Sistemas de Percepción en Robótica

Tema: Programación en Python de transformaciones basadas en el Histograma.

Nombre: Yoamy de la Caridad Agüero Nocedo

Introducción

En este trabajo se aplican técnicas básicas de procesamiento digital de imágenes sobre la imagen 'lena_color.tiff'. El objetivo es generar y documentar los resultados de las siguientes operaciones: rotación, negativo, umbralización, ecualización y eliminación de ruido. Se utiliza Python con OpenCV y NumPy para implementar los algoritmos y generar los archivos resultantes.

Desarrollo

Materiales y Métodos (incluye requisitos e instrucciones de ejecución):

Entorno:

- Python 3.8+.
- Librerías necesarias: opencv-python, numpy, matplotlib.

Instalación:

- pip install opencv-python numpy matplotlib

Archivos incluidos:

- procesamiento_'lena_color.tiff' (script que realiza todas las operaciones)
- 'lena_color.tiff'.png (imagen de entrada)
- 'lena_color.tiff'_rotada.png
- 'lena_color.tiff'_negativa.png
- 'lena_color.tiff'_umbralizada.png
- 'lena_color.tiff'_ecualizada.png
- 'lena_color.tiff'_sin_ruido.png

Instrucciones para ejecutar:

1. Colocar 'lena_color.tiff' y 'procesamiento_lena_color.tiff'en la misma carpeta.
2. Ejecutar en terminal: python procesamiento_'lena_color.tiff'
3. El script mostrará las imágenes y guardará los archivos resultantes en la misma carpeta.

Descripción de los algoritmos:

Rotación:

Método: Matriz de transformación afín.

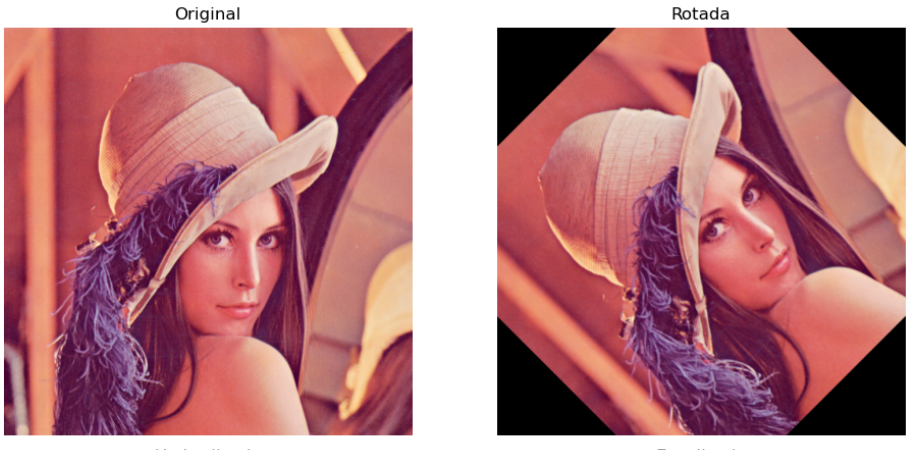
Descripción: Se usó la función cv2.getRotationMatrix2D(center, angle, scale) y cv2.warpAffine para rotar la imagen 45° alrededor de su centro. El tamaño de salida se mantuvo igual a la imagen original.

Parámetro usado: angúlo = 45 grados.

Código:

```
# 1. Imagen rotada
(h, w) = img.shape[:2]
centro = (w // 2, h // 2)
M = cv2.getRotationMatrix2D(centro, 45, 1.0)  # Rotar 45 grados
rotada = cv2.warpAffine(img, M, (w, h))
```

Resultado:



Negativo:

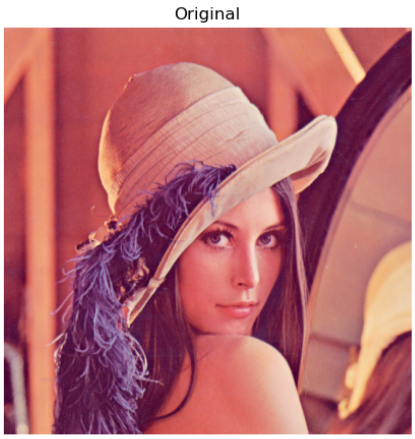
Método: Transformación punto a punto.

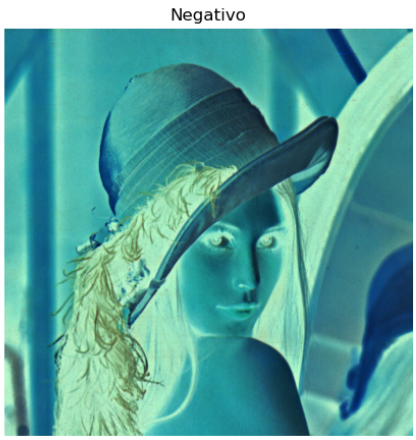
Descripción: Se realizó la operación negativa aplicando 255 - I a cada canal (I es el valor del píxel). Esto invierte la intensidad de color creando el negativo.

Código:

```
# 2. Negativo de la imagen
negativo = 255 - img
```

Resultado:





Umbralización (binarización):

Método: Thresholding global.

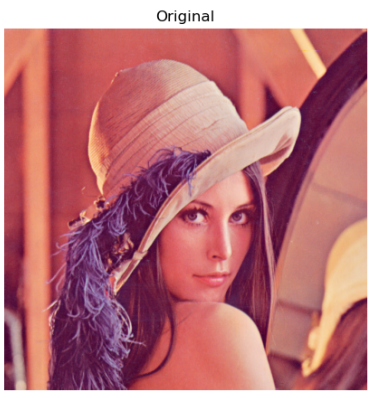
Descripción: Se convirtió la imagen a escala de grises y se aplicó cv2.threshold con un umbral fijo de 127 (valores >=127 → 255; <127 → 0).

Parámetro usado: THRESH_BINARY, threshold = 127.

Código:

```
# 3. Imagen umbralizada (binarización)
_, umbralizada = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)
```

Resultado:



Ecualización:

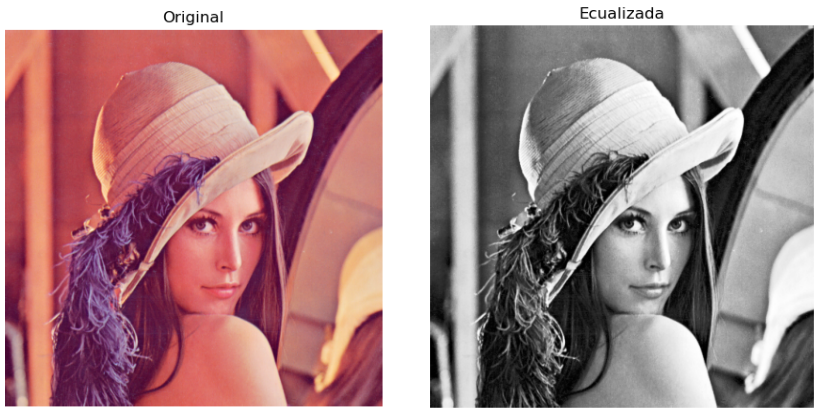
Método: Ecualización del histograma (cv2.equalizeHist).

Descripción: Se aplicó a la imagen en escala de grises para distribuir mejor las intensidades y mejorar el contraste.

Código:

```
# 4. Imagen ecualizada (mejora de contraste)
ecualizada = cv2.equalizeHist(gray)
```

Resultado:



Eliminación de ruido:

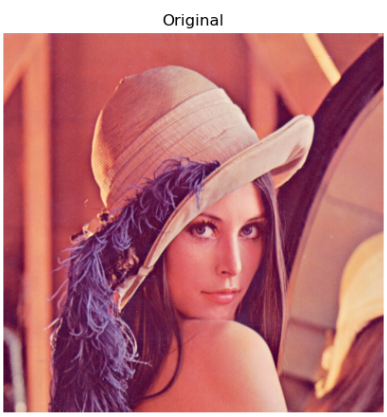
Método: Filtro Gaussiano.

Descripción: Se utilizó cv2.GaussianBlur con kernel (5,5) para suavizar la imagen y reducir ruido de alta frecuencia.

Código:

```
# 5. Eliminación de ruido (filtro Gaussiano)
sin_ruido = cv2.GaussianBlur(img, (5, 5), 0)
```

Resultado:



Análisis:

Los resultados muestran el comportamiento esperado de cada operación. La rotación mantiene la estructura general, el negativo invierte las intensidades, la umbralización segmenta la imagen en dos niveles, la ecualización mejora el contraste general y el filtro Gaussiano reduce ruido a costa de suavizar detalles finos. Para futuros trabajos se podría explorar: rotación con ajuste de lienzo para evitar recortes, umbral adaptativo, ecualización por contraste limitado (CLAHE) y filtros más avanzados de reducción de ruido (p. ej., medianas o Non-Local Means).

Conclusiones

Se implementaron correctamente las cinco operaciones solicitadas usando Python y OpenCV. Los archivos resultantes se generan y están incluidos. El trabajo demuestra los efectos básicos de cada técnica sobre una imagen estándar.