguridad, sería relevante estudiar la resistencia del método frente a ataques de detección o manipulación, reforzando su solidez dentro del campo de la criptografía. Finalmente, el desarrollo de una interfaz gráfica mejoraría la accesibilidad de la herramienta, facilitando su uso por parte de usuarios no técnicos y permitiendo su despliegue como aplicación multiplataforma o servicio web.

En definitiva, este trabajo ha alcanzado sus objetivos principales, proporcionando una solución funcional y una base metodológica sólida para futuras investigaciones. Más allá de sus contribuciones técnicas, ha sentado un precedente para la exploración de nuevas estrategias de ocultación.

Capítulo 6

Análisis de impacto

Este trabajo fin de grado ha generado impactos significativos en distintos ámbitos. A nivel personal, ha sido una oportunidad para consolidar conocimientos técnicos y desarrollar habilidades como la planificación y el análisis crítico. La implementación de herramientas automatizadas ha permitido una visión integral del desarrollo de software aplicado a la investigación.

En el plano empresarial, la estagnografía a día de hoy tiene innumerables aplicaciones, desde la más evidente como las comunicaciones encubiertas, hasta llegar incluso a ser utilizada para proteger los derechos de autor [18]. Esto la convierte en una base útil para la ciberseguridad y la ocultación de datos.

Socialmente, la esteganografía juega un papel clave en la privacidad y la comunicación segura, pero también presenta riesgos. Para abordar esta dualidad, el proyecto ha priorizado la transparencia y la educación sobre su uso responsable.

Económicamente, su impacto directo es reducido, pero su valor radica en su potencial como base para futuras investigaciones. Además, el uso de software libre refuerza un modelo accesible y sostenible.

Además, se alinea con varios Objetivos de Desarrollo Sostenible (ODS) [19], contribuyendo a la educación, la innovación y la protección de la privacidad. La transparencia del código y la inclusión de métricas visuales reflejan un enfoque consciente del impacto del proyecto.

En definitiva, este trabajo no solo representa un hito académico, sino una base sólida para desarrollos futuros con impacto tecnológico, ético y social.

Bibliografía

[1] Shaadow.io. «¿Qué es la Esteganografía? Descubriendo su historia y su impacto en la seguridad digital». (2024-07-26), dirección: https://www.shaadow.io/posts/what-is-steganography-uncovering-its-history-and-impact-on-digital-security (visitado 21-03-2024).

[2] F. J. Motte. «Esteganografía: el arte de ocultar información en imágenes, audio y video». (2024-02-21), dirección: https://es.linkedin.com/pulse/esteganograf%C3%ADa-el-arte-de-ocultar-informaci%C3%B3n-en-y-jimenez-motte-yxwif (visitado 21-03-2024).

[3] C. Cachin. «An information-theoretic model for steganography». (2004-03-16), dirección: https://www.sciencedirect.com/science/article/pii/S0890540104000409 (visitado 16-03-2025).

[4] P. Classroom. «RGB Color Addition Interactive». (2025), dirección: https://www.physicsclassroom.com/Physics-Interactives/Light-and-Color/RGB-Color-Addition/RGB-Color-Addition-Interactive (visitado 16-05-2025).

[5] A. G. A. D. Cerro. «Esteganografía en archivos digitales - Digital File Steganography». (2021), dirección: https://docta.ucm.es/rest/api/core/bitstreams/3568b46d-2503-4bd8-b9c9-1aba06c5d931/content (visitado 04-03-2025).

[6] C. E. Shannon, «A Mathematical Theory of Communication», *The Bell System Technical Journal*, págs. 379-423, 1948.

[7] MATLAB. «PSNR». (2025), dirección: https://www.mathworks.com/help/vision/ref/psnr.html (visitado 06-04-2025).

[8] Wikipedia. «PSNR». (2019-08-02), dirección: https://es.wikipedia.org/wiki/PSNR (visitado 06-04-2025).

[9] MATLAB. «SSIM». (2025), dirección: https://www.mathworks.com/help/images/ref/ssim.html (visitado 06-04-2025).

[10] Wikipedia. «Structural similarity index measure». (2025-04-06), dirección: https://en.wikipedia.org/wiki/Structural_similarity_index_measure (visitado 06-04-2025).

[11] J. M. S. Arteaga. «Esteganografía, Disciplina para ocultar información». (2017), dirección: http://bibliotecadigital.econ.uba.ar/download/tpos/1502-1765_SanchezArteagaJM.pdf (visitado 17-03-2025).

[12] A. P. Bolaños. «Esteganografía en imágenes digitales». (2023), dirección: https://oa.upm.es/72988/ (visitado 04-03-2025).

[13] J. M. Mateo, «Teoría de la información y la codificación», *Universidad Politécnica de Madrid (UPM - ETSIINF)*, 2024-2025.

[14] Microsoft. «Visual Studio Code». (2025), dirección: https://code.visualstudio.com (visitado 17-04-2025).

[15] P. S. Foundation. «Python». (2025), dirección: https://www.python.org/downloads/ (visitado 17-04-2025).

[16] C. Ltd. «Ubuntu». (2025), dirección: https://ubuntu.com/download/desktop (visitado 17-04-2025).

[17] A. N. Fernández-Calvo. «Stego Project». (2025), dirección: https://github.com/aleexnager/stego-project (visitado 23-04-2025).

[18] J. Guaña-Moya, «Usos y aplicaciones de la esteganografía en la era digital - Uses and applications of steganography in the digital age», *Instituto Superior Tecnológico Japón, Quito, Ecuador*, 2023-06-29.

[19] U. Nations. «Objetivos de Desarrollo Sostenible». (2025), dirección: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/ (visitado 17-05-2025).

Anexos

Apéndice A

Código ficheros fuente

Listing A.1: Programa principal (`main.py`)

```
# Copyright (c) Universidad Politecnica Madrid, 2025
# Authors: Alejandro Náger Fernández-Calvo <a.nager@alumnos.upm.es>
# Dates:
#   Creation: Mar. 24, 2025
#   Modification: Apr. 17, 2025
# Documented by: Alejandro Náger Fernández-Calvo <a.nager@alumnos.upm.es>

# main.py

import argparse
from stegano.lsb import encode_lsb, decode_lsb
from stegano.lsb_optimized import encode_lsb_optimized, decode_lsb_optimized
from stegano.lsb_adaptive import encode_lsb_adaptive, decode_lsb_adaptive
from stegano.utils import load_text_file

def main():
    parser = argparse.ArgumentParser(description="LSB Steganography CLI")
    parser.add_argument("--mode", choices=["encode", "decode"], required=True, help="Modo: encode o decode")
    parser.add_argument("--input", required=True, help="Ruta de la imagen de entrada")
    parser.add_argument("--output", help="Ruta de la imagen de salida (solo para encode)")
    parser.add_argument("--message", help="Ruta del archivo de texto con el mensaje (solo para encode)")
    parser.add_argument("--delimiter", default="#####", help="Delimitador de fin de mensaje")
    parser.add_argument(
        "--method",
```

```
25        choices=["lsb", "lsb_opt", "lsb_adapt"],
26        default="lsb",
27        help="Método: lsb, lsb_opt o lsb_adapt"
28      )
29      parser.add_argument("--original", help="Ruta de la imagen
    original (solo para lsb_adapt decode)")
30      # Parámetros adaptativos
31      parser.add_argument(
32        "--thresholds",
33        nargs=2,
34        type=float,
35        default=(0.5, 0.75),
36        help="Umbrales de entropía t0 t1 para lsb_adapt"
37      )
38      parser.add_argument(
39        "--bits-per-channel",
40        nargs=3,
41        type=int,
42        default=(0, 1, 2),
43        help="Bits por canal (b0 b1 b2) para lsb_adapt"
44      )
45
46      args = parser.parse_args()
47
48      if args.mode == "encode":
49        if not args.output or not args.message:
50          print("Para 'encode' necesitas --output y --message")
51          return
52        message = load_text_file(args.message)
53        if args.method == "lsb":
54          encode_lsb(args.input, message, args.output, delimiter=args
    .delimiter)
55        elif args.method == "lsb_opt":
56          encode_lsb_optimized(args.input, message, args.output,
    delimiter=args.delimiter)
57        else:  # lsb_adapt
58          encode_lsb_adaptive(
59            args.input,
60            message,
61            args.output,
62            window_size=3,
63            thresholds=tuple(args.thresholds),
64            bits_per_channel=tuple(args.bits_per_channel),
65            delimiter=args.delimiter
66          )
67        print(f"Mensaje ocultado en {args.output}")
68
69      else:  # decode
70        if args.method == "lsb":
71          message = decode_lsb(args.input, delimiter=args.delimiter)
72        elif args.method == "lsb_opt":
73          message = decode_lsb_optimized(args.input, delimiter=args.
    delimiter)
```

```
    else:  # lsb_adapt
      if not args.original:
        print("Para 'lsb_adapt' decode necesitas --original")
        return
      message = decode_lsb_adaptive(
        args.input,
        args.original,
        window_size=3,
        thresholds=tuple(args.thresholds),
        bits_per_channel=tuple(args.bits_per_channel),
        delimiter=args.delimiter
      )
      print("Mensaje extraído:")
      print(message)

if __name__ == "__main__":
  main()
```

Listing A.2: Funciones auxiliares (`stegano/utils.py`)

```
# Copyright (c) Universidad Politecnica Madrid, 2025
# Authors: Alejandro Náger Fernández-Calvo <a.nager@alumnos.upm.es>
# Dates:
#  Creation: Mar. 24, 2025
#  Modification: Apr. 17, 2025
# Documented by: Alejandro Náger Fernández-Calvo <a.nager@alumnos.upm.es>

# stegano/utils.py

def text_to_bits(text: str) -> str:
  """Convierte una cadena de texto a una cadena binaria."""
  return ''.join(format(ord(char), '08b') for char in text)

def bits_to_text(bits: str) -> str:
  """Convierte una cadena binaria a texto."""
  chars = [bits[i:i+8] for i in range(0, len(bits), 8)]
  return ''.join(chr(int(b, 2)) for b in chars)

def add_delimiter(message: str, delimiter: str = "#####") -> str:
  """Añade un delimitador al final del mensaje para indicar su final."""
  return message + delimiter

def remove_delimiter(message: str, delimiter: str = "#####") -> str:
  """Elimina el delimitador del mensaje extraído."""
  return message.split(delimiter)[0]

def load_text_file(path: str) -> str:
  """Lee un archivo de texto y devuelve su contenido."""
```

```
29      with open(path, "r", encoding="utf-8") as file:
30        return file.read()
```

Apéndice B

Código fuente algoritmos LSB

Listing B.1: Implementación LSB clásica (`stegano/lsb.py`)

```python
# Copyright (c) Universidad Politecnica Madrid, 2025
# Authors: Alejandro Náger Fernández-Calvo <a.nager@alumnos.upm.es>
# Dates:
#  Creation: Mar. 24, 2025
#  Modification: Apr. 17, 2025
# Documented by: Alejandro Náger Fernández-Calvo <a.nager@alumnos.upm.es>

# stegano/lsb.py

from PIL import Image
from stegano.utils import text_to_bits, bits_to_text, add_delimiter, remove_delimiter

def encode_lsb(image_path: str, message: str, output_path: str, delimiter: str = "#####") -> None:
    """Oculta un mensaje en la imagen utilizando LSB y guarda el resultado."""
    # 1. Cargar y convertir a RGB
    image = Image.open(image_path).convert("RGB")
    pixels = list(image.getdata())

    # 2. Transformar mensaje a bits y añadir delimitador
    binary_message = text_to_bits(add_delimiter(message, delimiter))
    message_len = len(binary_message)

    # 3. Comprobar capacidad: 3 bits por pixel
    if message_len > len(pixels) * 3:
        raise ValueError("El mensaje es demasiado largo para la capacidad de la imagen.")

    # 4. Inserción bit a bit de los LSB de R, G y B
```

```
    new_pixels = []
    bit_index = 0
    for r, g, b in pixels:
        if bit_index < message_len:
        r = (r & ~1) | int(binary_message[bit_index])
      bit_index += 1
        if bit_index < message_len:
        g = (g & ~1) | int(binary_message[bit_index])
      bit_index += 1
      if bit_index < message_len:
        b = (b & ~1) | int(binary_message[bit_index])
        bit_index += 1
      new_pixels.append((r, g, b))

    # 5. Guardar imagen estego
    image.putdata(new_pixels)
    image.save(output_path)

  def decode_lsb(image_path: str, delimiter: str = "#####") -> str:
    """Extrae un mensaje oculto de la imagen utilizando LSB."""
    image = Image.open(image_path)
    image = image.convert("RGB")
    pixels = list(image.getdata())

    # 1. Leer los LSB de cada canal
    bits = ""
    for r, g, b in pixels:
      bits += str(r & 1) + str(g & 1) + str(b & 1)

    # 2. Reconstruir texto y eliminar delimitador
    message = bits_to_text(bits)
    return remove_delimiter(message, delimiter)
```

Listing B.2: Implementación LSB optimizado (`stegano/lsb_optimized.py`)

```
# Copyright (c) Universidad Politecnica Madrid, 2025
# Authors: Alejandro Náger Fernández-Calvo <a.nager@alumnos.upm.
  es>
# Dates:
#  Creation: Mar. 24, 2025
#  Modification: Apr. 17, 2025
# Documented by: Alejandro Náger Fernández-Calvo <a.nager@alumnos
  .upm.es>

# stegano/lsb_optimized.py

import os
from PIL import Image
import numpy as np
from stegano.utils import text_to_bits, add_delimiter
from stegano.entropy import compute_entropy_map,
  normalize_entropy_map
```

```
15
16  def encode_lsb_optimized(image_path: str,
17              message: str,
18              output_path: str,
19              window_size: int = 3,
20              threshold: float = 0.5,
21              delimiter: str = "#####") -> None:
22
23    # 1. Carga y preparación
24    img = Image.open(image_path).convert("RGB")
25    arr = np.array(img)
26    ent_norm = normalize_entropy_map(compute_entropy_map(img,
      window_size))
27    mask_positions = np.argwhere(ent_norm >= threshold)
28
29    # 2. Comprobar capacidad
30    if mask_positions.size == 0:
31      raise ValueError("No se encontraron regiones con entropía
      suficiente.")
32    bits = text_to_bits(add_delimiter(message, delimiter))
33    total_bits = len(bits)
34    if total_bits > len(mask_positions) * 3:
35      raise ValueError("El mensaje es demasiado largo para la
      capacidad de la imagen.")
36
37    # 3. Embedding LSB en píxeles seleccionados
38    stego = arr.copy()
39    bit_idx = 0
40    for y, x in mask_positions:
41      if bit_idx >= total_bits: break
42      for c in range(3):
43      if bit_idx < total_bits:
44        stego[y, x, c] = (stego[y, x, c] & 0xFE) | int(bits[bit_idx
      ])
45        bit_idx += 1
46
47    # 4. Guardar imagen estego
48    Image.fromarray(stego).save(output_path)
49
50
51  def decode_lsb_optimized(image_path: str,
52              window_size: int = 3,
53              threshold: float = 0.5,
54              delimiter: str = "#####") -> str:
55    """
56    Extrae un mensaje oculto usando LSB en píxeles seleccionados
      según entropía local
57    calculada sobre la imagen original.
58    """
59    # 1. Cargar imagenes estego y original
60    stego_img = Image.open(image_path).convert("RGB")
61    stego_arr = np.array(stego_img)
62
```

```
    # 2. Derivar ruta de la imagen original a partir del nombre de archivo
    base = os.path.basename(image_path)
    if base.startswith("lsbopt_"):
      orig_name = base[len("lsbopt_"):]
    elif base.startswith("lsb_"):
      orig_name = base[len("lsb_"):]
    else:
      orig_name = base
    orig_path = os.path.join("data", "input", orig_name)
    orig_img = Image.open(orig_path).convert("RGB")

    # 3. Reconstrucción de la máscara de entropía
    ent_norm = normalize_entropy_map(compute_entropy_map(orig_img, window_size))
    mask_positions = np.argwhere(ent_norm >= threshold)

    # 4. Extracción bit a bit con parada en delimitador
    bits = ""
    message = ""
    for y, x in mask_positions:
      for c in range(3):
      bits += str(int(stego_arr[y, x, c]) & 1)
      if len(bits) >= 8:
        byte = bits[:8]
        bits = bits[8:]
        char = chr(int(byte, 2))
        message += char
        if message.endswith(delimiter):
          return message[:-len(delimiter)]

    # 5. Si nunca vimos el delimitador, devolvemos todo lo leído
    return message
```

Listing B.3: Implementación LSB adaptativo (`stegano/lsb_adaptive.py`)

```
# Copyright (c) Universidad Politecnica Madrid, 2025
# Authors: Alejandro Náger Fernández-Calvo <a.nager@alumnos.upm.es>
# Dates:
#  Creation: Apr. 3, 2025
#  Modification: Apr. 17, 2025
# Documented by: Alejandro Náger Fernández-Calvo <a.nager@alumnos.upm.es>

# stegano/lsb_adaptive.py

import os
from PIL import Image
import numpy as np
from stegano.utils import text_to_bits, add_delimiter
from stegano.entropy import compute_entropy_map,
```

```
    normalize_entropy_map
15
16  def encode_lsb_adaptive(image_path: str,
17              message: str,
18              output_path: str,
19              window_size: int = 3,
20              thresholds: tuple[float,float] = (0.5, 0.75),
21              bits_per_channel: tuple[int,int,int] = (0, 1, 2),
22              delimiter: str = "#####") -> None:
23      """
24      Multi-LSB adaptativo por canal:
25      entropy < t0       -> 0 bits/canal
26      t0 <= entropy < t1 -> 1 bit/canal
27      entropy >= t       -> 2 bits/canal
28      """
29
30      # 1. Cargar imagen
31      img = Image.open(image_path).convert("RGB")
32      arr = np.array(img)
33      h, w, _ = arr.shape
34
35      # 2. Calc. y normalizar mapa de entropía local sobre imagen
        original
36      ent_norm = normalize_entropy_map(compute_entropy_map(img,
        window_size))
37
38      # 3. Extraer los umbrales y configuración de bits por canal
39      t0, t1 = thresholds
40      b0, b1, b2 = bits_per_channel
41
42      # 4. Generar mapa de bits por canal
43      chan_slots = np.zeros((h, w), dtype=int)
44      chan_slots[ent_norm < t0] = b0
45      chan_slots[(ent_norm >= t0) & (ent_norm < t1)] = b1
46      chan_slots[ent_norm >= t1] = b2
47
48      # 5. Preparar el mensaje para su inserción
49      bits = text_to_bits(add_delimiter(message, delimiter))
50      total_bits = len(bits)
51      capacity = int(np.sum(chan_slots) * 3) # Capacidad total en
        bits (3 bits por canal)
52
53      # 6. Comprobar capacidad
54      if total_bits > capacity:
55          raise ValueError("El mensaje es demasiado largo para la
        capacidad de la imagen.")
56
57      # 7. Crear copia de la imagen para modificar
58      stego = arr.copy()
59      bit_idx = 0
60
61      # 8. Inserción de bits canal por canal
62      ys, xs = np.nonzero(chan_slots > 0)
```

```
63      positions = sorted(zip(ys, xs), key=lambda p: chan_slots[p[0],
        p[1]], reverse=True)
64
65      for y, x in positions:
66        bpc = chan_slots[y, x]   # Bits por canal en este píxel
67        for channel in range(3): # Iterar sobre canales R, G, B
68          for k in range(bpc): # Insertar bits menos significativos
69            if bit_idx >= total_bits:
70              break
71            # Limpiza el bit k
72            mask = (~(1 << k)) & 0xFF
73            orig_val = int(stego[y, x, channel])
74            # Construir nuevo bit
75            new_bit = (int(bits[bit_idx]) & 1) << k
76            # Sustituir bit k
77            stego[y, x, channel] = (orig_val & mask) | new_bit
78            bit_idx += 1
79          if bit_idx >= total_bits:
80            break
81        if bit_idx >= total_bits:
82          break
83
84      # 9. Guardar imagen estego
85      Image.fromarray(stego.astype(np.uint8)).save(output_path)
86
87
88  def decode_lsb_adaptive(image_path: str,
89                original_path: str,
90                window_size: int = 3,
91                thresholds: tuple[float,float] = (0.5, 0.75),
92                bits_per_channel: tuple[int,int,int] = (0, 1, 2),
93                delimiter: str = "#####") -> str:
94      """
95      Decodifica el multi-LSB adaptativo.
96      """
97
98      # 1. Cargar imagenes estego y original
99      stego_img = Image.open(image_path).convert("RGB")
100     stego = np.array(stego_img)
101     orig_img = Image.open(original_path).convert("RGB")
102
103     # 2. Calc. y normalizar mapa de entropía local sobre imagen
        original
104     ent_norm = normalize_entropy_map(compute_entropy_map(orig_img,
        window_size))
105     # 3. Extraer los umbrales y configuración de bits por canal
106     t0, t1 = thresholds
107     b0, b1, b2 = bits_per_channel
108
109     # 4. Mapa indicando n bits por canal extraer en cada píxel
110     h, w, _ = stego.shape
111     chan_slots = np.zeros((h, w), dtype=int)
112
```

```
    # 5. Asignar bits por canal según el valor de entropía local
    chan_slots[ent_norm < t0] = b0
    chan_slots[(ent_norm >= t0) & (ent_norm < t1)] = b1
    chan_slots[ent_norm >= t1] = b2

    # 6. Coordenadas de píxeles con al menos 1 bit disponible
    ys, xs = np.nonzero(chan_slots > 0)

    # 7. Ordenar las posiciones por cantidad de bits disponibles (de mayor a menor)
    positions = sorted(zip(ys, xs), key=lambda p: chan_slots[p[0], p[1]], reverse=True)

    # 8. Extraer bits de la imagen estego
    bits = ""
    message = ""
    for y, x in positions:
      bpc = chan_slots[y, x]   # Bits por canal en este píxel
      for channel in range(3): # Iterar sobre canales R, G, B
        for k in range(bpc): # Extraer bits menos significativos
          bits += str((stego[y, x, channel] >> k) & 1)
          # Cuando se acumulan 8 bits, convertirlos a un carácter
          if len(bits) >= 8:
            byte, bits = bits[:8], bits[8:]
            char = chr(int(byte, 2))
            message += char
            if message.endswith(delimiter):
              return message[:-len(delimiter)]
            # 9. Devolver mensaje completo si no se encontró el delimitador
    return message
```

Apéndice C

Código fuente tests automáticos

Listing C.1: Tests LSB clásica (`tests/test_lsb.py`)

```python
# Copyright (c) Universidad Politecnica Madrid, 2025
# Authors: Alejandro Náger Fernández-Calvo <a.nager@alumnos.upm.es>
# Dates:
#  Creation: Mar. 24, 2025
#  Modification: Apr. 17, 2025
# Documented by: Alejandro Náger Fernández-Calvo <a.nager@alumnos.upm.es>

#tests/test_lsb.py

import os
import pytest
import numpy as np
from stegano.lsb import encode_lsb, decode_lsb
from stegano.utils import load_text_file
from stegano.metrics import calculate_psnr, calculate_ssim, calculate_capacity, generate_heatmap, calculate_pixels_modified_ratio, calculate_bits_modified_ratio
from PIL import Image

# Rutas
INPUT_IMAGE_DIR = "data/input"
MESSAGE_DIR = "data/messages"
OUTPUT_IMAGE_DIR = "data/output"
RESULTS_DIR = "data/results"

os.makedirs(OUTPUT_IMAGE_DIR, exist_ok=True)
os.makedirs(RESULTS_DIR, exist_ok=True)

# Casos de prueba
test_cases = [
  ("pixel_flowers.png", "msg1.txt"),
  ("pixel_flowers.png", "msg2.txt"),
```

```
    ("pixel_flowers.png", "msg3.txt"),
    ("yak.png", "msg1.txt"),
    ("yak.png", "msg2.txt"),
    ("yak.png", "msg3.txt"),
    ("snake.png", "msg1.txt"),
    ("snake.png", "msg2.txt"),
    ("snake.png", "msg3.txt"),
    ("yellow_flowers.png", "msg1.txt"),
    ("yellow_flowers.png", "msg2.txt"),
    ("yellow_flowers.png", "msg3.txt")
]

# Resultados
@pytest.fixture(scope="session", autouse=True)
def share_results(request):
    return request.config.results

# Luego, dentro del test usa esta forma:
def test_lsb_pipeline(image_name, message_name, share_results):
    ...
    share_results.append({...})

@pytest.mark.parametrize("image_name, message_name", test_cases)
def test_lsb_pipeline(image_name, message_name, share_results):
    input_image_path = os.path.join(INPUT_IMAGE_DIR, image_name)
    message_path = os.path.join(MESSAGE_DIR, message_name)
    output_image_path = os.path.join(OUTPUT_IMAGE_DIR, f"lsb_{
    image_name}")

    message = load_text_file(message_path)
    encode_lsb(input_image_path, message, output_image_path)

    extracted_message = decode_lsb(output_image_path)
    assert message == extracted_message, "El mensaje extraído no
    coincide con el original"

    original_img = Image.open(input_image_path).convert("RGB")
    stego_img = Image.open(output_image_path).convert("RGB")

    psnr = calculate_psnr(original_img, stego_img)
    ssim = calculate_ssim(original_img, stego_img)
    capacity = calculate_capacity(message, stego_img)

    w, h = original_img.size
    slots_map = np.full((h, w), 3, dtype=int)

    pixels_ratio = calculate_pixels_modified_ratio(message,
    slots_map)
    bit_ratio = calculate_bits_modified_ratio(original_img,
    stego_img)

    # Guardar métricas
    share_results.append({
```

```
80        "image": image_name,
81        "message": message_name,
82        "psnr": psnr,
83        "ssim": ssim,
84        "capacity": capacity,
85        "pixels_modified_ratio": pixels_ratio,
86        "bits_modified_ratio": bit_ratio,
87        "length": len(message),
88        "method": "lsb"
89    })
90
91    heatmap_path = os.path.join(
92        "data", "heatmaps",
93        f"heatmap_lsb_{image_name.replace('.png','')}_{message_name.
      replace('.txt','')}.png"
94    )
95    generate_heatmap(
96        os.path.join(INPUT_IMAGE_DIR, image_name),
97        output_image_path,
98        output_path=heatmap_path,
99        title=f"Heatmap LSB: {image_name} / {message_name}"
100   )
```

Listing C.2: Tests LSB optimizado (`tests/test_lsb_optimized.py`)

```
1   # Copyright (c) Universidad Politecnica Madrid, 2025
2   # Authors: Alejandro Náger Fernández-Calvo <a.nager@alumnos.upm.
      es>
3   # Dates:
4   #   Creation: Mar. 24, 2025
5   #   Modification: Apr. 17, 2025
6   # Documented by: Alejandro Náger Fernández-Calvo <a.nager@alumnos
      .upm.es>
7
8   # tests/test_lsb_optimized.py
9
10  import os
11  import pytest
12  from stegano.lsb_optimized import encode_lsb_optimized,
      decode_lsb_optimized
13  from stegano.utils import load_text_file
14  from stegano.metrics import calculate_psnr, calculate_ssim,
      calculate_capacity, generate_heatmap,
      calculate_pixels_modified_ratio, calculate_bits_modified_ratio
15  from stegano.entropy import compute_entropy_map,
      normalize_entropy_map
16  from PIL import Image
17
18  # Rutas
19  INPUT_IMAGE_DIR = "data/input"
20  MESSAGE_DIR = "data/messages"
21  OUTPUT_IMAGE_DIR = "data/output"
```

```
22  RESULTS_DIR = "data/results"
23
24  os.makedirs(OUTPUT_IMAGE_DIR, exist_ok=True)
25  os.makedirs(RESULTS_DIR, exist_ok=True)
26
27  # Casos de prueba
28  test_cases = [
29    ("pixel_flowers.png", "msg1.txt"),
30    ("pixel_flowers.png", "msg2.txt"),
31    ("pixel_flowers.png", "msg3.txt"),
32    ("yak.png", "msg1.txt"),
33    ("yak.png", "msg2.txt"),
34    ("yak.png", "msg3.txt"),
35    ("snake.png", "msg1.txt"),
36    ("snake.png", "msg2.txt"),
37    ("snake.png", "msg3.txt"),
38    ("yellow_flowers.png", "msg1.txt"),
39    ("yellow_flowers.png", "msg2.txt"),
40    ("yellow_flowers.png", "msg3.txt")
41  ]
42
43  @pytest.fixture(scope="session", autouse=True)
44  def share_results(request):
45    return request.config.results
46
47  @pytest.mark.parametrize("image_name, message_name", test_cases)
48  def test_lsb_optimized_pipeline(image_name, message_name,
     share_results):
49    input_image_path = os.path.join(INPUT_IMAGE_DIR, image_name)
50    message_path = os.path.join(MESSAGE_DIR, message_name)
51    output_image_path = os.path.join(OUTPUT_IMAGE_DIR, f"lsbopt_{
     image_name}")
52
53    message = load_text_file(message_path)
54    encode_lsb_optimized(input_image_path, message,
     output_image_path)
55
56    extracted_message = decode_lsb_optimized(output_image_path)
57    assert message == extracted_message, "El mensaje extraído no
     coincide con el original"
58
59    original_img = Image.open(input_image_path).convert("RGB")
60    stego_img = Image.open(output_image_path).convert("RGB")
61
62    psnr = calculate_psnr(original_img, stego_img)
63    ssim = calculate_ssim(original_img, stego_img)
64    capacity = calculate_capacity(message, stego_img)
65
66    ent_norm = normalize_entropy_map(compute_entropy_map(
     original_img))
67    slots_map = (ent_norm >= 0.5).astype(int) * 3
68
69    pixels_ratio = calculate_pixels_modified_ratio(message,
```

```
    slots_map)
    bit_ratio = calculate_bits_modified_ratio(original_img,
    stego_img)

    # Guardar métricas
    share_results.append({
      "image": image_name,
      "message": message_name,
      "psnr": psnr,
      "ssim": ssim,
      "capacity": capacity,
      "pixels_modified_ratio": pixels_ratio,
      "bits_modified_ratio": bit_ratio,
      "length": len(message),
      "method": "lsbopt"
    })

    # Generar y guardar el heatmap
    heatmap_path = os.path.join(
      "data", "heatmaps",
      f"heatmap_lsbopt_{image_name.replace('.png','')}_{
    message_name.replace('.txt','')}.png"
    )
    generate_heatmap(
      os.path.join(INPUT_IMAGE_DIR, image_name),
      output_image_path,
      output_path=heatmap_path,
      title=f"Heatmap LSBopt: {image_name} / {message_name}"
    )
```

Listing C.3: Tests LSB adaptativo (`tests/test_lsb_adaptive.py`)

```
# Copyright (c) Universidad Politecnica Madrid, 2025
# Authors: Alejandro Náger Fernández-Calvo <a.nager@alumnos.upm.
  es>
# Dates:
#   Creation: Apr. 3, 2025
#   Modification: Apr. 17, 2025
# Documented by: Alejandro Náger Fernández-Calvo <a.nager@alumnos
  .upm.es>

# tests/test_lsb_adaptive.py

import os
import pytest
import numpy as np
from PIL import Image

from stegano.entropy import compute_entropy_map,
  normalize_entropy_map
from stegano.lsb_adaptive import encode_lsb_adaptive,
  decode_lsb_adaptive
```

```
17  from stegano.utils import load_text_file
18  from stegano.metrics import calculate_psnr, calculate_ssim,
     calculate_capacity, generate_heatmap,
     calculate_pixels_modified_ratio, calculate_bits_modified_ratio
19
20  INPUT_DIR = "data/input"
21  MESSAGE_DIR = "data/messages"
22  OUTPUT_DIR = "data/output"
23
24  os.makedirs(OUTPUT_DIR, exist_ok=True)
25
26  # Casos de prueba
27  test_cases = [
28    ("pixel_flowers.png", "msg1.txt"),
29    ("pixel_flowers.png", "msg2.txt"),
30    ("pixel_flowers.png", "msg3.txt"),
31    ("yak.png", "msg1.txt"),
32    ("yak.png", "msg2.txt"),
33    ("yak.png", "msg3.txt"),
34    ("snake.png", "msg1.txt"),
35    ("snake.png", "msg2.txt"),
36    ("snake.png", "msg3.txt"),
37    ("yellow_flowers.png", "msg1.txt"),
38    ("yellow_flowers.png", "msg2.txt"),
39    ("yellow_flowers.png", "msg3.txt")
40  ]
41
42  @pytest.fixture(scope="session", autouse=True)
43  def share_results(request):
44    return request.config.results
45
46  @pytest.mark.parametrize("image_name,message_name", test_cases)
47  def test_lsb_adaptive_pipeline(image_name, message_name,
     share_results):
48    inp = os.path.join(INPUT_DIR, image_name)
49    msgp = os.path.join(MESSAGE_DIR, message_name)
50    out = os.path.join(OUTPUT_DIR, f"lsbadapt_{image_name}")
51
52    # Leer y ocultar
53    message = load_text_file(msgp)
54    encode_lsb_adaptive(
55      inp,
56      message,
57      out,
58      window_size=3,
59      thresholds=(0.5, 0.75),
60      bits_per_channel=(0, 1, 2),
61      delimiter="#####"
62    )
63
64    # Extraer
65    extracted = decode_lsb_adaptive(
66      out,
```

```
      inp,
      window_size=3,
      thresholds=(0.5, 0.75),
      bits_per_channel=(0, 1, 2),
      delimiter="#####"
    )
    assert extracted == message, "El mensaje extraído no coincide"

    # Cargar para métricas
    original = Image.open(inp).convert("RGB")
    stego    = Image.open(out).convert("RGB")

    psnr     = calculate_psnr(original, stego)
    ssim     = calculate_ssim(original, stego)
    capacity = calculate_capacity(message, stego)

    # Reconstruir slot_map de bits por canal
    ent_norm = normalize_entropy_map(compute_entropy_map(original))
    t0, t1 = (0.5, 0.75)
    b0, b1, b2 = (0, 1, 2)
    h, w = original.height, original.width

    chan_slots = np.zeros((h, w), dtype=int)
    chan_slots[ent_norm < t0] = b0
    chan_slots[(ent_norm >= t0) & (ent_norm < t1)] = b1
    chan_slots[ent_norm >= t1] = b2

    # Total bits disponibles por píxel = slots_canal * 3 canales
    bits_per_pixel_map = chan_slots * 3

    # Métrica de píxeles modificados
    pixels_ratio = calculate_pixels_modified_ratio(message,
    bits_per_pixel_map)
    # Métrica de bits modificados
    bit_ratio    = calculate_bits_modified_ratio(original, stego)

    # Guardar resultados
    share_results.append({
      "image": image_name,
      "message": message_name,
      "psnr": psnr,
      "ssim": ssim,
      "capacity": capacity,
      "pixels_modified_ratio": pixels_ratio,
      "bits_modified_ratio": bit_ratio,
      "length": len(message),
      "method": "lsbadapt"
    })

    # Generar y guardar el heatmap
    heatmap_path = os.path.join(
      "data", "heatmaps",
      f"heatmap_lsbadapt_{image_name.replace('.png','')}_{
```

```
    message_name.replace('.txt','')}.png"
119  )
120  generate_heatmap(
121    os.path.join(INPUT_DIR, image_name),
122    out,
123    output_path=heatmap_path,
124    title=f"Heatmap LSBadapt: {image_name} / {message_name}"
125  )
```

Apéndice D

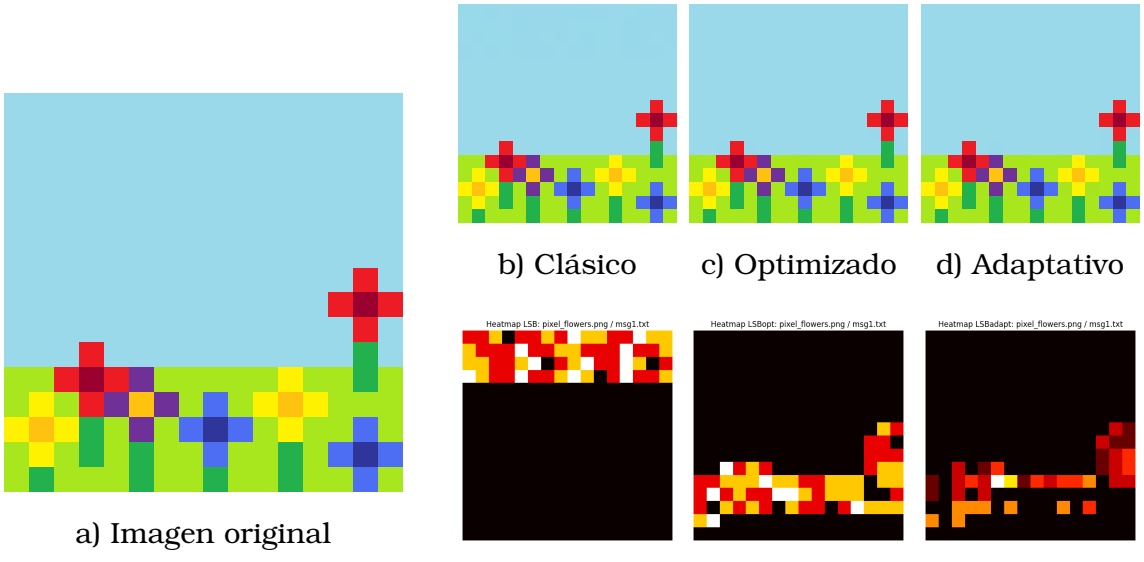
Tabla con resultados .csv

Cuadro D.1: Comparativa de métricas entre los algoritmos LSB clásico, adaptativo y optimizado por entropía.

Imagen	Mensaje	Método	PSNR	SSIM	Capacidad	Píxeles Mod.	Bits Mod.
pixel_flowers	msg1	lsb	56.94	0.9991	0.5938	0.1992	0.016438
		lsbadapt	55.03	0.9999	0.5938	0.1992	0.015462
		lsbopt	57.07	0.9999	0.5938	0.1992	0.015951
pixel_flowers	msg2	**Error: El mensaje es demasiado largo para la capacidad de la imagen.**					
pixel_flowers	msg3	**Error: El mensaje es demasiado largo para la capacidad de la imagen.**					
yak	msg1	lsb	87.36	1.0000	0.0006	0.0002	0.000015
		lsbadapt	83.32	1.0000	0.0006	0.0002	0.000014
		lsbopt	87.45	1.0000	0.0006	0.0002	0.000015
yak	msg2	lsb	70.47	0.9999	0.0338	0.0113	0.000728
		lsbadapt	66.87	0.9999	0.0338	0.0113	0.000721
		lsbopt	70.52	0.9999	0.0338	0.0113	0.000721
yak	msg3	lsb	56.57	0.9981	0.8590	0.2863	0.017896
		lsbadapt	52.99	0.9981	0.8590	0.1679	0.017898
		lsbopt	56.56	0.9985	0.8590	0.2863	0.017956
snake	msg1	lsb	88.11	1.0000	0.0005	0.0002	0.000013
		lsbadapt	84.26	1.0000	0.0005	0.0002	0.000011
		lsbopt	88.29	1.0000	0.0005	0.0002	0.000012
snake	msg2	lsb	71.40	0.9999	0.0282	0.0094	0.000589
		lsbadapt	67.54	1.0000	0.0282	0.0094	0.000594
		lsbopt	71.48	1.0000	0.0282	0.0094	0.000578
snake	msg3	lsb	57.57	0.9934	0.7150	0.2383	0.014209
		lsbadapt	53.59	0.9991	0.7150	0.0297	0.014912
		lsbopt	57.39	0.9996	0.7150	0.2383	0.014837
yellow_flowers	msg1	lsb	94.16	1.0000	0.0001	0.0000	0.000003
		lsbadapt	88.88	1.0000	0.0001	0.0000	0.000004
		lsbopt	94.55	1.0000	0.0001	0.0000	0.000003
yellow_flowers	msg2	lsb	77.18	1.0000	0.0075	0.0025	0.000156
		lsbadapt	73.41	1.0000	0.0075	0.0025	0.000155
		lsbopt	77.19	1.0000	0.0075	0.0025	0.000155
yellow_flowers	msg3	lsb	63.13	0.9999	0.1910	0.0637	0.003950
		lsbadapt	59.34	0.9998	0.1910	0.0637	0.003965
		lsbopt	63.14	0.9999	0.1910	0.0637	0.003947

Apéndice E

Comparación resultados de imágenes



a) Imagen original

b) Clásico

c) Optimizado

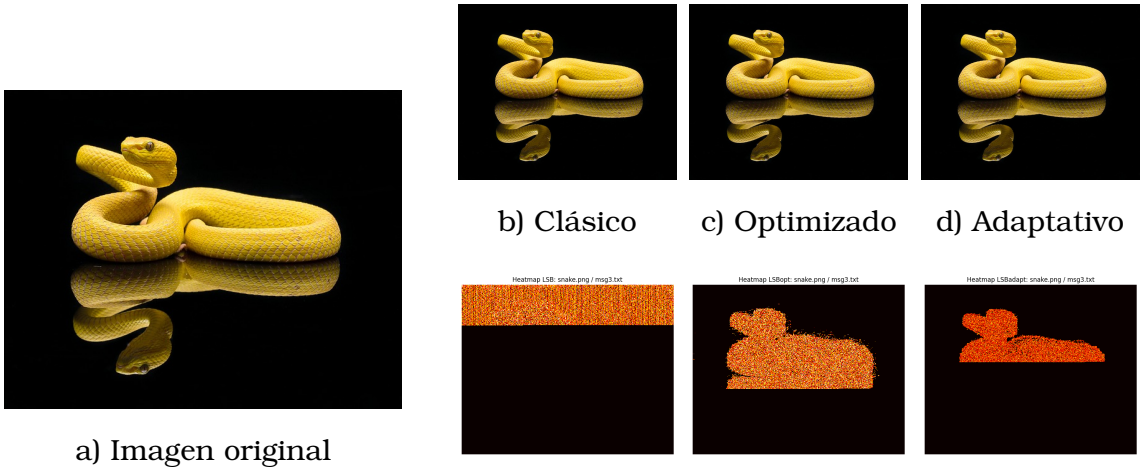
d) Adaptativo

e) Heatmap clásico

f) Heatmap optimizado

g) Heatmap adaptativo

Figura E.1: Comparación visual entre la imagen original `pixel_flowers.png`, versiones codificadas y sus respectivos mapas de calor con `msg1.txt`.



a) Imagen original

b) Clásico

c) Optimizado

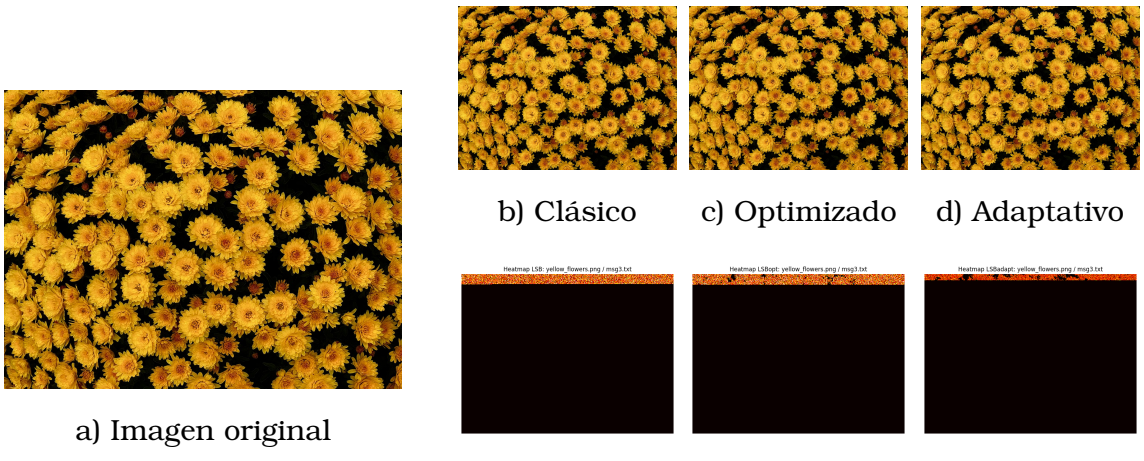
d) Adaptativo

e) Heatmap clásico

f) Heatmap optimizado

g) Heatmap adaptativo

Figura E.2: Comparación visual entre la imagen original `snake.png`, versiones codificadas y sus respectivos mapas de calor con `msg3.txt`.



a) Imagen original

b) Clásico

c) Optimizado

d) Adaptativo

e) Heatmap clásico

f) Heatmap optimizado

g) Heatmap adaptativo

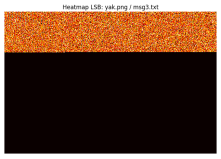
Figura E.3: Comparación visual entre la imagen original `yellow_flowers.png`, versiones codificadas y sus respectivos mapas de calor con `msg3.txt`.

a) Imagen original

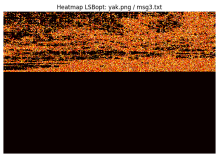
b) Clásico

c) Optimizado

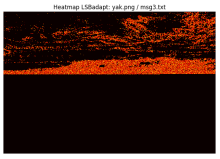
d) Adaptativo



e) Heatmap clásico



f) Heatmap optimizado



g) Heatmap adaptativo

Figura E.4: Comparación visual entre la imagen original `yak.png`, versiones codificadas y sus respectivos mapas de calor con `msg3.txt`.

Apéndice F

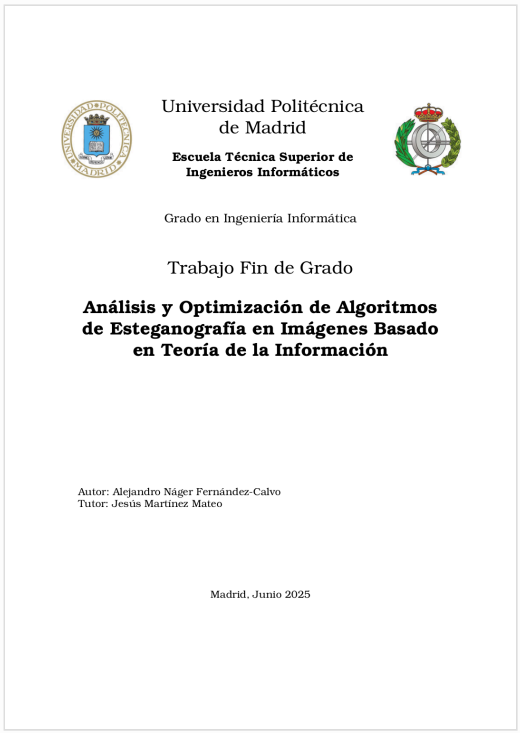
Informe originalidad

Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author:	ALEJANDRO NAGER FERNANDEZ-CALVO
Assignment title:	Turnitin Memoria Final
Submission title:	tfg_etsiinf_AlejndroNagerFernandez-Calvo.pdf
File name:	27929_ALEJANDRO_NAGER_FERNANDEZ-CALVO_tfg_etsiinf_Ale...
File size:	12.88M
Page count:	85
Word count:	18,956
Character count:	99,999
Submission date:	27-May-2025 07:40PM (UTC+0200)
Submission ID:	2686187560



Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Análisis y Optimización de Algoritmos de Esteganografía en Imágenes Basado en Teoría de la Información

Autor: Alejandro Náger Fernández-Calvo
Tutor: Jesús Martínez Mateo

Madrid, Junio 2025

ALEJANDRO NAGER FERNANDEZ-CALVO

tfg_etsiinf_AlejndroNagerFernandez-Calvo.pdf

Turnitin Memoria Final

TFG ETSIINF (Moodle PP)

Universidad Politecnica de Madrid

Detalles del documento

Identificador de la entrega

trn:oid:::1:3261842038

Fecha de entrega

27 may 2025, 7:40 p.m. GMT+2

Fecha de descarga

28 may 2025, 9:54 a.m. GMT+2

Nombre de archivo

27929_ALEJANDRO_NAGER_FERNANDEZ-CALVO_tfg_etsiinf_AlejndroNagerFernandez-Calvo_83714....pdf

Tamaño de archivo

12.9 MB

85 Páginas

18.956 Palabras

99.999 Caracteres

4% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography
- Quoted Text

Top Sources

0% Internet sources
0% Publications
4% Submitted works (Student Papers)

Top Sources

0% Internet sources
0% Publications
4% Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1 Student papers
Universidad Politécnica de Madrid <1%

2 Student papers
Brunel University <1%

3 Student papers
UNIBA <1%

4 Student papers
American University of Central Asia <1%

5 Student papers
University of Queensland <1%

6 Student papers
Heriot-Watt University <1%

7 Student papers
BB9.1 PROD <1%

8 Student papers
Universidad Internacional de la Rioja <1%

9 Student papers
Universitat Politècnica de València <1%

10 Student papers
Georgia Institute of Technology Main Campus <1%

11 Student papers
Universidad de Valladolid <1%

12 Student papers
University of Edinburgh <1%

13 Student papers
University of North Florida <1%

14 Student papers
Royal Holloway and Bedford New College <1%

15 Student papers
Instituto Superior de Artes, Ciencias y Comunicación IACC <1%

16 Student papers
Universidad de Cantabria <1%

17 Student papers
Mohamad Bin Zayed University of Artificial Intelligence <1%

18 Student papers
Pennsylvania State System of Higher Education <1%

19 Student papers
Pontificia Universidad Catolica del Peru <1%

20 Student papers
IUBH - Internationale Hochschule Bad Honnef-Bonn <1%

21 Student papers
Technological Institute of the Philippines <1%

22 Student papers
University of Sydney <1%

23 Student papers
usach <1%

24 Student papers
Corporación Universitaria Minuto de Dios, UNIMINUTO <1%

25 Student papers
Universidad Autónoma de Ciudad Juárez <1%

26 Student papers

Escuela Politecnica Nacional <1%

27 Student papers

Universidad TecMilenio <1%

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Sun Jun 01 19:24:13 CEST 2025
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)