INSTITUTO POLITÉCTICO NACIONAL

ESCUELA SUPERIOR DE COMPUTO

PRÁCTICA 1. "RETO: CONTEO DE OBJETOS EN UNA IMAGEN"

GRUPO:

4BM2

Materia:

Procesamiento digital de imágenes

Presenta:

Arana Acosta Shantal Vanessa

González Estrada Carlos Yael

Herrera Monroy Abraham Andre

Ortiz Santillan Hannia Maybelline

Rojas Muciño Cecilia

Profesor de Materia:

Elena Cruz Meza

México, CDMX. a 07 de noviembre del 2025

INSTITUTO POLITÉCNICO NACIONAL

ESCOM

Contenido

Tabla de Figuras

Resumen

En esta práctica se aplicaron diferentes técnicas de PDI, incluyendo operaciones aritméticas y lógicas, binarización, etiquetado de componentes conexas y detección de contornos. Para ello se trabajó con conjuntos de imágenes: fotografías reales del trolebús, utilizadas para evaluar las operaciones entre dos imágenes y por un escalar; y cuatro imágenes binarias diseñadas específicamente para analizar el comportamiento de las funciones de segmentación y conectividad.

Se implementaron procedimientos como suma, resta y multiplicación de imágenes, así como operaciones lógicas que permitieron combinar o resaltar regiones de interés.

Posteriormente, se llevó a cabo la binarización de las imágenes y su análisis mediante el etiquetado de componentes conexas con vecindad-4 y vecindad-8, lo que posibilitó identificar y contar objetos presentes en la escena. Finalmente, se utilizaron técnicas de detección de contornos para visualizar y numerar los objetos obtenidos en la segmentación.

Introducción

El procesamiento digital de imágenes (PDI) es una disciplina fundamental dentro de la inteligencia artificial, pues permite transformar, analizar y extraer información relevante mediante operaciones matemáticas aplicadas a imágenes digitales. En esta práctica se aplicaron operaciones aritméticas, operaciones lógicas y técnicas de etiquetado de componentes conexas con el propósito de comprender cómo estas transformaciones influyen en la estructura visual de una imagen y cómo pueden emplearse en segmentación y conteo de objetos.

Cruz Meza (2025) establece que la presente práctica tiene como objetivo *"aplicar transformaciones lógicas (AND, OR, XOR, NOT) y operaciones relacionales en imágenes digitales para modificar sus características y extraer información relevante"*. Asimismo, la extensión de la práctica específica el uso del etiquetado de componentes conexas como herramienta para contar objetos, señalando que su objetivo es *"aplicar la técnica de etiquetado de componentes conexas como una estrategia de solución para conteo de objetos en imágenes en Python"*.

Objetivos

Objetivo general

Aplicar operaciones aritméticas, lógicas y el etiquetado de componentes conexas para modificar, segmentar y analizar imágenes digitales.

Objetivos específicos

- Realizar suma, resta, multiplicación y división sobre imágenes digitales.
- Implementar operadores lógicos AND, OR y XOR.
- Aplicar umbralización fija y adaptativa.
- Identificar objetos mediante conectividad 4 y 8.
- Analizar los efectos de cada proceso en la imagen.

Marco teórico

Operaciones Aritméticas en Imágenes

Las operaciones aritméticas permiten combinar imágenes o modificar valores de los píxeles de una imagen. Según Cruz (2025), "las operaciones aritméticas permiten combinar imágenes o modificar valores de los píxeles de una imagen" (p. 2). Estas operaciones son útiles para ajustar el brillo, el contraste o combinar imágenes.

Las operaciones aritméticas más usadas en procesamiento de imágenes son: suma, resta y multiplicación.

Por ejemplo, al sumar un escalar a una imagen, lo que se obtiene es el aumento de brillo en la imagen resultante y el histograma se desplaza a la derecha.

Su fórmula está dada por:

$$g(x,y) = f(x,y) + k$$

donde:

$f(x,y) + k$ es la imagen original

k es el escalar

¿Como funciona? La imagen es una matriz de intensidades (0-255), el escalar se suma a cada pixel individualmente. Para cada pixel: $g(i.j) = f(i,j) + k$

Si tenemos una imagen de 3x3:

100	120	130
80	90	100
50	60	70

Tabla 1. Matriz f (i, j) para suma

Si k=20:

$$g(i.j) = f(i,j) + 20$$

El resultado es:

120	140	150
100	110	120
70	80	90

Tabla 2. Matriz g(i, j), resultado de suma

Los cambios que se generan en la imagen es lo antes mencionado, sube el brillo, toda la imagen se hace más clara por igual, el contraste no cambia (porque todos los pixeles aumentan la misma cantidad).

Si la suma produce valores mayores a 255, pasa lo siguiente:

$$\text{Si } f(i,j) + k > 255 \ \ entonces \ \ g(i,j) = 255$$

$$\text{Si } f(i,j) + k < 0 \ \ entonces \ \ g(i,j) = 0$$

Al saturarse un pixel $(f(i,j) > 255)$ se produce perdida de información.

Por otro lado, la restar de un escalar a una imagen, resta el valor de cada pixel de la imagen el escalar. Como resultado se obtiene una imagen más oscura (disminuye el brillo de la imagen). Al realizar esta operación el histograma se desplaza a la izquierda.

Su fórmula está dada por:

$$g(x,y) = f(x,y) - k$$

donde:

$f(x,y)$ es la imagen original y,

k es el escalar

¿Como funciona? La imagen es una matriz de intensidades (0-255), el escalar se resta a cada pixel individualmente.

Por ejemplo, si tenemos una imagen de 3x3:

120	140	150
100	110	120
70	80	90

Tabla 3. Matriz f (x, y) para resta

Si k=30:

$$g(i.j) = f(i,j) - 30$$

La matriz nueva de la imagen obtenida seria:

90	110	120
70	80	90
40	50	60

Tabla 4. Matriz g (x, y) resultado de Resta

Los cambios que se pueden visualizar es la disminución del brillo de toda la imagen, se hace más oscura, no cambia el contraste porque todos los pixeles bajan por igual.

Al igual que la suma, si la resta produce valores:

$$f(i,j) + k > 255 \ \ entonces \ \ g(i,j) = 255$$

$$f(i,j) + k < 0 \ \ entonces \ \ g(i,j) = 0$$

Por ejemplo: f(i,j) = 20 y k = 50 ------> 20 – 50 = -30 (no válido), por lo tanto, g(i,j) = 0.

Por último, la multiplicación de un escalar a una imagen, multiplica el valor de cada pixel de la imagen por el escalar. Como resultado es la modificación del contraste. Al realizar esta operación el histograma se estira o se comprime.

Su fórmula está dada por:

$$g(x,y) = f(x,y) \cdot k$$

Ya mencionado anteriormente que:

$f(x,y)$ es la imagen original y, k el escalar.

Como la suma y la resta, la imagen es una matriz de intensidades (0-255), el escalar se multiplica a cada pixel individualmente.

Por ejemplo, si tenemos una imagen de 3x3:

120	140	150
100	110	120
70	80	90

Tabla 5. Matriz f (x, y) para multiplicación

Se multiplica por k=1.2

$$g(i.j) = f(i,j) \cdot 1.2$$

La matriz nueva que obtiene es:

144	168	180
120	132	144
84	96	108

Tabla 6. Matriz g(x, y) resultado de multiplicación

Lo que esta operación hace es aumentar las diferencias entre zonas claras y oscuras; hace más clara las partes claras y más oscuras las partes oscuras. Si k < 1, reduce el contraste (imagen apagada). Intensificaría o reduce el contraste.

Al igual que las otras dos operaciones aritméticas más utilizadas, sus condiciones son las mismas:

$$f(i,j) + k > 255 \ \ entonces \ \ g(i,j) = 255$$

$$f(i,j) + k < 0 \ \ entonces \ \ g(i,j) = 0$$

Y como una condición adicional, es que el factor escalar debe ser mayor que 1.

Operaciones relacionales y lógicas

Las operaciones lógicas (AND, OR, XOR, NOT) se aplican principalmente sobre imágenes binarias y permiten combinar regiones, detectar intersecciones, diferencias o complementar objetos. Son indispensables en segmentación y conteo.

- AND: deja solo las áreas donde ambos operandos tienen píxeles con valor 1.
- OR: une regiones de ambas imágenes.
- XOR: resalta diferencias.
- NOT: invierte el valor de cada píxel, transformando blancos a negros y viceversa.

Según Forsyth y Ponce (2012), estas operaciones son esenciales para la morfología matemática y la construcción de máscaras lógicas para análisis estructural.

Cruz Meza (2025) destaca su utilidad en la práctica argumentando que *"las operaciones lógicas son útiles para combinar imágenes según condiciones específicas"*.

El operador AND toma como entrada dos imágenes con valores binarios, el resultado es una tercera imagen binaria donde un pixel tendrá el valor de 1 si y solo si ambos pixeles correspondientes en las imágenes de entrada son 1. . En cualquier otro caso el píxel resultante será 0.

$$AND(a,b) = \begin{Bmatrix} 1 \quad si\ las\ dos\ entradas\ son\ 1 \\ 0 \quad si\ las\ entradas\ son\ diferentes\ de\ 1 \end{Bmatrix}$$

Si las 2 imágenes de entrada se consideran conjuntos, se diría que el operador AND, es su intersección, siendo su tabla de verdad

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 7. Tabla de verdad de AND

El siguiente ejercicio es un ejemplo de su aplicación.

Sea la imagen A, y la imagen B, aplicar el operador AND y obtener la nueva imagen AB.

Como se muestra en la figura,

Imagen A				
1	0	0	1	1
0	1	1	0	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1

A)

Imagen B				
0	0	1	1	0
1	0	1	0	1
1	0	0	1	1
0	1	0	1	0
1	1	0	0	0

B)

A B				
0	0	0	1	0
0	0	1	0	1
1	0	0	0	1
0	0	0	1	0
1	0	0	0	0

C)

Tabla 8. Matrices de imagen A, Imagen B aplicando AND

En la figura se muestra como

$$AB = A\,(x, y) * B\,(x, y)$$

El operador OR toma como entrada dos imágenes con valores binarios, el resultado es una tercera imagen binaria donde un píxel tendrá el valor de 1 si al menos una de las entradas es 1. En cualquier otro caso el píxel resultante será 0.

$$OR(a,b) = \left\{\begin{matrix} 1 \;\; si\; almenos\; una\; de\; las\; entradas\; es\; 1 \\ 0 \;\; si\; las\; dos\; entradas\; son\; 0 \end{matrix}\right\}$$

Si las 2 imágenes de entrada se consideran conjuntos, se diría que el operador OR, es su union, siendo su tabla de verdad la siguiente

A	B	A AND B
0	0	0
0	1	1
1	0	1
1	1	1

Tabla 9. Tabla de verdad de OR

El siguiente ejercicio es un ejemplo de su aplicación.

Sea la imagen A, y la imagen B, aplicar el operador OR y obtener la nueva imagen A + B.

Como se muestra en la figura,

Imagen A				
1	0	0	1	1
0	1	1	0	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1

A)

Imagen B				
0	0	1	1	0
1	0	1	0	1
1	0	0	1	1
0	1	0	1	0
1	1	0	0	0

B)

A + B				
1	0	1	1	1
1	1	1	0	1
1	1	0	1	1
0	1	1	1	0
1	1	1	0	0

C)

Tabla 10. Matrices de imagen A, Imagen B aplicando OR

En la figura se muestra como

$$A + B = A(x, y) + B(x, y)$$

El operador XOR, se utiliza para comparar, detectar diferencias o extraer información.

El operador XOR, toma como entrada dos imágenes de las cuales por lo menos una debe ser binaria, genera una tercera imagen que replica los valores de la imagen original en las posiciones de pixeles donde la máscara está en bajo (0), y negativiza los valores en las posiciones de pixeles donde la máscara está en alto (1). Si se trabaja con dos imágenes binarias el operador XOR se entiende como la unión menos la intersección

Esto permite detectar: objetos movidos, cambios entre fotogramas, errores de transmisión y manipulación en imágenes.

Recordando la tabla de verdad para una operación XOR es la siguiente:

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 11. Tabla de verdad de XOR

Supongamos dos imágenes binarias, sus respectivas matrices son:

Imagen A				
1	0	0	1	1
0	1	1	0	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1

A)

Imagen B				
0	0	1	1	0
1	0	1	0	1
1	0	0	1	1
0	1	0	1	0
1	1	0	0	0

B)

Tabla 12. Matrices de imágenes A y B

Se aplica XOR elemento por elemento. Por ejemplo:

Posición (0,0): $1 \oplus 0 = 1$

Posición (0,1): $0 \oplus 0 = 0$

Posición (0,2): $0 \oplus 1 = 1$

Asi con todas las posiciones, hasta obtener:

A ⊕ B				
1	0	1	0	1
1	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	0	1

Tabla 13. Matriz de A XOR B

El operador NOT toma como entrada 1 imágen con valores binarios, el resultado es una imagen binaria donde un píxel tendrá el valor de 1 si y solo si su entrada es 0 y viceversa.

$$NOT(a) = \left\{\frac{1 \quad si\ su\ entrada\ es\ 0}{0 \quad si\ su\ entrada\ es\ 1}\right\}$$

Si la imagen de entrada se considera un conjunto, se dice que el NOT es su complemento , siendo su tabla de verdad la siguiente

A	NOT A
1	0
0	1

Tabla 14. Tabla de verdad de NOT

El siguiente ejercicio es un ejemplo de su aplicación.

Sea la imagen A, y la imagen B, aplicar el operador OR y obtener la nueva imagen A + B.

Como se muestra en la figura,

Imagen A				
1	0	0	1	1
0	1	1	0	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1

A)

NOT A				
0	1	1	0	0
1	0	0	1	1
0	0	1	1	0
1	1	0	0	1
0	1	0	1	0

B)

Tabla 15. Matriz de imagen A y su complemento

Binarización y segmentación

La binarización es un proceso de segmentación donde la imagen se reduce a dos valores: fondo y objeto. Este proceso generalmente se realiza mediante umbral fijo o adaptativo. La segmentación adecuada es crucial porque constituye la base para las siguientes etapas de análisis, especialmente en operaciones lógicas y etiquetado.

Umbralización

La umbralización es una técnica de segmentación que convierte una imagen en escala de grises a una imagen binaria. Según Cruz (2025):

"Aplicar umbralizado simple a una imagen antes de aplicar las operaciones aritmético-lógicas. Esto se debe a que el umbralizado convierte una imagen en una versión binaria basada en un valor umbral asegurando que la imagen de entrada en efecto es binaria, de este modo, al aplicar los operadores relacionales y lógicos se tendrán resultados adecuados" (p. 3).

Etiquetado de Componentes Conexas

El etiquetado de componentes conexas es un proceso que identifica y numera regiones conectadas en una imagen binaria. La conectividad puede definirse como vecindad-4 (ortogonal) o vecindad-8 (incluyendo diagonales). Cruz (2025) explica:

"La vecindad 8 generalmente es más precisa para definir regiones conectadas porque permite detectar conectividades más amplias, evitando que los objetos con conexiones en diagonal o curvas se etiqueten como elementos separados. La vecindad 4 es ideal cuando quieres limitar la conexión a píxeles ortogonales" (p. 2).

Además, según la misma autora, "el etiquetado de componentes conexas en una imagen binaria utilizando Python. Este enfoque utiliza scipy.ndimage.label para aplicar tanto la vecindad 4 como la vecindad 8" (Cruz, 2025, p. 2).

Histograma

Autores como Karl Pearson y ASQ concuerdan con que es una representación gráfica de la distribución de frecuencias de datos cuantitativos. Se utiliza para visualizar la dispersión y

la frecuencia de los datos en intervalos, permitiendo identificar patrones y comprender la forma de la distribución.

En el caso de las imágenes se utiliza para graficar la distribución de colores aplicados en una imagen en un intervalo de 0 a 255 donde 0 se toma como color negro y el 255 se visualiza como el color blanco, se lee respecto a los ejes el horizontal representa los tonos y en vertical la cantidad de pixeles de cada tonalidad.

Descripción del conjunto de imágenes

Para esta práctica se utilizaron dos tipos de imágenes. El primero corresponde a cuatro imágenes binarias formadas por diferentes figuras geométricas y símbolos en blanco y negro como se puede observar en la Figura 1. Estas imágenes fueron creadas para evaluar el funcionamiento de los algoritmos de componentes conexas y detección de contornos, ya que permiten identificar con claridad regiones independientes y analizar cómo la conectividad afecta el número de objetos detectados.

El segundo conjunto está compuesto por dos fotografías reales de un trolebús, las cuales se emplearon en las secciones de operaciones aritméticas y lógicas entre imágenes. Estas fotografías permiten observar los efectos de las distintas transformaciones sobre imágenes con contenido real, iluminación variable y múltiples detalles visuales. En conjunto, ambos tipos de imágenes proporcionan material adecuado para explorar tanto técnicas básicas de manipulación de imágenes como procedimientos de segmentación y análisis estructural.

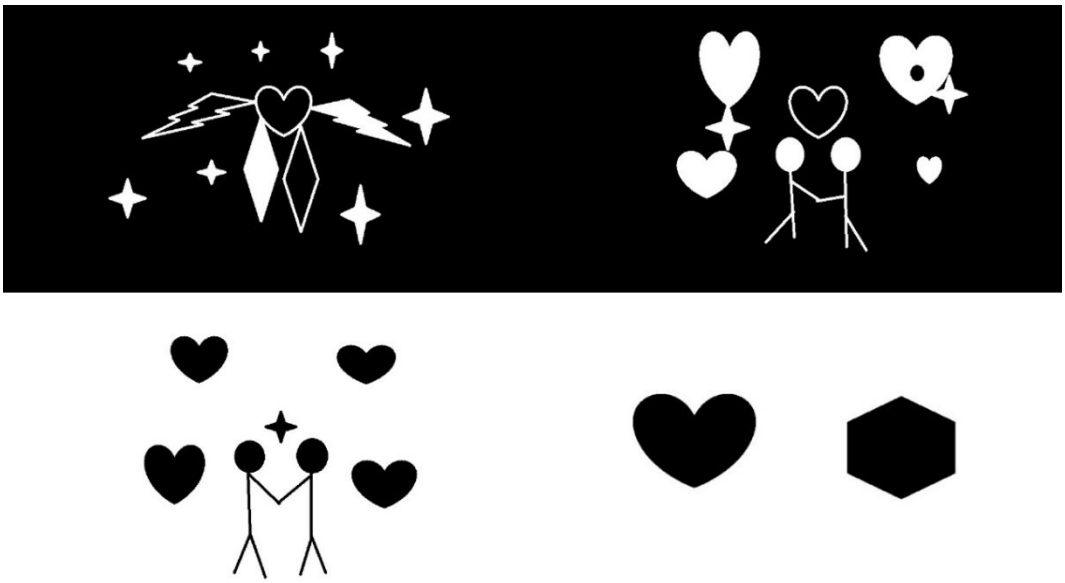


Figura 1: Imágenes generadas en Paint.

Figura 2: Fotografías del trolebús.

Desarrollo

Operaciones relacionales y lógicas por un escalar y entre imágenes

Operaciones aritméticas con un escalar aplicadas a una imagen binaria

En la **Figura 3** se muestra un fragmento del código encargado de gestionar las operaciones aritméticas mediante las funciones arithmetic_operation_scalar y arithmetic_operation_image. Estas funciones permiten aplicar suma, resta y multiplicación, ya sea utilizando un valor constante (escalar) o bien empleando una segunda imagen como operando. Su finalidad es modificar los valores de los píxeles de la imagen actualmente en procesamiento de acuerdo con la operación seleccionada.

```
# ---------------- operaciones aritméticas ----------------
def arithmetic_operation_scalar(self, operation):
    src = self.processed_image if self.processed_image is not None else self.original_image
    if src is None:
        messagebox.showwarning("Advertencia", "Primero carga una imagen.")
        return

    # si la imagen es binaria: permitir entrada pero no cambiar visualmente
    if self.binary_image is not None and self.processed_image is not None and np.array_equal(self.processed_image, self.binary_image):
        if operation == 'add':
            value = self.get_numeric_input("Suma con Escalar", "Ingresa el valor a sumar a cada píxel:", 50)
            if value is None: return
            self.set_message(f"Suma con escalar +{value} solicitada sobre imagen binarizada (sin cambios visibles).")
        elif operation == 'subtract':
            value = self.get_numeric_input("Resta con Escalar", "Ingresa el valor a restar a cada píxel:", 50)
            if value is None: return
            self.set_message(f"Resta con escalar -{value} solicitada sobre imagen binarizada (sin cambios visibles).")
        elif operation == 'multiply':
            value = self.get_numeric_input("Multiplicación con Escalar", "Ingresa el factor de multiplicación:", 1.2)
            if value is None: return
            self.set_message(f"Multiplicación con escalar x{value} solicitada sobre imagen binarizada (sin cambios visibles).")
        self.display_image(self.processed_image, self.processed_canvas)
        self.show_histogram_auto()
        return
```

Figura 3: Fragmento de código correspondiente a la implementación de operaciones aritméticas con un escalar y entre imágenes mediante las funciones arithmetic_operation_scalar y arithmetic_operation_image.

Operaciones aritméticas con una imagen binaria y un escalar

El proceso comienza seleccionando la imagen de origen, priorizando la imagen procesada, el código incluye una lógica de protección que detecta si la imagen actual es binarizada, si es así le permite al usuario ingresar el valor escalar, pero no producirá un cambio significativo como se muestra en la **figura 4,** esto ocurre debido a que la suma aritmética de una imagen binaria con un valor escalar no es posible, pues la imagen binaria tiene valores únicamente blancos y negros, (1 y 0 respectivamente).

Figura 4: Imagen resultante de aplicar la operación de suma con un escalar sobre la imagen binarizada previamente mostrada en la Figura 2.

Operaciones aritméticas con una imagen en color/grises y un escalar

En el fragmento de código mostrado en la **Figura 5**, se observa la lógica principal utilizada para aplicar operaciones aritméticas —suma, resta y multiplicación— sobre una imagen en escala de grises o en color utilizando un **valor escalar**. El bloque está estructurado mediante una serie de condiciones if y elif que verifican qué operación ha solicitado el usuario y ejecutan la función correspondiente de OpenCV.

```
# pedir valor y aplicar sobre imagen original
if operation == 'add':
    value = self.get_numeric_input("Suma con Escalar", "Ingresa el valor a sumar a cada píxel:", 50)
    if value is None: return
    self.processed_image = cv2.add(src, value)
    self.set_message(f"Suma con escalar +{value} aplicada.")
elif operation == 'subtract':
    value = self.get_numeric_input("Resta con Escalar", "Ingresa el valor a restar a cada píxel:", 50)
    if value is None: return
    self.processed_image = cv2.subtract(src, value)
    self.set_message(f"Resta con escalar -{value} aplicada.")
elif operation == 'multiply':
    value = self.get_numeric_input("Multiplicación con Escalar", "Ingresa el factor de multiplicación:", 1.2)
    if value is None: return
    res = cv2.multiply(src.astype(np.float32), float(value))
    self.processed_image = np.clip(res,0,255).astype(np.uint8)
    self.set_message(f"Multiplicación con escalar x{value} aplicada.")
self.display_image(self.processed_image, self.processed_canvas)
self.show_histogram_auto()
```

Figura 5: Fragmento de código para aplicar operaciones aritméticas con un escalar sobre imágenes en color y en escala de grises.

En el caso de la suma (representada como "add"), el programa solicita un valor escalar y posteriormente aplica la operación mediante cv2.add(src, value). Esta función realiza una suma a nivel de píxel, manejando adecuadamente la saturación: si el resultado excede 255, el valor se mantiene en 255. El efecto visual puede observarse en las **Figuras 6-a** (imagen en

color) y **6-b** (imagen en escala de grises), donde el incremento en el brillo es evidente en ambas representaciones.

a)

b)

Figura 6: Resultados de la operación de suma con un escalar aplicada a la imagen del trolebús en color (a) y en escala de grises (b).

Si revisamos el histograma de la figura mostrado en la **Figura 7**, donde se aprecia el **desplazamiento uniforme de los valores de intensidad** hacia la derecha. Este fenómeno ocurre porque al sumar un escalar fijo, todos los píxeles de la imagen aumentan su valor en la misma proporción, modificando el brillo de manera homogénea. Dicho desplazamiento confirma el comportamiento esperado en las operaciones aritméticas realizadas a nivel de

intensidad en cada uno de los canales (RGB en color o nivel de gris en imágenes monocromáticas).

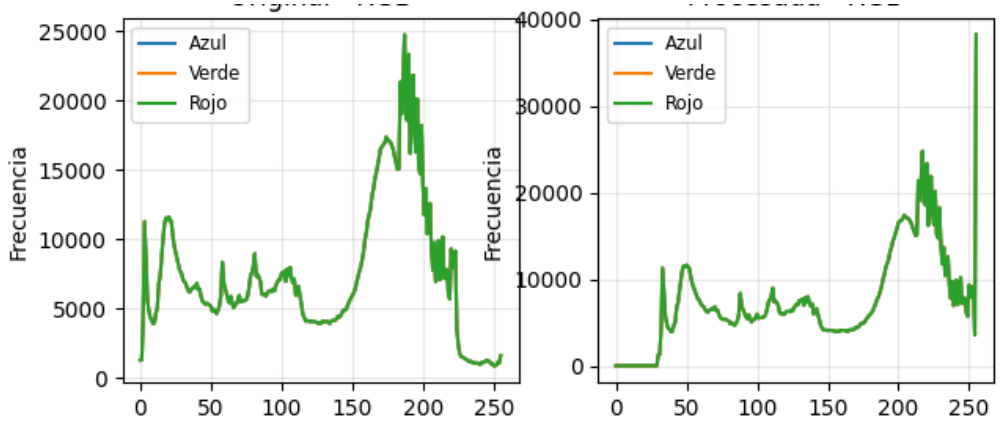


Figura 7: Histogramas de la Figura 2 y la imagen procesada tras la operación de suma con un escalar

La operación de **resta** (denominada "subtract" en el código) solicita al usuario un valor escalar y posteriormente aplica cv2.subtract(src, value). Esta función realiza una resta por píxel y, al igual que en la suma, OpenCV implementa un manejo automático de saturación: cuando el resultado es negativo, el valor se ajusta a 0. Este efecto puede observarse en el ejemplo presentado en la Figura #, donde la disminución uniforme de intensidad genera una imagen visualmente más oscura.

a)

b)

Figura 8: Aplicación de la resta aritmética con un escalar a una imagen gris y color.

En el caso de la **multiplicación**, el procedimiento requiere un manejo más cuidadoso debido a que esta operación puede incrementar rápidamente los valores de intensidad. Primero, la imagen original src se convierte a tipo de dato float32, ya que cv2.multiply opera de manera más precisa y flexible en este formato. Una vez realizada la multiplicación por el factor escalar solicitado, el resultado es recortado (clipping) para garantizar que ningún píxel exceda el rango permitido de 0 a 255. Finalmente, la imagen se convierte nuevamente a tipo uint8 para su visualización y uso dentro del programa.

El resultado de esta operación se muestra en la **Figura 9**, donde es evidente el incremento del contraste y brillo derivado del factor multiplicativo aplicado.

a)

b)

Figura 9: Aplicación de la multiplicación aritmética con un escalar a una imagen gris y color.

Operaciones aritméticas entre 2 imágenes

El fragmento de código mostrado en la **Figura 10** corresponde a la función arithmetic_operation_images, cuya finalidad es iniciar el proceso para aplicar una operación aritmética —suma, resta o multiplicación— entre la imagen actualmente en uso y una segunda imagen cargada por el usuario.

```
def arithmetic_operation_images(self, operation):
    imgA = self.processed_image if self.processed_image is not None else self.original_image
    if imgA is None:
        messagebox.showwarning("Advertencia", "Primero carga la imagen 1.")
        return
    if self.image2 is None:
        messagebox.showwarning("Advertencia", "Para esta operación necesitas cargar la imagen 2.")
        return

    # si binaria
    if self.binary_image is not None and self.processed_image is not None and np.array_equal(self.processed_image, self.binary_image):
        self.set_message("Operación aritmética entre imágenes solicitada sobre imagen binarizada (sin cambios visibles).")
        self.display_image(self.processed_image, self.processed_canvas)
        self.show_histogram_auto()
        return

    h1,w1 = imgA.shape[:2]
    h2,w2 = self.image2.shape[:2]
    h,w = min(h1,h2), min(w1,w2)
    A = cv2.resize(imgA, (w,h))
    B = cv2.resize(self.image2, (w,h))

    A_gray = (A.ndim == 2)
    B_gray = (B.ndim == 2)
```

Figura 10: Fragmento de código de la función arithmetic_operation_images para realizar operaciones aritméticas entre dos imágenes.

El procedimiento comienza con una serie de verificaciones que garantizan el funcionamiento correcto del sistema. En primer lugar, el método comprueba que **ambas imágenes hayan sido cargadas adecuadamente**; en caso contrario, se muestra un mensaje de advertencia y el proceso se detiene. Además, al igual que en otras funciones del programa, se incluye una **lógica de protección** que identifica si la imagen en proceso es binaria. Si esto ocurre, se evita

continuar con la operación para prevenir resultados inválidos o un comportamiento inesperado del programa.

Una vez validado que ambas imágenes son aptas para el procesamiento, el siguiente paso consiste en **normalizar sus dimensiones**. Para ello, se obtienen las alturas y anchos de ambas imágenes y se determina el tamaño común más pequeño. Posteriormente, ambas imágenes se redimensionan mediante cv2.resize para que coincidan en estas dimensiones. Esta normalización es esencial, ya que las operaciones aritméticas en OpenCV requieren que las imágenes involucradas tengan el mismo tamaño para realizar cálculos píxel a píxel de manera correcta y consistente.

El fragmento de código mostrado en la **Figura 11** corresponde a la segunda parte del método arithmetic_operation_images, cuya función es **homogeneizar los formatos de ambas imágenes** y posteriormente **aplicar la operación aritmética seleccionada** por el usuario.

```
    if A_gray and not B_gray:
        B_proc = cv2.cvtColor(B, cv2.COLOR_BGR2GRAY)
        A_proc = A
    elif not A_gray and B_gray:
        B_proc = cv2.cvtColor(B, cv2.COLOR_GRAY2BGR)
        A_proc = A
    else:
        A_proc = A
        B_proc = B

    try:
        if operation == 'add':
            result = cv2.add(A_proc, B_proc); self.set_message("Suma de imágenes aplicada.")
        elif operation == 'subtract':
            result = cv2.subtract(A_proc, B_proc); self.set_message("Resta de imágenes aplicada.")
        elif operation == 'multiply':
            result = cv2.multiply(A_proc, B_proc); self.set_message("Multiplicación de imágenes aplicada.")
        else:
            messagebox.showerror("Error", "Operación desconocida.")
            return
        if result.dtype != np.uint8:
            result = np.clip(result,0,255).astype(np.uint8)
        self.processed_image = result
        self.display_image(self.processed_image, self.processed_canvas)
        self.show_histogram_auto()
    except Exception as e:
        messagebox.showerror("Error", f"No se pudo realizar la operación: {e}")
```

Figura 11: Fragmento de la segunda parte del método arithmetic_operation_images donde se homogeneizan los formatos de color y se aplican las operaciones aritméticas entre dos imágenes.

Primero, el código maneja la compatibilidad de canales de color entre las dos imágenes redimensionadas, si la Imagen A está en escala de grises y la Imagen B no, convierte la Imagen B a escala de grises para que ambas imágenes tengan un solo canal y puedan operarse,

si la imagen A está a color y la Imagen B está en escala de grises, convierte la imagen B a color para que ambas tengan tres canales.

Si ambas imágenes ya son compatibles, se usan directamente sin conversión. Después de asegurar la compatibilidad, el código ejecuta la operación aritmética elegida (suma, resta o multiplicación), en caso de que el tipo de dato resultante no sea compatible, el código recorta los valores para asegurarse de que permanezcan en el rango 0-255 y los convierte a enteros.

El funcionamiento de las operaciones es muy similar a su ejecución con un escalar, pero toman como el escalar, a el valor del píxel que estén trabajando, en el caso de las sumas, se suman los valores de los 2 pixeles y si se supera el límite de valor de 255, se queda ahí, de la misma manera con la resta y multiplicación; en la **figura 12** se muestran los resultados.

a)

b)

c)

Figura 12: Resultados de las operaciones aritméticas entre las dos imágenes del trolebús: suma (a), resta (b) y multiplicación (c).

Operaciones aritméticas entre 2 imágenes binarias

El siguiente experimento se realizó utilizando la función logical_operation, mostrada en la **Figura 13**, la cual tiene como propósito aplicar operaciones lógicas a nivel de bit entre imágenes binarias. Este método permite ejecutar las operaciones AND, OR, XOR y NOT, empleando las funciones bitwise de OpenCV.

```
# ----------------- operaciones lógicas -----------------
def logical_operation(self, operation):
    if self.original_image is None:
        messagebox.showwarning("Advertencia", "Primero carga la imagen 1.")
        return
    if operation != 'not' and self.image2 is None:
        messagebox.showwarning("Advertencia", "Para esta operación necesitas cargar la imagen 2.")
        return

    if operation != 'not':
        h1,w1 = self.original_image.shape[:2]
        h2,w2 = self.image2.shape[:2]
        h,w = min(h1,h2), min(w1,w2)
        A = cv2.resize(self.original_image, (w,h))
        B = cv2.resize(self.image2, (w,h))
        if operation == 'and':
            res = cv2.bitwise_and(A,B); self.set_message("Operación: AND aplicada.")
        elif operation == 'or':
            res = cv2.bitwise_or(A,B); self.set_message("Operación: OR aplicada.")
        elif operation == 'xor':
            res = cv2.bitwise_xor(A,B); self.set_message("Operación: XOR aplicada.")
        else:
            messagebox.showerror("Error", "Operación desconocida.")
            return
        self.processed_image = res
    else:
        self.processed_image = cv2.bitwise_not(self.original_image)
        self.set_message("Operación: NOT aplicada.")
    self.display_image(self.processed_image, self.processed_canvas)
    self.show_histogram_auto()
```

Figura 13: Fragmento de la función logical_operation utilizada para aplicar operaciones lógicas entre imágenes binarias.

La función comienza con verificaciones de imagen, se asegura que la imagen original este cargada. Para las operaciones binarias **AND, OR y XOR**, también verifica que la imagen 2 esté disponible; si falta alguna de ellas se emite una advertencia.

Si la operación es otra que **NOT**, el código procede con las operaciones, redimensiona las imágenes para asegurar la compatibilidad y luego dependiendo de la operación solicitada, utiliza las funciones de OpenCV correspondientes para calcular el resultado.

Si la operación solicitada es **NOT**, se aplica *cv2.bitwise:not* directamente a la imagen, invirtiendo el valor de los pixeles como se muestra en la **figura 14**.

Figura 14: Resultado de la operación lógica NOT aplicada a la imagen binaria del trolebús.

La operación **AND** al igual que las operaciones aritméticas trabaja píxel a píxel, esta operación resulta en un píxel blanco -de valor 255 representado como 1 en binario- en la imagen de salida si el píxel correspondiente es blanco en ambas imágenes de entrada, de otra forma, el píxel de salida será negro -valor de 0 representado como 0 en binario- como se muestra en la **Figura 15.**

Figura 15: Resultado de la operación lógica AND aplicada a la imagen binaria del trolebús.

La operación **OR** resulta en un píxel blanco en a la imagen de salida si el píxel correspondiente es blanco en cualquiera de las dos imágenes de entrada, solo resultara en negro si ambos son negros, demostrado en la **Figura 16**.

Figura 16: Resultado de la operación lógica OR aplicada a la imagen binaria del trolebús.

Por último, la operación **XOR o OR Exclusiva**, resulta en un píxel blanco solo si los pixeles correspondientes de cada imagen son diferentes, es decir, uno debe ser blanco y otro negro, pero nunca los 2 negros ni los 2 blancos, en caso de que sean así, el pixel resultante será negro como se muestra en la **Figura 17**.

Figura 17: Resultado de la operación lógica XOR aplicada a la imagen binaria del trolebús.

Se ha de mencionar que es posible aplicar las operaciones lógicas con imágenes no binarias, contrario a lo que se pensaría, pero su funcionamiento es mucho mas complicado y resulta en una imagen con valores menos predecibles.

Por ejemplo, al momento de comparar una imagen a color, cada uno de sus canales posee valores no binarios, los métodos ofrecidos por OpenCV manejan estos casos de la siguiente manera; Obtienen el valor binario de cualquier valor no binario del pixel, si es a color , cada canal obtendrá su propio valor binario, lo mismo con la segunda imagen, una vez que las 2 imágenes tienen sus valores binarios, se aplica el operador, siguiendo las reglas antes mencionadas para cada bit, resultando ,por ejemplo, en la **Figura 18**.

Figura 18: Operación AND aplicada a 2 imágenes a color.

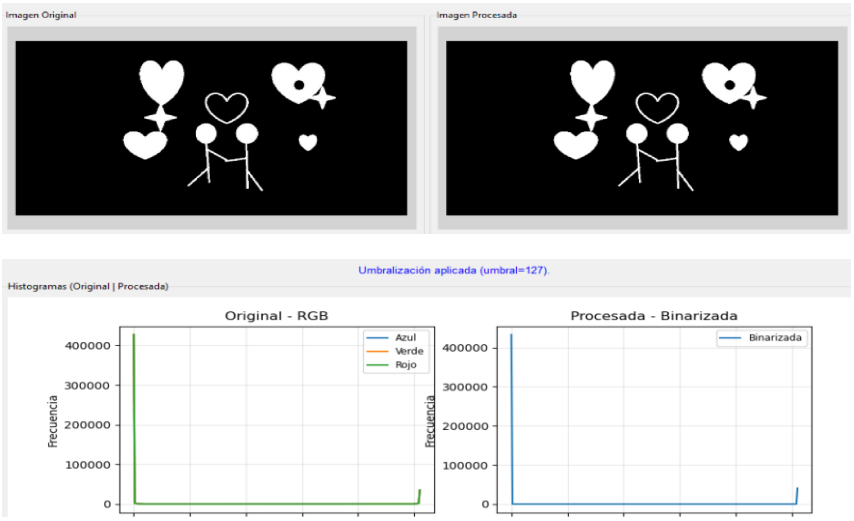
Etiquetado de componentes conexas

Carga y Binarización de la imagen

En esta sección del código (Figura **19.b**) se realiza la lectura y preparación inicial de la imagen que será utilizada para el análisis de componentes conexas. La imagen se carga directamente en **escala de grises** mediante la instrucción cv2.imread utilizando la bandera IMREAD_GRAYSCALE, lo cual permite trabajar únicamente con la intensidad de los píxeles. Tras la carga, el programa verifica que el archivo haya sido leído correctamente, evitando errores posteriores en caso de rutas incorrectas o imágenes faltantes.

Una vez cargada, se aplica una **binarización con umbral fijo** utilizando la función cv2.threshold, donde se establece un valor de **umbral = 127**. En esta operación, todos los píxeles con intensidad mayor al umbral se convierten en 255 (blanco), mientras que los inferiores pasan a 0 (negro). Este proceso transforma la imagen en una **máscara binaria**, simplificando la información visual y permitiendo identificar fácilmente las regiones independientes u objetos presentes en la escena.

La **Figura 19.a** muestra el resultado de esta binarización, donde la imagen ha sido reducida a únicamente dos valores de intensidad, lo cual es esencial para las etapas posteriores de etiquetado y detección de contornos.



a)

```python
# Importar librerías
import cv2
import numpy as np
import matplotlib.pyplot as plt

# ----- 1. Cargar y binarizar la imagen -----
# Cargar imagen en escala de grises
image = cv2.imread('imagen_prueba.png', cv2.IMREAD_GRAYSCALE)

# Verificar si la imagen fue cargada correctamente
if image is None:
    print("Error al cargar la imagen.")
    exit()

# ---Umbralización para binarizar la imagen---
_, binary_image = cv2.threshold(image, thresh: 127, maxval: 255, cv2.THRESH_BINARY)

# Mostrar imagen binarizada
plt.figure(figsize=(6, 6))
plt.imshow(binary_image, cmap='gray')
plt.title('Imagen Binarizada')
plt.axis('off')
plt.show()
```

b)

Figura 19: Imagen binarizada obtenida mediante umbral fijo (a) y fragmento de código correspondiente al proceso de carga y binarización (b).

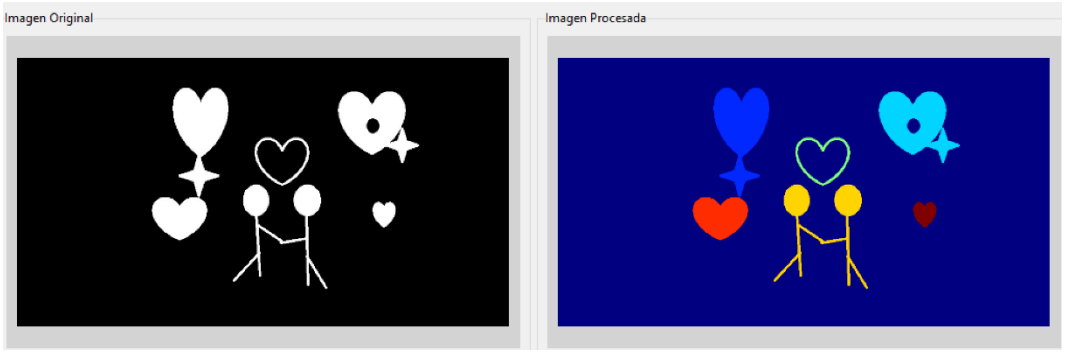
Etiquetado de Componentes Conexas (Vecindad-4 y Vecindad-8)

En esta sección del código (Figura **20.c**) se realiza el etiquetado de componentes conexas utilizando la función cv2.connectedComponents, cuyo propósito es identificar y separar los objetos independientes presentes en la imagen binaria. Este proceso asigna una etiqueta numérica diferente a cada región conectada de píxeles con valor 255, permitiendo distinguirlos claramente entre sí.

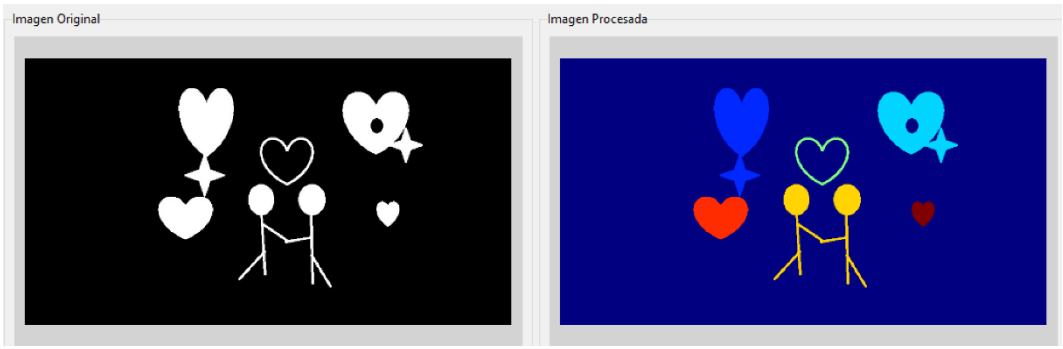
Primero se aplica el etiquetado usando **vecindad-4**, donde cada píxel se considera conectado únicamente con sus vecinos superior, inferior, izquierdo y derecho. El resultado se muestra en la **Figura 20.a**, donde cada región recibe un color distinto en el mapa de etiquetas (cmap='jet'), facilitando la visualización de los objetos detectados.

Posteriormente se repite el proceso utilizando **vecindad-8**, donde además se incluyen conexiones diagonales. Este tipo de conectividad es más flexible, por lo que puede unir regiones que con vecindad-4 permanecerían separadas. La salida correspondiente se observa en la **Figura 20.b**, donde se aprecia la diferencia en la segmentación y el número de objetos detectados.

El programa también imprime la cantidad de regiones encontradas (sin contar el fondo), permitiendo comparar la sensibilidad del etiquetado bajo ambos criterios de conectividad.



a)



b)

```python
# ----- 2. Etiquetado de componentes conexas -----
# Vecindad-4
num_labels_4, labels_4 = cv2.connectedComponents(binary_image, connectivity=4)

# Vecindad-8
num_labels_8, labels_8 = cv2.connectedComponents(binary_image, connectivity=8)

print(f"Número de objetos detectados con vecindad-4: {num_labels_4 - 1}")
print(f"Número de objetos detectados con vecindad-8: {num_labels_8 - 1}")

# Mostrar resultados de vecindad-4
plt.figure(figsize=(6, 6))
plt.imshow(labels_4, cmap='jet')
plt.title('Etiquetado con Vecindad-4')
plt.colorbar()
plt.axis('off')
plt.show()

# Mostrar resultados de vecindad-8
plt.figure(figsize=(6, 6))
plt.imshow(labels_8, cmap='jet')
plt.title('Etiquetado con Vecindad-8')
plt.colorbar()
plt.axis('off')
plt.show()
```

c)

Figura 20: Etiquetado de componentes conexas utilizando vecindad-4 (a), vecindad-8 (b) y fragmento del código correspondiente (c).

Detección de Contornos y Numeración de Objetos

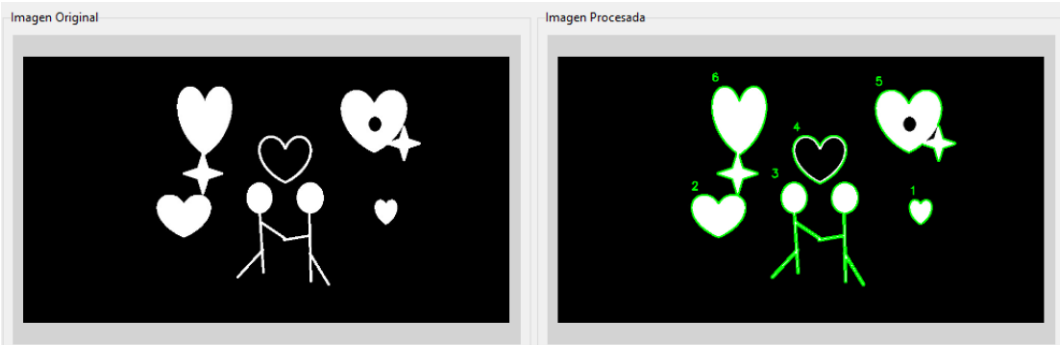
En esta sección (Figura **21.c**) se realiza la detección de contornos sobre la imagen binaria con el objetivo de identificar visualmente los objetos detectados y numerarlos para facilitar su inspección. Primero, la máscara binaria se convierte a una imagen a color mediante cv2.cvtColor, lo que permite dibujar los contornos en un color distintivo.

Los contornos se obtienen usando cv2.findContours con el modo RETR_EXTERNAL, que recupera únicamente los contornos externos de los objetos. Luego, cada contorno se dibuja en color verde sobre la imagen (cv2.drawContours) y se calcula su caja delimitadora (cv2.boundingRect) para colocar un número identificador utilizando

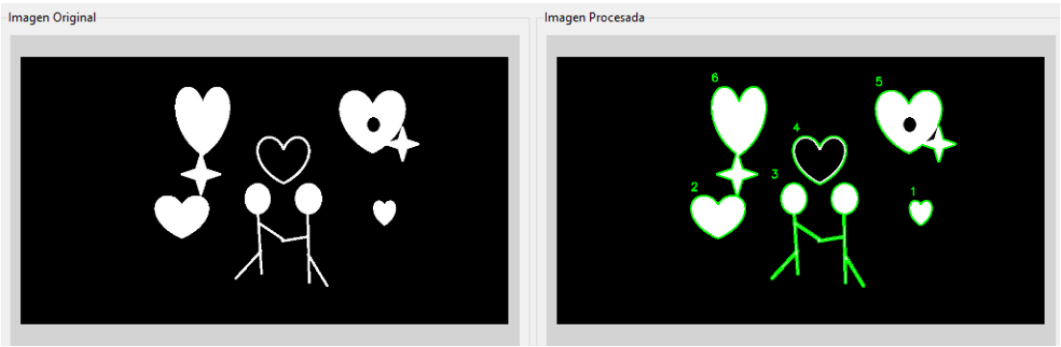
cv2.putText. Esto permite diferenciar rápidamente cada objeto y confirmar visualmente el número de regiones detectadas.

La **Figura 21.a y Figura 21.b** muestran los resultados obtenidos al aplicar este procedimiento sobre las imágenes etiquetadas con vecindad-4 y vecindad-8, respectivamente. Aunque ambas imágenes provienen de la misma binarización, las diferencias en conectividad influyen en el número de objetos detectados: la vecindad-4 tiende a separar regiones que solo se tocan en diagonales, mientras que la vecindad-8 las considera conectadas y, por tanto, genera menos contornos. La enumeración permite verificar visualmente estas diferencias.

Finalmente, en la **Figura 21.c** se presenta el fragmento de código responsable de este proceso.



a)



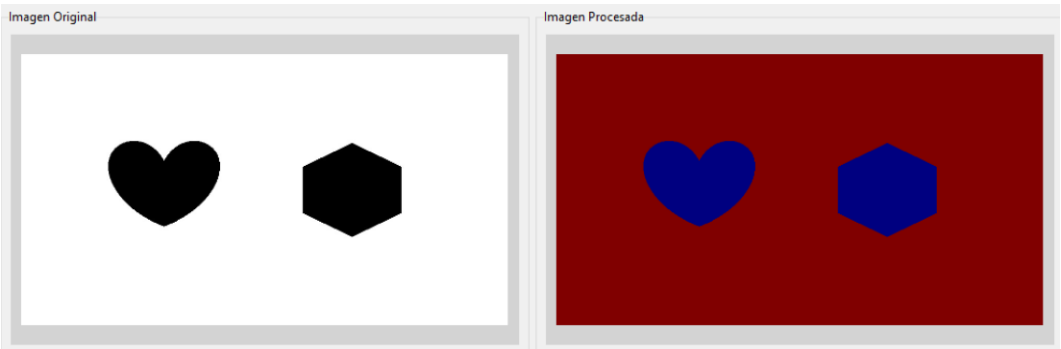
b)

```python
# ----- 3. Dibujar contornos y numerar los objetos -----
# Convertir imagen binaria a imagen en color para dibujar contornos
image_color = cv2.cvtColor(binary_image, cv2.COLOR_GRAY2BGR)

# Encontrar contornos en la imagen (corregido para diferentes versiones de OpenCV)
contours, _ = cv2.findContours(binary_image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Dibujar los contornos y numerar los objetos
for i, contour in enumerate(contours):
    # Dibujar contorno (color verde)
    cv2.drawContours(image_color, contours: [contour], -1, color: (0, 255, 0), thickness: 2)

    # Encontrar el centro del objeto y colocar el número
    x, y, w, h = cv2.boundingRect(contour)
    cv2.putText(image_color, text: f'{i + 1}', org: (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, fontScale

# Mostrar la imagen con los contornos y etiquetas
plt.figure(figsize=(6, 6))
plt.imshow(cv2.cvtColor(image_color, cv2.COLOR_BGR2RGB))
plt.title('Objetos Detectados y Numerados')
plt.axis('off')
plt.show()

# ----- 4. Comparación entre vecindad-4 y vecindad-8 -----
diferencia = abs(num_labels_4 - num_labels_8)
print(f"Diferencia entre vecindad-4 y vecindad-8: {diferencia}")
```

c)

Figura 21: Contornos detectados y numeración de objetos vecindad-4 (a), vecindad-8 (b) y fragmento del código utilizado (c).

Nota: Durante el procesamiento de imágenes mediante binarización y componentes conexas, es común que surja un comportamiento particular: cuando se utiliza una imagen con **fondo blanco** y objetos en negro, el programa interpreta al **fondo como un objeto**. Esto ocurre debido a la forma en que OpenCV maneja la binarización y el etiquetado de regiones conectadas.



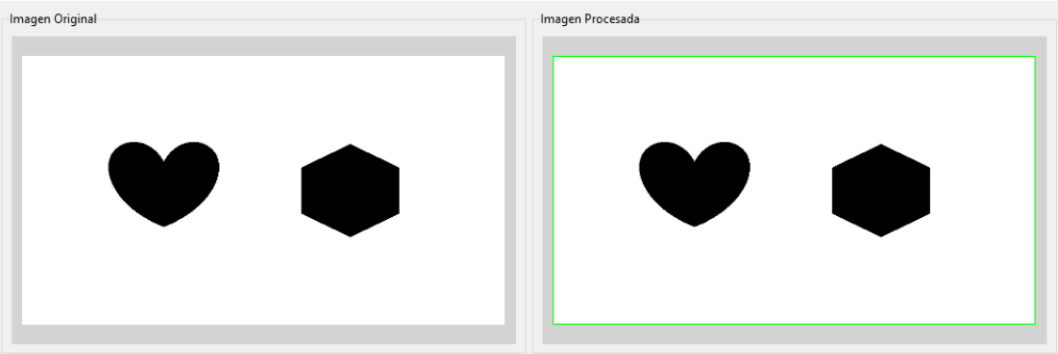


Figura 22: Imagen con fondo blanco.

Análisis de Resultados

Los resultados obtenidos a lo largo de esta practica permiten evaluar diversas técnicas de PDI aplicadas a imágenes reales e imágenes binarias diseñadas para pruebas.

En primer lugar, **las operaciones aritméticas** por un escalar mostraron efectos sobre la intensidad de los pixeles. La suma genero un incremento uniforme del brillo, y esto se vio reflejado en el desplazamiento hacia la derecha del histograma. La resta produjo un oscurecimiento en la imagen al reducir los valores de intensidad, mientras que la multiplicación permitió modificar el contraste dependiendo del factor utilizado. En todos los casos, OpenCV manejó correctamente la saturación, manteniendo los valores dentro del rango permitido (0–255).

Asimismo, se observó que cuando estas operaciones se aplican sobre imágenes binarias no producen cambios visibles, debido a la naturaleza limitada de estos datos (solo 0 y 255), lo cual confirma la validez de las protecciones implementadas en el código.

En las **operaciones aritméticas entre dos imágenes**, los resultados muestran combinaciones visuales correspondientes a la suma, resta y multiplicación píxel a píxel. La calidad de los resultados dependió directamente de la correcta normalización de tamaños entre ambas imágenes, lo cual fue gestionado por el programa mediante redimensionamiento automático. Este proceso aseguró que las operaciones se realizaran adecuadamente sin generar errores por incompatibilidad de dimensiones.

Por otro lado, las **operaciones lógicas** aplicadas sobre imágenes binarias permitieron observar patrones claros de cómo interactúan los valores de píxel entre dos imágenes

- AND conserva únicamente las regiones donde ambas imágenes comparten píxeles blancos,
- OR combina las zonas blancas de ambas imágenes,
- XOR resalta las diferencias entre ellas.
- La operación NOT invirtió correctamente los valores de fondo y figura, demostrando su utilidad como herramienta de realce o preprocesamiento.

Finalmente, el **etiquetado de componentes conexas** permitió identificar y contar los objetos presentes en cada imagen. Al comparar la vecindad-4 y la vecindad-8, se observó que esta última tiende a agrupar un mayor número de píxeles dentro de un mismo objeto, produciendo diferencias en el conteo final. Esto confirma la importancia de seleccionar correctamente el tipo de conectividad según la aplicación deseada. La posterior **detección de contornos y numeración** mostró de forma visual los objetos segmentados, validando los resultados del etiquetado y facilitando su interpretación.

Reflexión

1. ¿Qué tipo de diferencias fueron detectadas?

Dependiendo del tipo de imagen, su comportamiento frente a las operaciones lógicas y aritméticas varía. Las imágenes del primer conjunto son limpias, sin ruido y con formas bien definidas, mientras que en imágenes reales —como las del segundo conjunto— su aplicación es más complicada debido a la naturaleza propia de la imagen.

Otras diferencias notorias y lógicas se observan en el estado de la imagen después de la operación utilizada. Como se comentó en la práctica, la suma —…, la resta —…, la multiplicación —… y la división —… generan efectos distintos en el resultado final.

2. ¿Cómo influye el umbral en la sensibilidad de la comparación?

El umbralizado es una parte esencial, ya que con un umbral definido obtenemos la binarización de la imagen. Si observamos el desarrollo de la práctica, vemos que es más sencillo obtener los bordes de una imagen con base negra y objetos blancos.

3. ¿Qué aplicaciones prácticas puede tener esta técnica en Visión por Computadora o en IA?

Poder obtener los bordes de los objetos en una imagen crea un espacio de oportunidad para la inteligencia artificial, ya que permite capturar la información de la imagen y enviarla a una configuración donde esta se procesa. Un ejemplo, como se propuso en la práctica, sería una imagen de un trolebús donde se obtiene su forma e identifica su entorno.

4. ¿Cómo se podría mejorar el análisis para detectar cambios más sutiles?

Esto puede mejorar mediante técnicas de suavizado de ruido, evitando que se unan objetos. Una imagen más clara ayudaría a mejorar el análisis, aunque en la vida real esto es menos probable. Por ello, es necesario recurrir a tratamientos externos para

evitar la combinación de objetos (como mapas de colores para diferenciar estructuras de distintos colores, por ejemplo).

Conclusión

La realización de esta práctica permitió obtener una visión más amplia del proceso básico del procesamiento digital de imágenes, desde las transformaciones iniciales hasta el análisis estructural mediante segmentación y etiquetado. Las operaciones aritméticas y lógicas mostraron cómo es posible modificar las características visuales de una imagen, resaltar zonas de interés o combinar información proveniente de dos fuentes distintas. Del mismo modo, al aplicar la técnica de umbralización demostró la importancia de una correcta binarización, ya que este paso es determinante para obtener resultados precisos en las etapas posteriores.

El uso del etiquetado de componentes conexas, tanto con vecindad-4 como con vecindad-8, permitió identificar y cuantificar objetos dentro de una imagen, evidenciando cómo la definición de la conectividad influye directamente en el número de regiones detectadas. Posteriormente, la detección de contornos y la numeración de objetos confirmaron visualmente la segmentación lograda, proporcionando una herramienta práctica de verificación.

Está práctica no solo fortaleció el entendimiento técnico de cada procedimiento, sino que también mostró la relación entre ellos y su relevancia en aplicaciones más complejas como la visión por computadora, el conteo automático, la inspección de patrones y el análisis de estructuras. Finalmente, el trabajo realizado evidencia que el procesamiento digital de imágenes requiere tanto precisión en la implementación como criterio en la elección de técnicas, ya que pequeñas variaciones en umbralización, conectividad o preprocesamiento pueden modificar completamente el resultado final.

Metacognición

Durante el desarrollo de esta práctica pude reconocer con claridad cómo ha ido evolucionando mi forma de aprender dentro del área de procesamiento digital de imágenes. Al inicio, muchos de los conceptos los entendía únicamente desde la teoría, pero no conseguía visualizar cómo cada transformación afectaba realmente la estructura de una imagen. Conforme avancé en la implementación y observé los resultados directamente en pantalla, noté que mi comprensión pasó de ser descriptiva a analítica.

Algo que identifiqué sobre mi propio proceso de aprendizaje es que suelo confiar demasiado en mi intuición antes de validar los resultados con evidencia visual. Por ejemplo, inicialmente pensé que aplicar operaciones aritméticas a una imagen binaria generaría cambios visibles, y solo al ver los resultados entendí por qué no ocurría: la naturaleza de los valores (0 y 255) limita completamente su comportamiento. Esta confrontación entre lo que creo entender y lo que realmente ocurre se convirtió en un punto clave para ajustar mi forma de estudiar.

También descubrí que comprender el por qué detrás de cada función de OpenCV me permitió corregir errores más rápido. Antes tendía a probar código por ensayo y error, pero en esta práctica comencé a analizar más cuidadosamente la lógica interna: saturación, compatibilidad de canales, redimensionamiento, manejo de tipos de datos, etc. Esa transición me hizo sentir más seguro y autónomo al programar.

Otro aprendizaje importante fue darme cuenta de que, aunque las herramientas de visión por computadora parecen técnicas, en realidad requieren mucho pensamiento crítico: cuestionar si la binarización fue adecuada, si la conectividad elegida afecta el resultado, si un contorno realmente representa un objeto o es ruido. Entendí que no se trata solo de ejecutar funciones, sino de interpretar lo que significan y cómo influyen en el análisis final.

Finalmente, concluí que esta práctica fortaleció no solo mi comprensión técnica, sino mi capacidad para reflexionar sobre mi propio desempeño. Aprendí que antes de buscar soluciones automáticas, debo asegurarme de comprender el fundamento matemático y visual de cada operación. Esta forma más consciente y autorregulada de aprender es algo que llevo como una mejora real en mi manera de enfrentar problemas dentro de la inteligencia artificial y el procesamiento de imágenes.

Bibliografía

Gonzalez, R. C., & Woods, R. E. (2018). Digital Image Processing (4th ed.). Pearson.

Forsyth, D. A., & Ponce, J. (2012). Computer Vision: A Modern Approach (2nd ed.). Pearson.

Pratt, W. K. (2007). Digital Image Processing: PIKS Scientific Inside (4th ed.). Wiley.

OpenCV. (2023). *OpenCV Documentation*. https://docs.opencv.org/

OpenCV. (2023). *Image Processing Module*. https://docs.opencv.org/4.x/d7/d00/tutorial_meanshift.html

OpenCV. (2023). *connectedComponents — Connected Components Labeling*. https://docs.opencv.org/4.x/d3/dc0/group__imgproc__shape.html

Bradski, G., & Kaehler, A. (2008). *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media.

Matplotlib Contributors. (2023). *Matplotlib Documentation*. https://matplotlib.org/stable/

NumPy Developers. (2023). *NumPy Documentation*. https://numpy.org/doc/