



Rodrigo Hernández Zavala

15 de marzo de 2025

Ing. Mecatrónica

22310215

Umbralización

Visión Artificial

Explicación del Código

Este código usa **OpenCV** para aplicar diferentes métodos de **umbralización** (**thresholding**) a una imagen (`bookpage.jpg`), lo que convierte la imagen en blanco y negro resaltando características importantes.

Paso a Paso del Código

Importar Librerías


```
python
CopiarEditar
import cv2
import numpy as np
```

- ◆ **cv2 (OpenCV)** → Para procesamiento de imágenes.
 - ◆ **numpy (np)** → Para manipulación de matrices de píxeles.
-

Cargar la Imagen

```
python
CopiarEditar
img = cv2.imread('bookpage.jpg')
```

- ◆ **Carga la imagen "bookpage.jpg" en color** (`cv2.IMREAD_COLOR` por defecto).

 **Posible mejora:** Verificar si la imagen se cargó correctamente.

```
python
CopiarEditar
if img is None:
    print("Error: No se pudo cargar la imagen.")
    exit()
```

Aplicar Umbralización Binaria a la Imagen en Color

```
python
CopiarEditar
retval, threshold = cv2.threshold(img, 12, 255, cv2.THRESH_BINARY)
```

- ◆ **`cv2.threshold(img, 12, 255, cv2.THRESH_BINARY):`**

- Convierte **cada píxel** en blanco (255) si su valor es **mayor** a 12, de lo contrario lo convierte en negro (0).
- Este método **no funciona bien en imágenes en color**, por lo que generalmente se aplica en escala de grises.

✅ **Mejora:** Convertir la imagen a escala de grises antes de aplicar umbralización.

4 Convertir la Imagen a Escala de Grises

```
python
CopiarEditar
grayscaled = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

- ◆ Convierte `img` en escala de grises para mejorar la umbralización.
-

5 Aplicar Umbralización Binaria en Escala de Grises

```
python
CopiarEditar
retval12, threshold2 = cv2.threshold(grayscaled, 12, 255,
cv2.THRESH_BINARY)
```

- ◆ Aplica el mismo método de umbralización pero ahora en **escala de grises**.
 - ◆ La imagen umbralizada ahora tendrá solo **blancos y negros puros**.
-

6 Aplicar Umbralización Adaptativa (Gaussiana)

```
python
CopiarEditar
gaus = cv2.adaptiveThreshold(grayscaled, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 115, 1)
```

- ◆ Usa `cv2.adaptiveThreshold()` con el método **Gaussiano**, que ajusta el umbral localmente en diferentes partes de la imagen.
- ◆ Parámetros:
 - 255 → Máximo valor (blanco).
 - `cv2.ADAPTIVE_THRESH_GAUSSIAN_C` → Método que usa una media gaussiana en cada región.
 - `cv2.THRESH_BINARY` → Convierte a blanco/negro según el umbral.

- 115 → Tamaño del bloque (debe ser **impar**).
- 1 → Constante restada del promedio calculado en cada región.

✓ **Ventaja:** Funciona bien en imágenes con **iluminación desigual**.

❌(Error) Aplicar Umbralización con Otsu

```
python
CopiarEditar
retval2, otsu = threshold(graycaled, 125, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)
```

- ◆ **Error:** `threshold()` está mal escrito, la función correcta es `cv2.threshold()`.
- ◆ Corrige la línea así:

```
python
CopiarEditar
retval2, otsu = cv2.threshold(graycaled, 125, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)
```

- ◆ **Método Otsu:**
 - **Calcula automáticamente el mejor umbral** para separar los píxeles en **blanco y negro**.
 - Se usa cuando **no se conoce el mejor valor de umbral**.

✓ **Ventaja:** Automáticamente encuentra el mejor umbral sin que el usuario lo defina.

❌Mostrar las Imágenes Resultantes

```
python
CopiarEditar
cv2.imshow('original', img)
cv2.imshow('threshold', threshold)
cv2.imshow('threshold2', threshold2)
cv2.imshow('gaus', gaus)
cv2.imshow('otsu', otsu)
```

- ◆ Muestra cada imagen en una ventana diferente.
- ◆ **Resultados esperados:**
 - "threshold" → No funciona bien porque `cv2.threshold()` fue aplicado a una imagen en color.

- "threshold2" → Imagen binaria en blanco y negro con umbral de 12.
 - "gaus" → Umbral adaptativo que mejora detalles en zonas con distinta iluminación.
 - "otsu" → Usa un umbral automático óptimo.
-

Esperar una Tecla y Cerrar Ventanas

```
python
CopiarEditar
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- ◆ **Espera hasta que el usuario presione una tecla para cerrar todas las ventanas.**
-

Objetivos de la Práctica

- ✓ **Aplicar diferentes métodos de umbralización para convertir imágenes en binarias.**
 - ✓ **Entender la diferencia entre umbralización global (`cv2.threshold`) y adaptativa (`cv2.adaptiveThreshold`).**
 - ✓ **Utilizar el método de Otsu para calcular umbrales automáticamente.**
 - ✓ **Comparar cómo cada método afecta la calidad de la imagen binarizada.**
-

Posibles Mejoras

- ◆ **Corregir el error `threshold()` → `cv2.threshold()`.**
- ◆ **Usar imágenes con texto o alto contraste (ideal para mejorar el umbral).**
- ◆ **Permitir que el usuario ajuste el umbral manualmente:**

```
python
CopiarEditar
user_threshold = int(input("Ingrese un valor de umbral: "))
retval, user_thresh = cv2.threshold(grayscaled, user_threshold, 255,
cv2.THRESH_BINARY)
cv2.imshow('Umbral Personalizado', user_thresh)
```

- ◆ **Guardar las imágenes resultantes:**

```
python
CopiarEditar
```

```
cv2.imwrite('resultado_threshold.jpg', threshold)
cv2.imwrite('resultado_otsu.jpg', otsu)
```

Conclusión

- ✦ Este código demuestra cómo aplicar umbralización para convertir imágenes en blanco y negro, resaltando información importante.
- ✦ Diferentes métodos (`THRESH_BINARY`, `THRESH_OTSU`, `ADAPTIVE_THRESH_GAUSSIAN_C`) tienen ventajas según la imagen y su iluminación.
- ✦ Puede mejorarse permitiendo ajustes manuales de umbral o usando imágenes con más detalles.