CENTRO DE ENSEÑANZA TÉCNICA INDUSTRIAL

INGENIERÍA EN MECATRÓNICA

Visión artificial

Práctica 5:
Umbrales

Alumna:
Vanessa Aguirre Diaz

Registro:
22310274

Fecha:
26 de abril del 2025

Desarrollo teórico:

Umbralización

La umbralización es una técnica de OpenCV que consiste en la asignación de valores de píxel en relación con el valor umbral proporcionado. En la umbralización, cada valor de píxel se compara con dicho valor. Si el valor de píxel es menor que el umbral, se establece en 0; de lo contrario, se establece en un valor máximo (generalmente 255). La umbralización es una técnica de segmentación muy popular, utilizada para separar un objeto considerado como primer plano de su fondo. Un umbral es un valor que tiene dos regiones a cada lado, es decir, por debajo o por encima del umbral.

En visión artificial, esta técnica de umbralización se aplica a imágenes en escala de grises. Por lo tanto, inicialmente, la imagen debe convertirse al espacio de color de escala de grises.

Umbralizaciones Básicas (cv2.threshold())

1. **cv2.THRESH_BINARY**
   → Píxeles mayores al umbral se vuelven blancos (255), los demás se vuelven negros (0).
2. **cv2.THRESH_BINARY_INV**
   → Inverso del anterior: píxeles mayores al umbral se vuelven negros (0), los demás blancos (255).
3. **cv2.THRESH_TRUNC**
   → Los valores mayores al umbral se reducen al valor del umbral, los menores se mantienen igual.
4. **cv2.THRESH_TOZERO**
   → Los valores menores o iguales al umbral se vuelven negros (0), los mayores se mantienen igual.
5. **cv2.THRESH_TOZERO_INV**
   → Los valores mayores al umbral se vuelven negros (0), los menores se mantienen igual.

Umbralizaciones Adaptativas (cv2.adaptiveThreshold())

6. **cv2.ADAPTIVE_THRESH_MEAN_C**
   → Calcula el promedio de un bloque de píxeles vecinos y le resta C. Se adapta a la iluminación local.
7. **cv2.ADAPTIVE_THRESH_GAUSSIAN_C**
   → Igual que el anterior, pero usa una media ponderada Gaussiana del bloque y le resta C.

Umbralización Automática

8. **cv2.THRESH_OTSU**
   → Calcula automáticamente el mejor valor de umbral para separar fondo y letras (o figura principal).
   Se usa en combinación con cv2.THRESH_BINARY.

El método de Otsu es una técnica adaptativa y automatizada que determina el valor umbral óptimo minimizando la varianza en la intensidad de los píxeles. Esta técnica funciona mejor con imágenes bimodales donde existe una clara separación entre la intensidad del objeto y la del fondo.

Para aplicar el umbral de Otsu se debe:

- Calcular el histograma de intensidades de píxeles.
- Calcular las varianzas dentro de la clase y entre clases para cada umbral.
- Seleccione el umbral que minimice la variación dentro de la clase.

El método de Otsu es eficaz en la automatización, ya que no requiere entrada manual para seleccionar el valor umbral y puede producir una segmentación altamente precisa en entornos controlados.

Aplicaciones

La segmentación por umbrales es una técnica versátil aplicable a una amplia gama de campos. Analicemos algunos casos reales donde la segmentación basada en umbrales resulta invaluable.

1. Imágenes médicas: En imágenes médicas , la umbralización se utiliza ampliamente para segmentar estructuras anatómicas, como huesos, tejidos y órganos. Por ejemplo, las tomografías computarizadas pueden usar la umbralización para separar diferentes densidades, resaltando áreas como fracturas óseas, tumores o lesiones para facilitar el diagnóstico.
2. Análisis de documentos: El análisis de documentos se beneficia enormemente de la umbralización, especialmente en el reconocimiento óptico de caracteres (OCR). La umbralización convierte los documentos a formato binario, aislando el texto del ruido de fondo, lo que mejora la precisión del OCR al centrarse únicamente en el texto.
3. Vehículos autónomos: En la conducción autónoma , la umbralización ayuda a los vehículos a detectar los bordes de la carretera, las marcas de carril y los obstáculos. Al transformar imágenes a formatos binarios simplificados, los vehículos pueden identificar objetos rápidamente, lo que mejora la capacidad de toma de decisiones en tiempo real.
4. Control de calidad industrial: Las aplicaciones industriales utilizan la umbralización para la detección de defectos en las líneas de producción. Al aislar los productos del entorno, la umbralización ayuda a identificar irregularidades o defectos, lo que permite a los fabricantes automatizar los controles de calidad de forma eficiente.

Desarrollo práctico

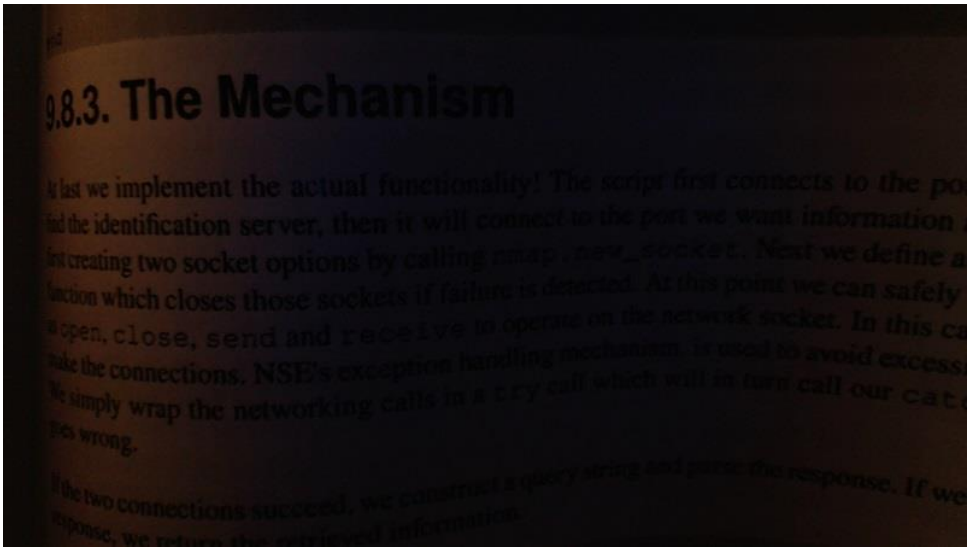
Tenemos en el código principal todos los umbrales solicitados en la descripción de la práctica:

```
#Practica 5: Umbrales

#Importar librerías
import numpy as np                # Librería para manejo de arreglos y matrices
import matplotlib.pyplot as plt   # Para mostrar imágenes gráficamente
import cv2                        # OpenCV: procesamiento de imágenes

# ---------------------------------------
# 1. Cargar imagen y convertir a escala de grises
# ---------------------------------------
img = cv2.imread('bookpage.jpg')                    # Cargar imagen en color
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)        # Convertir a escala de grises

# ---------------------------------------
# 2. Definir parámetros para umbrales fijos
# ---------------------------------------
thresh_val = 127    # Umbral manual
max_val = 255       # Valor máximo (blanco)

# ---------------------------------------
# 3. Aplicar técnicas de umbral fijo
# ---------------------------------------

# Threshold1: THRESH_BINARY
_, binary = cv2.threshold(gray, thresh_val, max_val, cv2.THRESH_BINARY)
# Si el valor del píxel ≥ 127 → se convierte en 255, si no → se convierte en 0 (blanco y
negro)

# THRESH_BINARY_INV
_, binary_inv = cv2.threshold(gray, thresh_val, max_val, cv2.THRESH_BINARY_INV)
# Igual que el anterior pero invertido: lo que era blanco ahora es negro, y viceversa

# THRESH_TRUNC
_, trunc = cv2.threshold(gray, thresh_val, max_val, cv2.THRESH_TRUNC)
# Si el valor del píxel > 127 → se convierte en 127, si no → se queda igual

# THRESH_TOZERO
_, tozero = cv2.threshold(gray, thresh_val, max_val, cv2.THRESH_TOZERO)
# Si el valor del píxel < 127 → se convierte en 0, si no → se mantiene

# THRESH_TOZERO_INV
_, tozero_inv = cv2.threshold(gray, thresh_val, max_val, cv2.THRESH_TOZERO_INV)
# Si el valor del píxel > 127 → se convierte en 0, si no → se mantiene
```

```
# ---------------------------------------
# 4. Aplicar umbrales adaptativos y Otsu
# ---------------------------------------

# ADAPTIVE_THRESH_MEAN_C (Mean)
mean = cv2.adaptiveThreshold(
    gray, max_val, cv2.ADAPTIVE_THRESH_MEAN_C,
    cv2.THRESH_BINARY, 11, 2)
# Calcula el umbral para cada bloque 11x11 como el promedio de sus vecinos menos 2

# ADAPTIVE_THRESH_GAUSSIAN_C (Gaus)
gaus = cv2.adaptiveThreshold(
    gray, max_val, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
    cv2.THRESH_BINARY, 11, 2)
# Similar al anterior pero con una media ponderada gaussiana (valores del centro pesan
más)

# THRESH_OTSU
otsu_val, otsu = cv2.threshold(
    gray, 0, max_val, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
# Encuentra automáticamente el mejor valor de umbral para separar fondo y objetos


# ---------------------------------------
# 5. Mostrar resultados - Ventana 1 (umbrales fijos)
# ---------------------------------------
plt.figure("Umbrales Fijos", figsize=(10, 6))

plt.subplot(2, 2, 1)
plt.imshow(binary, cmap='gray')
plt.title("THRESH_BINARY")
plt.axis('off')

plt.subplot(2, 2, 2)
plt.imshow(binary_inv, cmap='gray')
plt.title("THRESH_BINARY_INV")
plt.axis('off')

plt.subplot(2, 2, 3)
plt.imshow(trunc, cmap='gray')
plt.title("THRESH_TRUNC")
plt.axis('off')

plt.subplot(2, 2, 4)
plt.imshow(tozero, cmap='gray')
plt.title("THRESH_TOZERO")
```

```
plt.axis('off')

# ---------------------------------------
# 6. Mostrar resultados - Ventana 2 (adaptativos y Otsu)
# ---------------------------------------
plt.figure("Adaptativos y Otsu", figsize=(10, 6))

plt.subplot(2, 2, 1)
plt.imshow(tozero_inv, cmap='gray')
plt.title("THRESH_TOZERO_INV")
plt.axis('off')

plt.subplot(2, 2, 2)
plt.imshow(mean, cmap='gray')
plt.title("ADAPTIVE_THRESH_MEAN_C")
plt.axis('off')

plt.subplot(2, 2, 3)
plt.imshow(gaus, cmap='gray')
plt.title("ADAPTIVE_THRESH_GAUSSIAN_C")
plt.axis('off')

plt.subplot(2, 2, 4)
plt.imshow(otsu, cmap='gray')
plt.title(f"THRESH_OTSU (T={otsu_val:.0f})")  # Mostrar valor calculado por Otsu
plt.axis('off')

# ---------------------------------------
# 7. Mostrar todas las figuras
# ---------------------------------------
plt.tight_layout()
plt.show()
```

La imagen sobre la que se probará es la siguiente:



Como podemos observar, esta es la fotografía de la página de un libro con una baja iluminación y bajo contraste, por lo que la mayoría de sus valores se encuentran cercanos a 0 (negro).

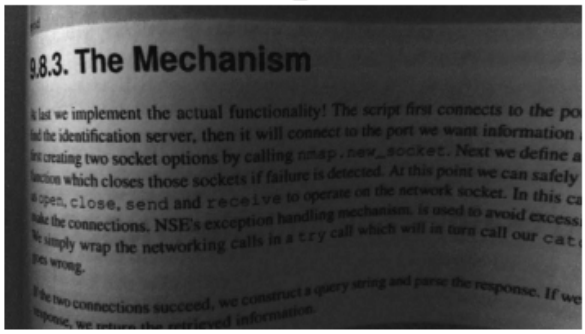
Para la primera parte probaremos distintos valores de umbral:

Con umbral = 127

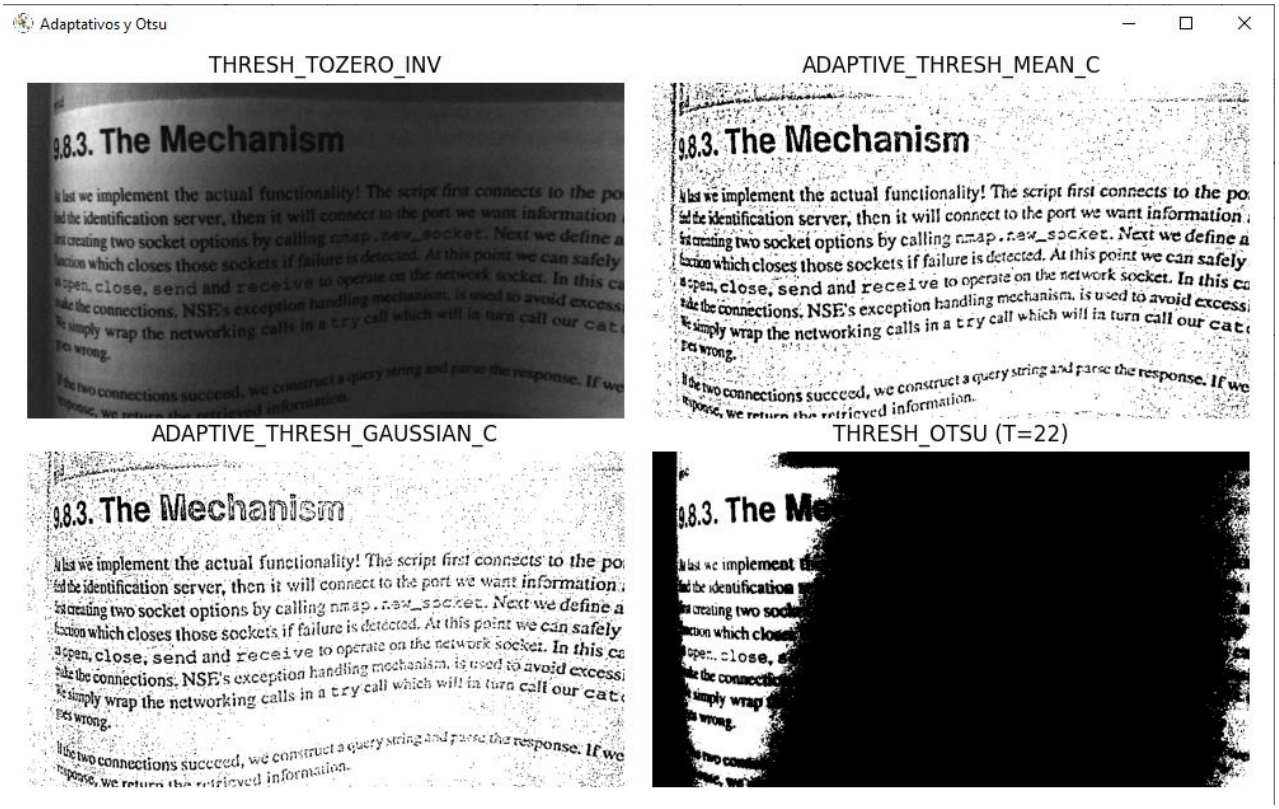
THRESH_BINARY

THRESH_BINARY_INV

THRESH_TRUNC



THRESH_TOZERO



Como podemos ver, los umbrales binarios hacen que la imagen se vuelva completamente negra. Esto es debido a que los valores menores a 127 se van a 0 y los mayores se van a 255, pero al ser una imagen muy oscura, todos los valores de la imagen son muy chicos y menores a 127, por lo que se van todos a 0. Lo mismo ocurre con el tozero.

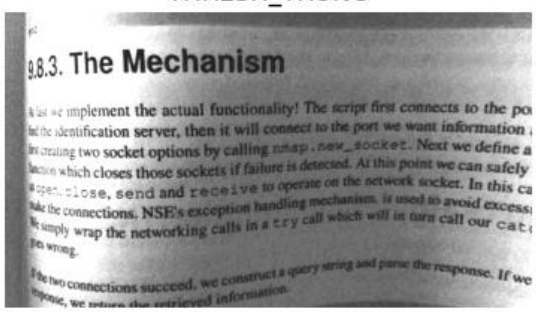
Mean, Gaussian otsu no dependen de este valor de umbral, por lo que los retomaremos posteriormente.

Con umbral = 25

THRESH_BINARY

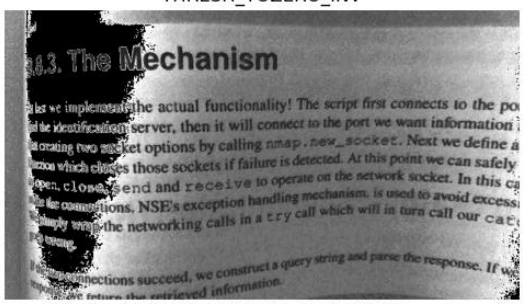
THRESH_BINARY_INV

THRESH_TRUNC

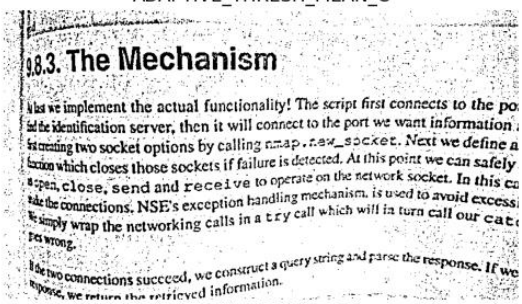


THRESH_TOZERO

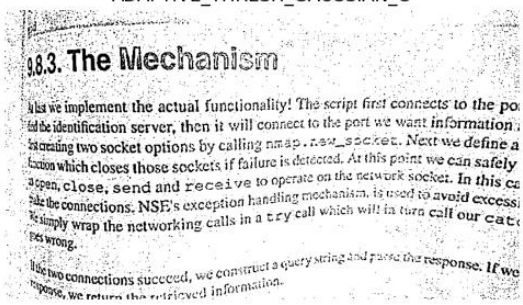
THRESH_TOZERO_INV



ADAPTIVE_THRESH_MEAN_C



ADAPTIVE_THRESH_GAUSSIAN_C

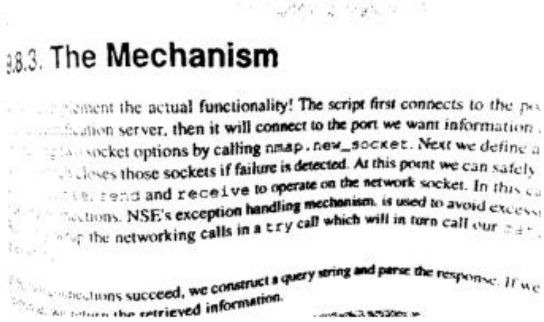


THRESH_OTSU (T=22)

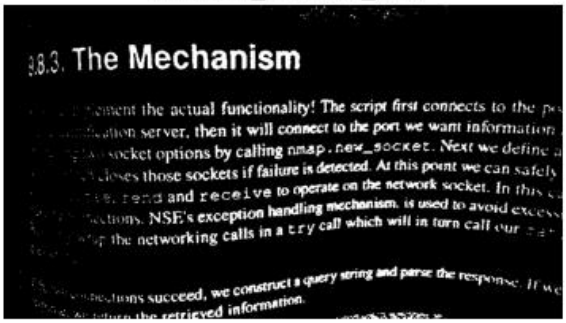
Reduciendo el valor del umbral vemos que en los umbrales binarios se comienzan a apreciar partes de la página, pero solo aquellas con mas iluminación, por lo que podemos intuir que estamos cerca del valor de los pixeles de las letras pero aun no llegamos, por lo que reducimos el umbral un poco más.

Con umbral = 7

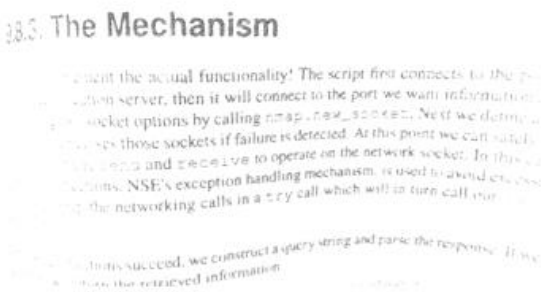
THRESH_BINARY



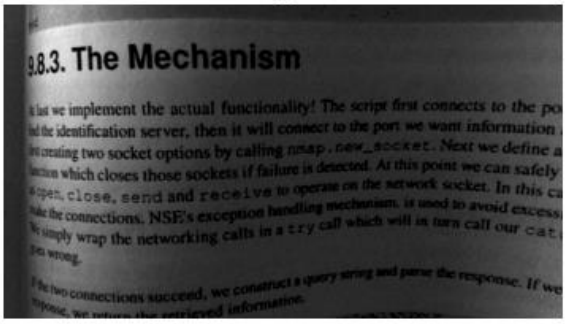
THRESH_BINARY_INV



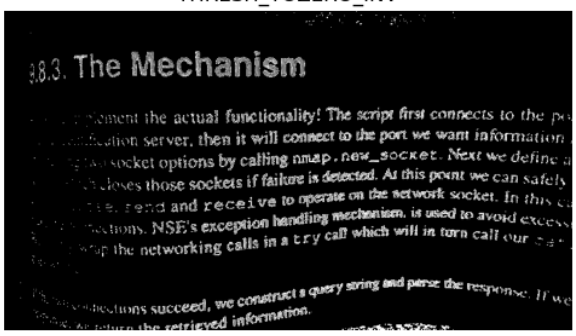
THRESH_TRUNC



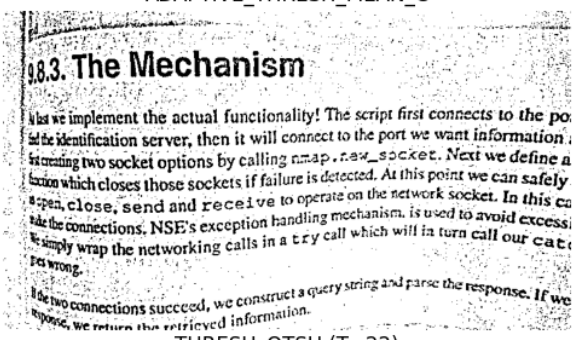
THRESH_TOZERO



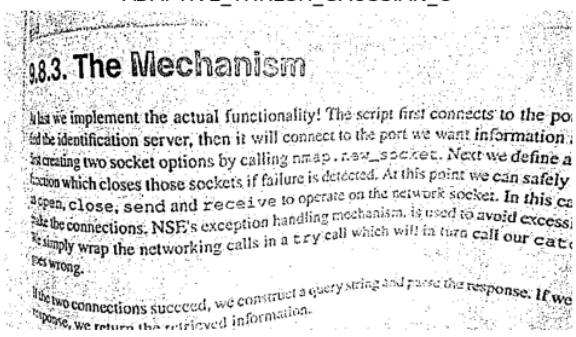
THRESH_TOZERO_INV



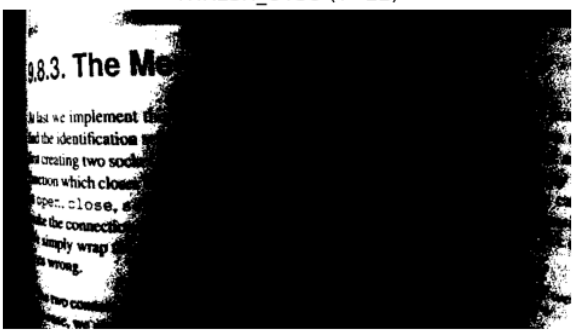
ADAPTIVE_THRESH_MEAN_C



ADAPTIVE_THRESH_GAUSSIAN_C



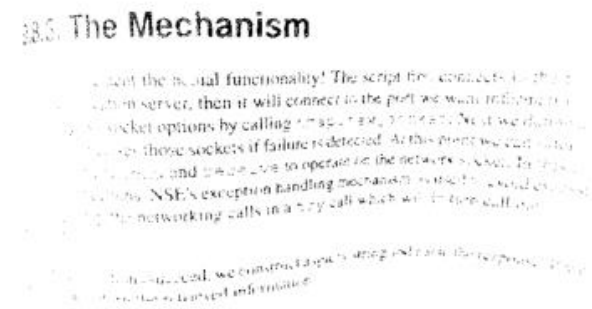
THRESH_OTSU (T=22)



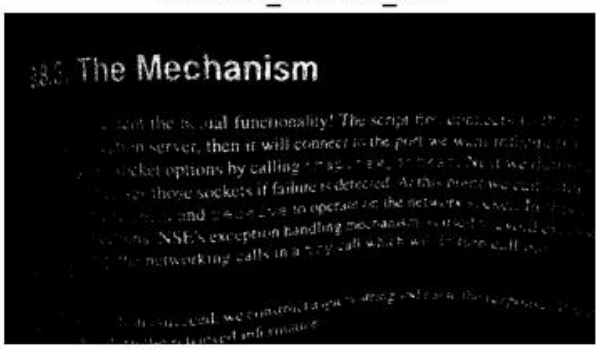
Reduciendo el valor de umbral a 7 podemos ver que ya se distinguen mejor las letras, ya que ese es el valor promedio que tienen los pixeles que corresponden a las letras, y los valores que corresponden a la hoja son mayores a 7, por lo que eso se mandan a 255.

Con umbral = 4

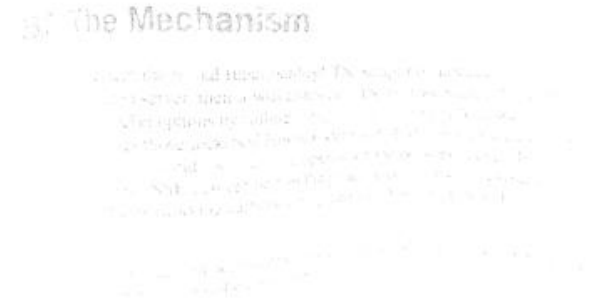
THRESH_BINARY



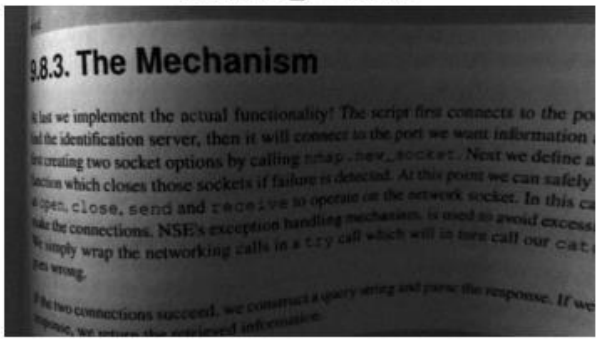
THRESH_BINARY_INV



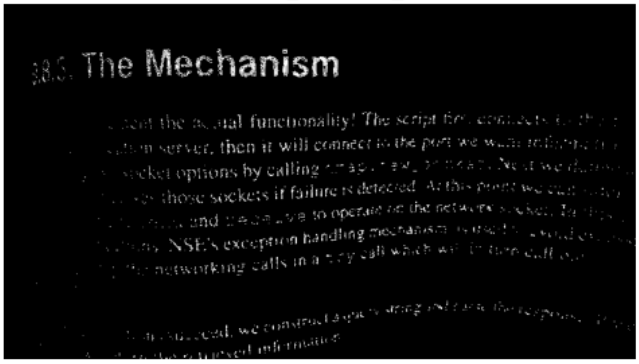
THRESH_TRUNC



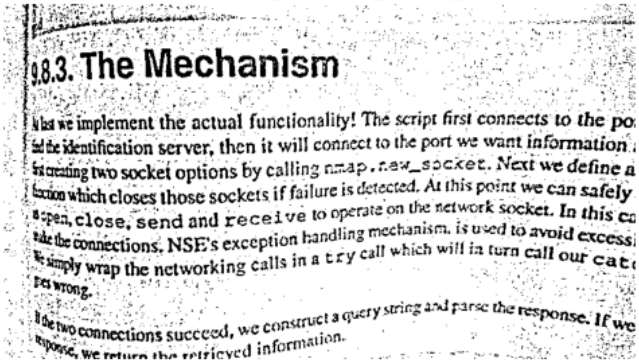
THRESH_TOZERO



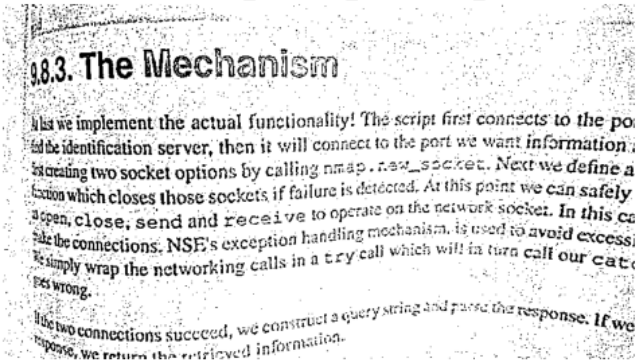
THRESH_TOZERO_INV



ADAPTIVE_THRESH_MEAN_C



ADAPTIVE_THRESH_GAUSSIAN_C



THRESH_OTSU (T=22)

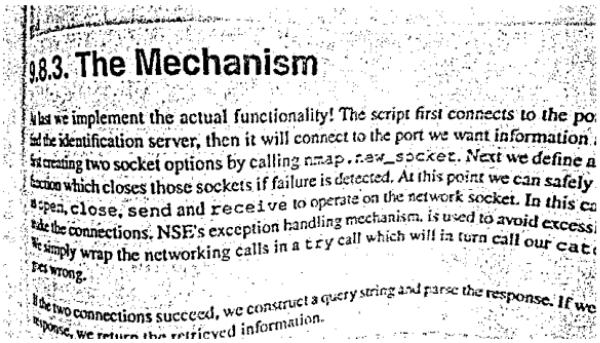
Si seguimos reduciendo el valor vemos que ya no se recupera del todo la información, ya que los valores de algunos pixeles de las letras se encuentran entre 4 y 7 y al reducir el umbral estas letras se pierden. En el caso de esta imagen, después de varias pruebas, se encontró que el valor promedio mas apto es el 7, aunque con este valor no se logra recuperar la información de algunas partes más claras porque al ser umbrales fijos, se aplica esta condición en toda la imagen sin tener en cuenta que hay zonas más oscuras y más claras, por lo que en las partes más iluminadas no logra recuperarse la información claramente.

Es aquí donde entran aquellos umbrales que no son fijos: mean, Gaussian y otsu, que corresponden a las ultimas 3 imágenes y que no se vieron alterados con el valor del umbral de la primera parte. Viendo estos 3, el que mejor resultado dio fueron el mean y el Gaussian, por lo que variaremos sus parámetros para intentar obtener una mejor imagen.

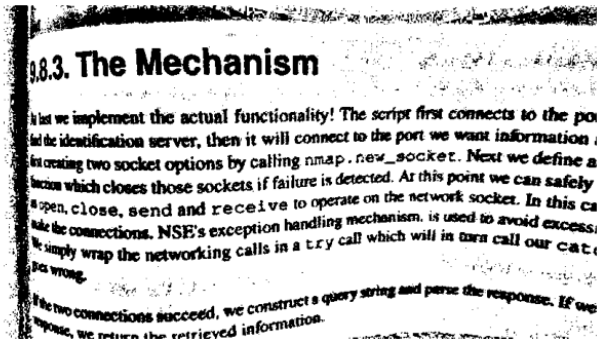
```
import numpy as np
import matplotlib.pyplot as plt
import cv2

# Cargar la imagen (asegúrate de tener una imagen de una página de un libro)
img = cv2.imread('bookpage.jpg')  # Reemplaza con el path de tu imagen
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # Convertir a escala de grises

# Definir el valor máximo para los píxeles umbralizados
max_val = 255

# Variar los parámetros para observar los efectos

# Primer caso:
thresh1 = cv2.adaptiveThreshold(
    gray, max_val, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 11, 2)

# Segundo caso:
thresh2 = cv2.adaptiveThreshold(
    gray, max_val, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 55, 2)

# Tercer caso:
thresh3 = cv2.adaptiveThreshold(
    gray, max_val, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)

# Cuarto caso:
thresh4 = cv2.adaptiveThreshold(
    gray, max_val, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 55, 2)

# Mostrar los resultados en distintas ventanas para comparar los efectos
plt.figure(figsize=(12, 12))

# Primer umbral adaptativo
plt.subplot(2, 2, 1)
```

```
plt.imshow(thresh1, cmap='gray')
plt.title("Caso 1: BLOQUE=11, C=2")
plt.axis('off')

# Segundo umbral adaptativo
plt.subplot(2, 2, 2)
plt.imshow(thresh2, cmap='gray')
plt.title("Caso 2: BLOQUE=55, C=2")
plt.axis('off')

# Tercer umbral adaptativo (gaussiano)
plt.subplot(2, 2, 3)
plt.imshow(thresh3, cmap='gray')
plt.title("Caso 3: GAUSSIANO, BLOQUE=11, C=2")
plt.axis('off')

# Cuarto umbral adaptativo (gaussiano)
plt.subplot(2, 2, 4)
plt.imshow(thresh4, cmap='gray')
plt.title("Caso 4: GAUSSIANO, BLOQUE=55, C=2")
plt.axis('off')

# Mostrar todas las imágenes
plt.tight_layout()
plt.show()
```

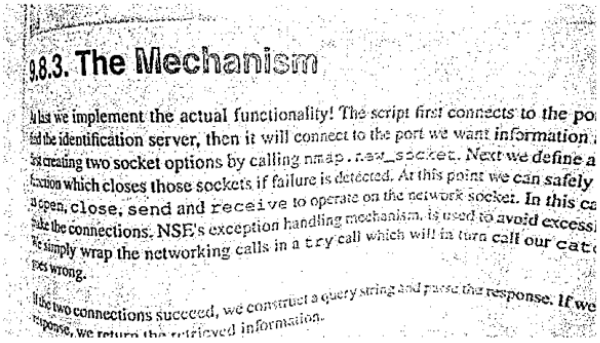
Caso 1: BLOQUE=11, C=2



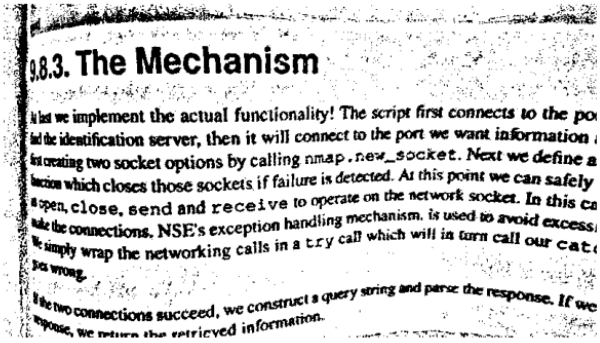
Caso 2: BLOQUE=55, C=2



Caso 3: GAUSSIANO, BLOQUE=11, C=2



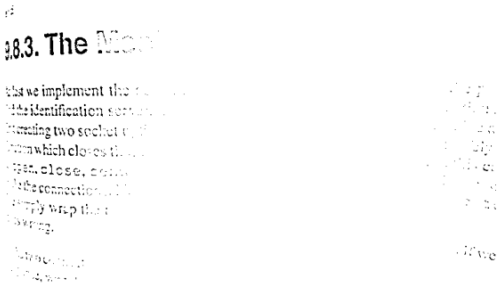
Caso 4: GAUSSIANO, BLOQUE=55, C=2



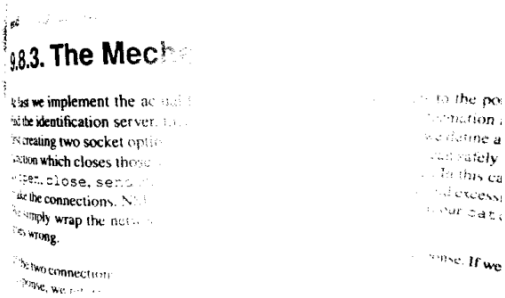
Podemos ver que en ambos casos, al aumentar el área de los pixeles que se ponderaran, se mejora la umbralización. En ambos casos ya no es un valor de umbral para toda la imagen, sino que este valor depende de los pixeles que se encuentren alrededor del pixel sobre el que se esta trabajando, por lo que no se manejara el mismo umbral para un pixel que se encuentra en un área mas iluminada que en un pixel que está en un área más oscura.

Tambien variamos el valor C, y si lo aumentamos mucho, vemos que se pierden las letras en zonas oscuras:

Caso 1: BLOQUE=11, C=10

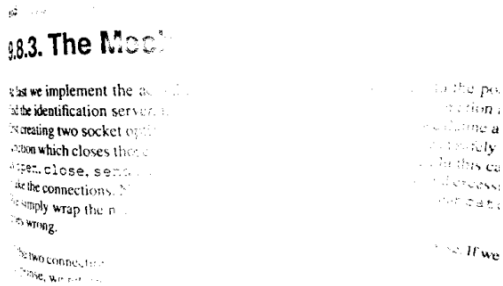


Caso 2: BLOQUE=55, C=10



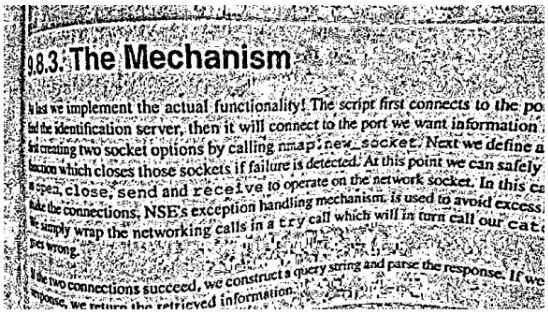
Caso 3: GAUSSIANO, BLOQUE=11, C=10

Caso 4: GAUSSIANO, BLOQUE=55, C=10

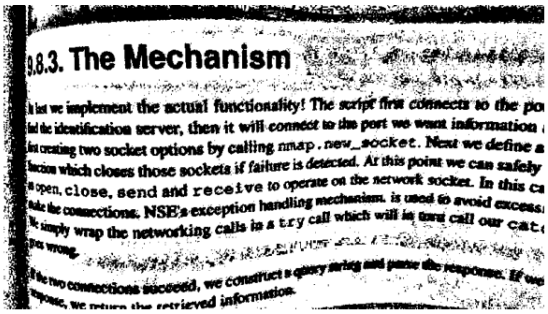


Y si C es muy chico no se logra aclarar la página completamente:

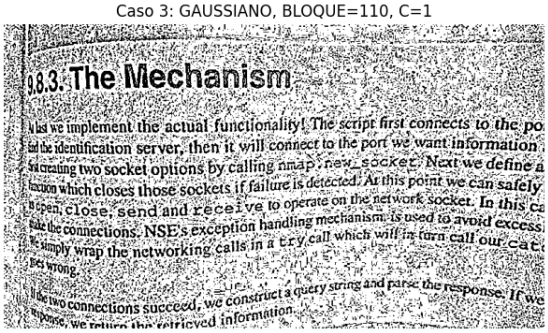
Caso 1: BLOQUE=110, C=1



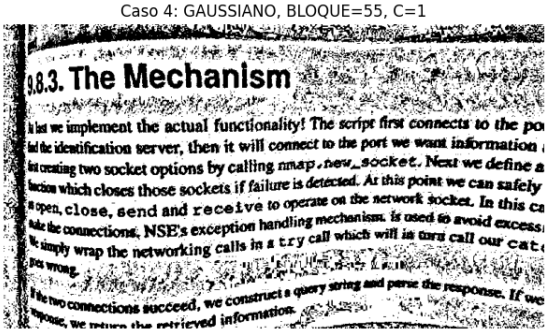
Caso 2: BLOQUE=55, C=1



Caso 3: GAUSSIANO, BLOQUE=110, C=1



Caso 4: GAUSSIANO, BLOQUE=55, C=1



Fuentes:

https://www.geeksforgeeks.org/python-thresholding-techniques-using-opencv-set-1-simple-thresholding/

https://pythonprogramming.net/thresholding-image-analysis-python-opencv-tutorial/

https://akridata.ai/blog/image-segmentation-thresholding-computer-vision/