

Laboratorio 2 - Visión por computadora

- Nelson García
- Joaquín Puente
- Diego Linares

Task 1 - Notch Filter para eliminar ruido periódico

Eliminación de ruido sinusoidal (patrón de rayas diagonales) en imágenes satelitales usando filtrado en el dominio de frecuencia.

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

img = cv2.imread("imgs/periodic_noise.jpg", cv2.IMREAD_GRAYSCALE)
print(f"Imagen cargada: {img.shape}")
plt.figure(figsize=(8, 6))
plt.imshow(img, cmap='gray')
plt.title("Imagen Original con Ruido Periódico")
plt.axis('off')
plt.show()

Imagen cargada: (512, 512)
```

Imagen Original con Ruido Periódico



```
F = np.fft.fft2(img.astype(np.float32))
Fshift = np.fft.fftshift(F)

magnitude = np.log1p(np.abs(Fshift))

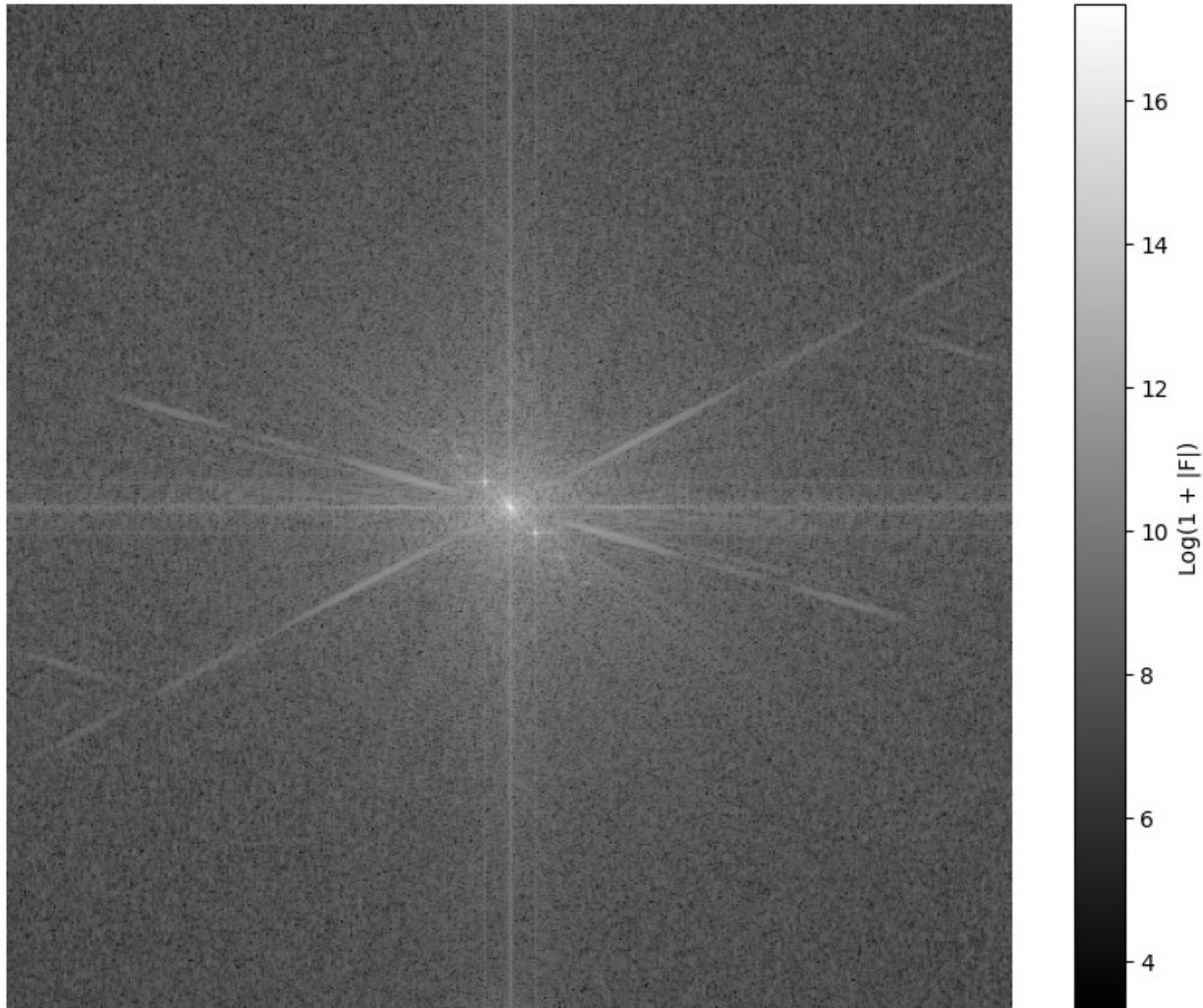
h, w = img.shape
cy, cx = h // 2, w // 2

yy, xx = np.mgrid[0:h, 0:w]
dist_from_center = np.sqrt((yy - cy)**2 + (xx - cx)**2)
angles_map = np.arctan2(yy - cy, xx - cx)

plt.figure(figsize=(10, 8))
plt.imshow(magnitude, cmap='gray')
plt.title("Espectro de Magnitud (escala logarítmica)")
plt.colorbar(label='Log(1 + |F|)')
plt.axis('off')
plt.show()
```

```
print("Las líneas brillantes que salen del centro representan las  
frecuencias del ruido periódico")
```

Espectro de Magnitud (escala logarítmica)



Las líneas brillantes que salen del centro representan las frecuencias del ruido periódico

```
from scipy.signal import find_peaks  
  
DC_RADIUS = 3  
  
n_angles = 360  
angles_test = np.linspace(0, np.pi, n_angles, endpoint=False)  
angular_profile = []  
  
for angle in angles_test:  
    angle_diff1 = np.abs(np.mod(angles_map - angle + np.pi, 2*np.pi) -
```

```

np.pi)
    angle_diff2 = np.abs(np.mod(angles_map - (angle + np.pi) + np.pi,
2*np.pi) - np.pi)
    angle_mask = (np.minimum(angle_diff1, angle_diff2) <
np.radians(1)) & (dist_from_center > DC_RADIUS)

    if np.any(angle_mask):
        angular_profile.append(np.sum(magnitude[angle_mask]))
    else:
        angular_profile.append(0)

angular_profile = np.array(angular_profile)

peaks_idx, _ = find_peaks(angular_profile,
height=np.percentile(angular_profile, 80), distance=15)
dominant_angles_rad = angles_test[peaks_idx]
dominant_angles = np.degrees(dominant_angles_rad)

print(f"Ángulos de ruido detectados automáticamente: {[round(a) for a
in dominant_angles]}°")

plt.figure(figsize=(12, 10))
plt.imshow(magnitude, cmap='gray')
plt.plot(cx, cy, 'g+', markersize=25, markeredgewidth=3, label='DC
(centro)')

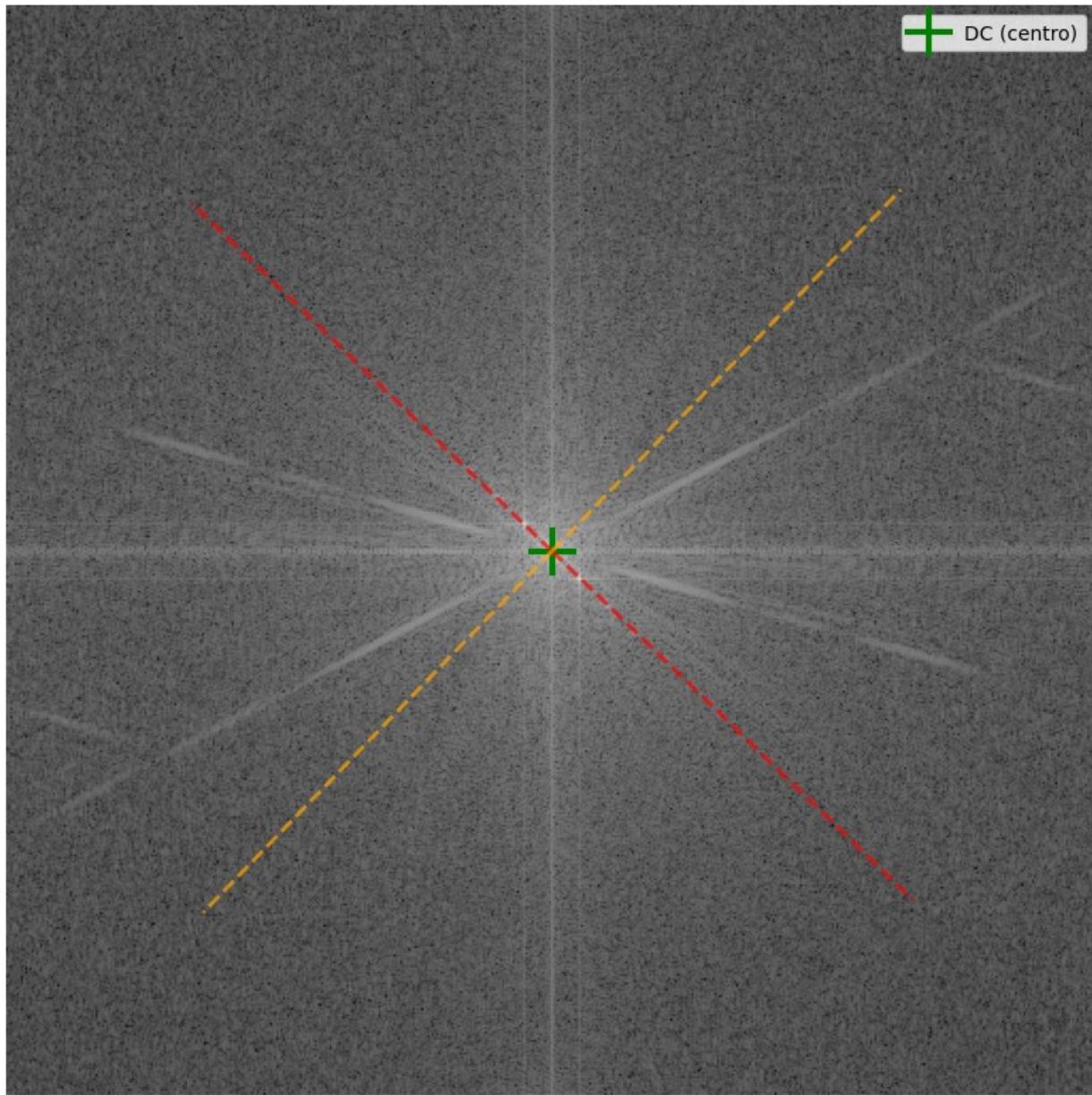
length = min(h, w) // 2 - 20
colors = ['r', 'orange', 'yellow', 'cyan']
for i, angle_deg in enumerate(dominant_angles):
    for offset in [0, 180]:
        angle_rad = np.radians(angle_deg + offset)
        x_end = cx + length * np.cos(angle_rad)
        y_end = cy + length * np.sin(angle_rad)
        plt.plot([cx, x_end], [cy, y_end], '--', color=colors[i %
len(colors)], linewidth=2, alpha=0.7)

plt.title(f"Espectro con líneas de ruido detectadas: {[round(a) for a
in dominant_angles]}°")
plt.legend(loc='upper right')
plt.axis('off')
plt.show()

Ángulos de ruido detectados automáticamente: [44, 134]°

```

Espectro con líneas de ruido detectadas: [44, 134]°



```
notch_mask = np.ones((h, w), dtype=np.float32)
BAND_WIDTH = np.radians(10)

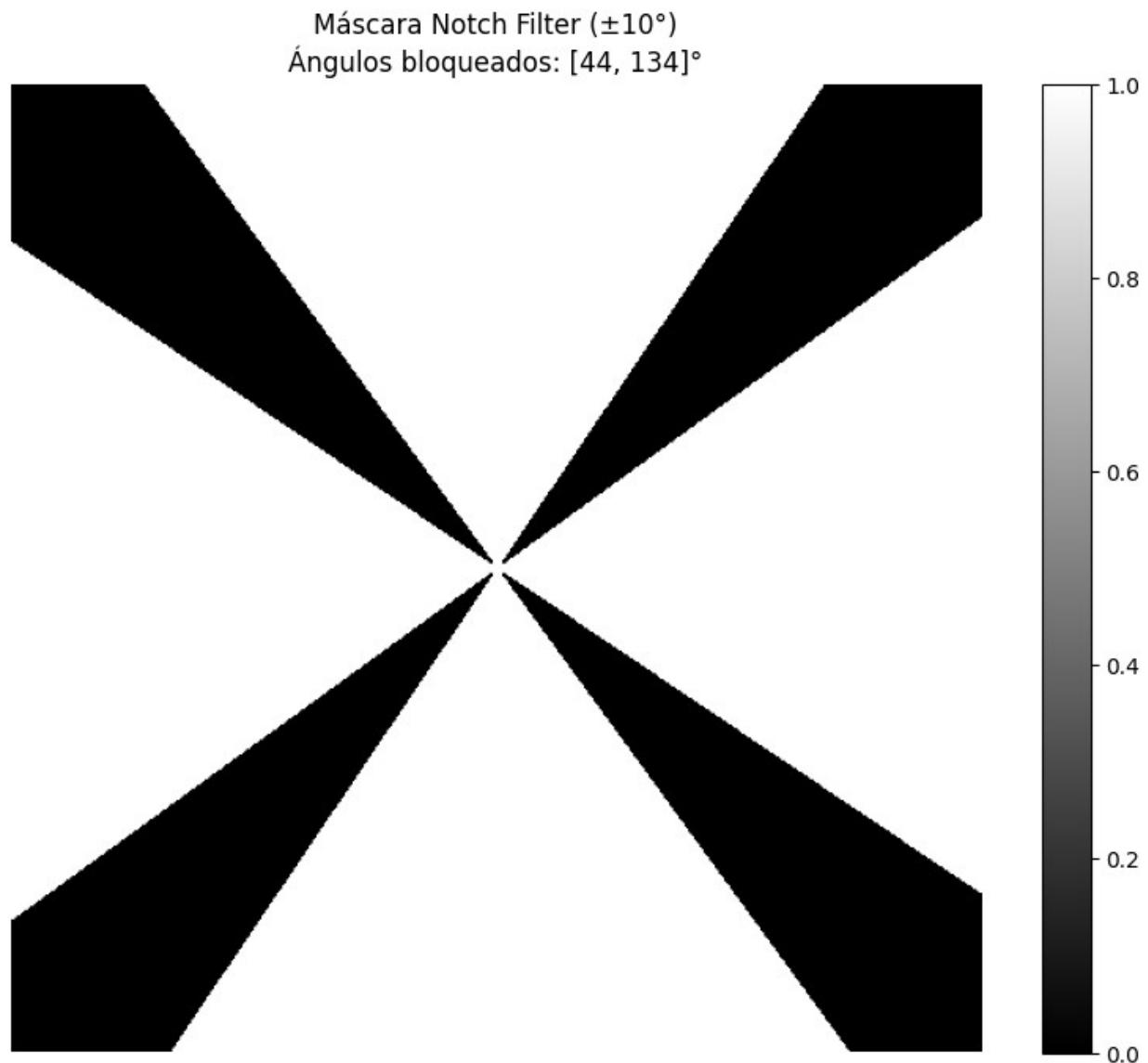
for angle_rad in dominant_angles_rad:
    for angle_offset in [0, np.pi]:
        target_angle = angle_rad + angle_offset
        angle_diff = np.abs(np.mod(angles_map - target_angle + np.pi,
        2*np.pi) - np.pi)
        band = (angle_diff < BAND_WIDTH) & (dist_from_center >
DC_RADIUS)
        notch_mask[band] = 0
```

```

plt.figure(figsize=(10, 8))
plt.imshow(notch_mask, cmap='gray')
plt.title(f"Máscara Notch Filter ( $\pm$ {int(np.degrees(BAND_WIDTH))}°)\nÁngulos bloqueados: {[round(a) for a in dominant_angles]}°")
plt.colorbar()
plt.axis('off')
plt.show()

print(f"Ángulos bloqueados: {[round(a) for a in dominant_angles]}°")
print(f"Ancho de banda:  $\pm$ {int(np.degrees(BAND_WIDTH))}°")
print(f"Radio DC preservado: {DC_RADIUS} pixels")

```



```

Ángulos bloqueados: [44, 134]°
Ancho de banda: ±10°
Radio DC preservado: 3 pixels

Fshift_filtered = Fshift * notch_mask

F_ishift = np.fft.ifftshift(Fshift_filtered)
img_restored = np.fft.ifft2(F_ishift)
img_restored = np.real(img_restored)

img_restored = np.clip(img_restored, 0, 255).astype(np.uint8)

magnitude_filtered = np.log1p(np.abs(Fshift_filtered))

plt.figure(figsize=(14, 5))

plt.subplot(1, 3, 1)
plt.imshow(magnitude, cmap='gray')
plt.title("Espectro Original")
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(notch_mask, cmap='gray')
plt.title("Máscara Notch")
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(magnitude_filtered, cmap='gray')
plt.title("Espectro Filtrado\n(sin las frecuencias de ruido)")
plt.axis('off')

plt.tight_layout()
plt.show()

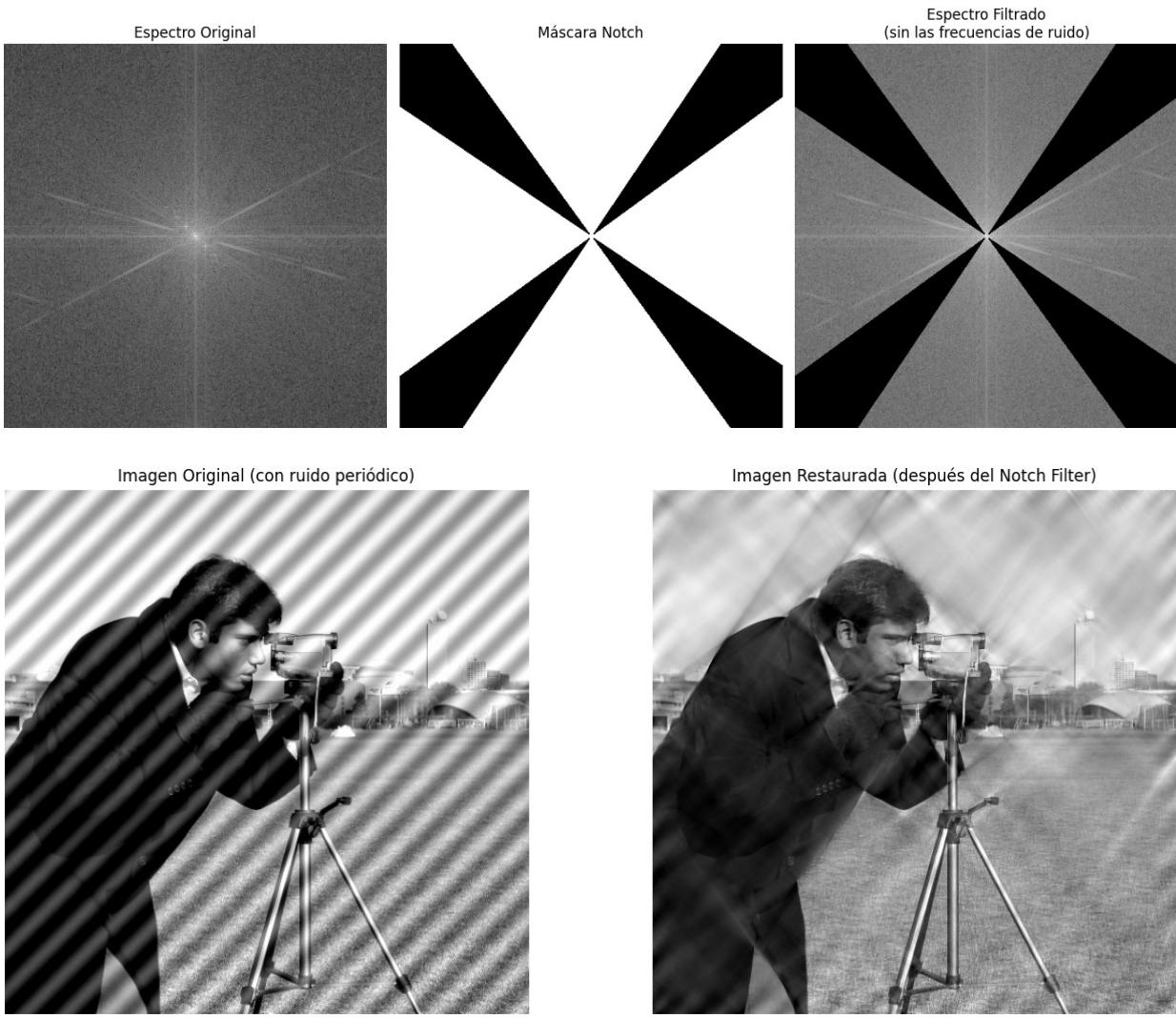
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.imshow(img, cmap='gray')
plt.title("Imagen Original (con ruido periódico)")
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(img_restored, cmap='gray')
plt.title("Imagen Restaurada (después del Notch Filter)")
plt.axis('off')

plt.tight_layout()
plt.show()

```



La imagen restaurada muestra los detalles sin las rayas diagonales del ruido

¿Por qué un filtro de promedio 5x5 sería una mala solución?

Un filtro de promedio (average filter) de 5x5 en el dominio espacial sería una mala solución para este problema específico porque el filtro de promedio actúa como un filtro pasa-bajos que suaviza toda la imagen, eliminando tanto el ruido como los detalles geográficos importantes (bordes, texturas naturales, estructuras pequeñas). Además, no es selectivo en frecuencia, el ruido periódico tiene frecuencias muy específicas (los picos que identificamos en el espectro). Un filtro espacial 5x5 no puede distinguir entre estas frecuencias parásitas y las frecuencias similares que pertenecen a la información real de la imagen.

```
kernel_avg = np.ones((5, 5), np.float32) / 25
img_avg_filtered = cv2.filter2D(img, -1, kernel_avg)

plt.figure(figsize=(16, 5))
```

```

plt.subplot(1, 3, 1)
plt.imshow(img, cmap='gray')
plt.title("Original (con ruido)")
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(img_avg_filtered, cmap='gray')
plt.title("Filtro Promedio 5x5\n(borroso, ruido aún visible)")
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(img_restored, cmap='gray')
plt.title("Notch Filter\n(nítido, sin ruido)")
plt.axis('off')

plt.tight_layout()
plt.show()

```



TASK 2

Está desarrollando un sistema biométrico de seguridad. El sensor de huellas dactilares está sucio y produce imágenes binarias con dos tipos de defectos:

1. Pequeños puntos blancos en los valles negros de la huella (Ruido Sal)
2. Las “crestas” de la huella tienen pequeñas roturas que impiden que el algoritmo de matching funcione (grietas).

```

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

img = cv2.imread("imgs/fingerprint_noisy.png", cv2.IMREAD_GRAYSCALE)

# Se transforma la imagen a formato binario
_, img_bin = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)

```

```

kernel_open = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3,3))
img_open = cv2.morphologyEx(img_bin, cv2.MORPH_OPEN, kernel_open)

kernel_close = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5,5))
img_final = cv2.morphologyEx(img_open, cv2.MORPH_CLOSE, kernel_close)

# Visualización
plt.figure(figsize=(12,4))

plt.subplot(1,3,1)
plt.imshow(img_bin, cmap='gray')
plt.title("Original")
plt.axis("off")

plt.subplot(1,3,2)
plt.imshow(img_open, cmap='gray')
plt.title("Después de Apertura")
plt.axis("off")

plt.subplot(1,3,3)
plt.imshow(img_final, cmap='gray')
plt.title("Resultado Final")
plt.axis("off")

plt.show()

```



¿El orden de los factores altera el producto?

Si se aplica Cierre luego Apertura: el cierre engorda el ruido blanco luego la apertura no logra eliminarlo lo que nos daria como resultado crestas más gruesas, ruido todavía presente

Demostración visual (orden incorrecto)

```

# Orden inverso (incorrecto)
img_close_first = cv2.morphologyEx(img_bin, cv2.MORPH_CLOSE,
kernel_close)

```

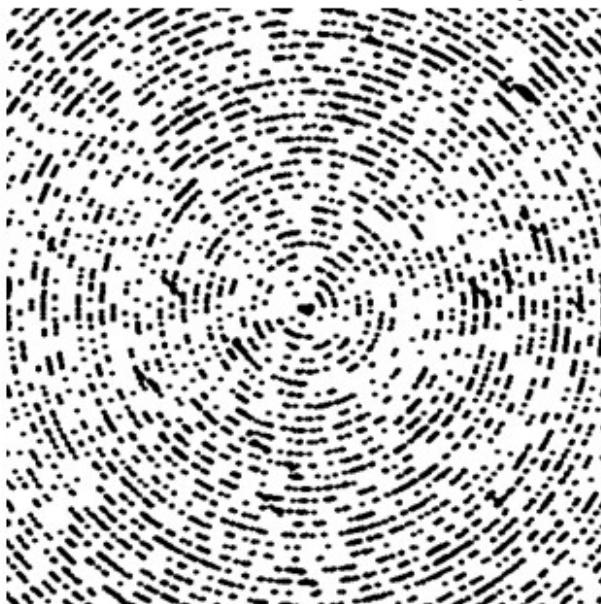
```

img_wrong = cv2.morphologyEx(img_close_first, cv2.MORPH_OPEN,
kernel_open)

plt.figure(figsize=(6,4))
plt.imshow(img_wrong, cmap='gray')
plt.title("Orden Incorrecto: Cierre LUEGO Apertura")
plt.axis("off")
plt.show()

```

Orden Incorrecto: Cierre LUEGO Apertura



El orden importa porque cada operación modifica el dominio sobre el que actúa la siguiente. Primero hay que eliminar ruido pequeño, luego reparar estructuras grandes. Hacerlo al revés amplifica el problema.

Task 3

Una fábrica textil necesita detectar rasgaduras en telas de mezclilla (denim) automáticamente. El problema es que la tela tiene una textura natural fuerte (patrón repetitivo) que confunde a los detectores de bordes simples (Canny), detectando el tejido como si fuera un defecto. Por ello se le pide que usted diseñe un pipeline híbrido que combine Fourier y Morfología para aislar solamente la rasgadura. Para ello comienza por probar su solución en una imagen que tiene a mano. Con esto en mente, realice:

```

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

```

1. Utilice Fourier para analizar la textura repetitiva de la tela. Diseñe un filtro que elimine las frecuencias altas/repetitivas del tejido, dejando una imagen "suavizada" donde solo resalte la anomalía (la rasgadura) y la iluminación global. (Supresión de Textura)

a. Hint: ¿Qué pasa si eliminamos las frecuencias altas periféricas o específicas?

```
IMG_NAME = "imgs/textile_defect.jpg"

LOWPASS_SIGMA = 35
NOTCH_PEAKS = 8
NOTCH_RADIUS = 10

img = cv2.imread(IMG_NAME, cv2.IMREAD_GRAYSCALE)
if img is None:
    raise FileNotFoundError(f"No pude leer {IMG_NAME}. ¿Está en el mismo folder?")

h, w = img.shape
cy, cx = h // 2, w // 2

# FFT (pasar a frecuencia)
F = np.fft.fft2(img.astype(np.float32))
Fshift = np.fft.fftshift(F)

# Magnitud
mag = np.log1p(np.abs(Fshift))

# Filtro 1: Low-pass gaussiano (quita altas frecuencias periféricas)
yy, xx = np.mgrid[0:h, 0:w]
dist2 = (yy - cy) ** 2 + (xx - cx) ** 2
lowpass = np.exp(-dist2 / (2.0 * (LOWPASS_SIGMA ** 2))).astype(np.float32)

# Filtro 2: Notch (apaga picos fuertes repetitivos del tejido)
mag2 = mag.copy()

# Quitar la zona central
center_block = 25
mag2[cy - center_block:cy + center_block, cx - center_block:cx + center_block] = 0

# Elegir los picos más fuertes
flat = mag2.ravel()
idxs = np.argpartition(flat, -NOTCH_PEAKS)[-NOTCH_PEAKS:]
coords = np.column_stack(np.unravel_index(idxs, mag2.shape))

notch = np.ones((h, w), dtype=np.float32)
for y, x in coords:
    cv2.circle(notch, (int(x), int(y)), NOTCH_RADIUS, 0, -1)
    y2, x2 = int(2 * cy - y), int(2 * cx - x)
    if 0 <= y2 < h and 0 <= x2 < w:
        cv2.circle(notch, (x2, y2), NOTCH_RADIUS, 0, -1)
```

```

# --- Máscara final en frecuencia ---
freq_mask = lowpass * notch

# --- Aplicar filtro en frecuencia ---
F_filt = Fshift * freq_mask

# --- IFFT (volver al dominio espacial) ---
img_back = np.fft.ifft2(np.fft.ifftshift(F_filt))
img_back = np.real(img_back)

# Normalizar para visualizar bien
smooth = cv2.normalize(img_back, None, 0, 255,
cv2.NORM_MINMAX).astype(np.uint8)

plt.figure()
plt.title("Original (gris)")
plt.imshow(img, cmap="gray")
plt.axis("off")

plt.figure()
plt.title("FFT Magnitud (log)")
plt.imshow(mag, cmap="gray")
plt.axis("off")

plt.figure()
plt.title("Máscara Low-pass")
plt.imshow(lowpass, cmap="gray")
plt.axis("off")

plt.figure()
plt.title("Máscara Notch (picos apagados)")
plt.imshow(notch, cmap="gray")
plt.axis("off")

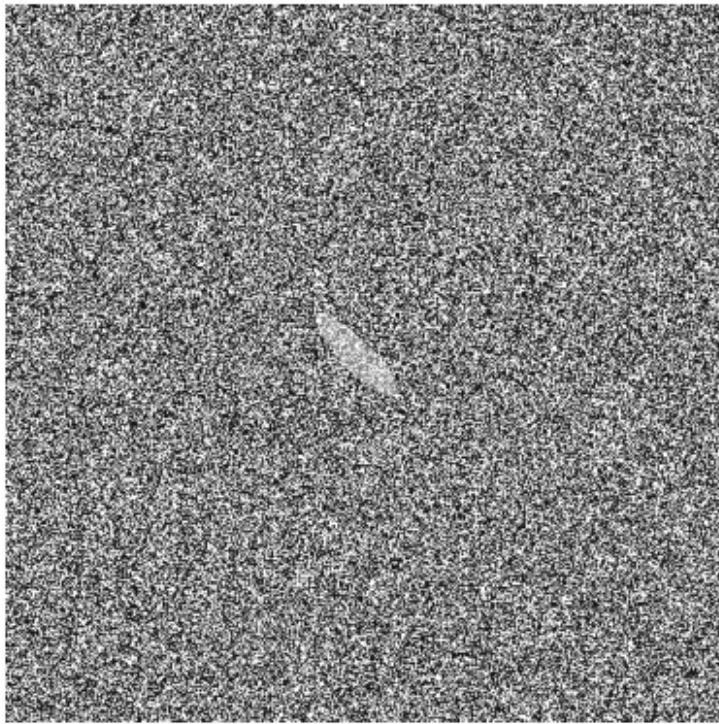
plt.figure()
plt.title("Máscara final (Low-pass * Notch)")
plt.imshow(freq_mask, cmap="gray")
plt.axis("off")

plt.figure()
plt.title("Resultado IFFT (textura suprimida)")
plt.imshow(smooth, cmap="gray")
plt.axis("off")

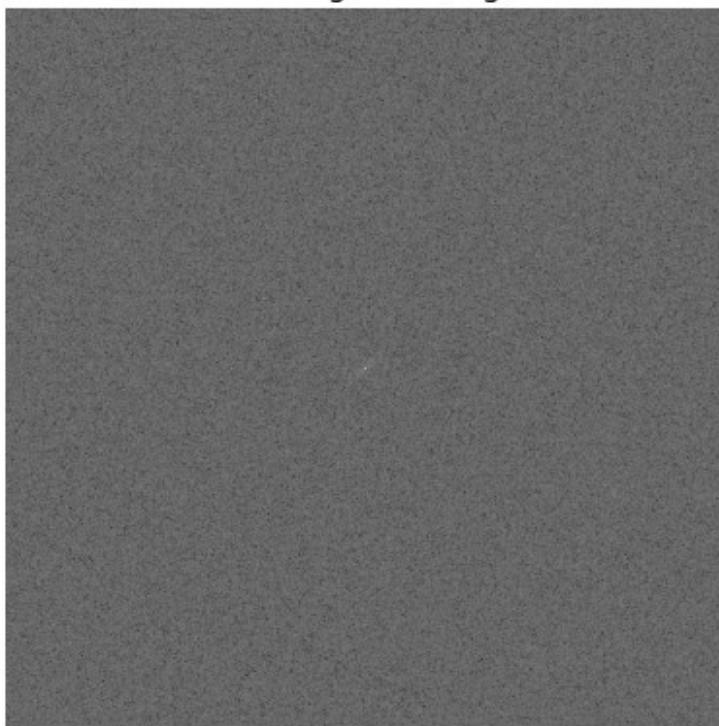
(np.float64(-0.5), np.float64(511.5), np.float64(511.5), np.float64(-
0.5))

```

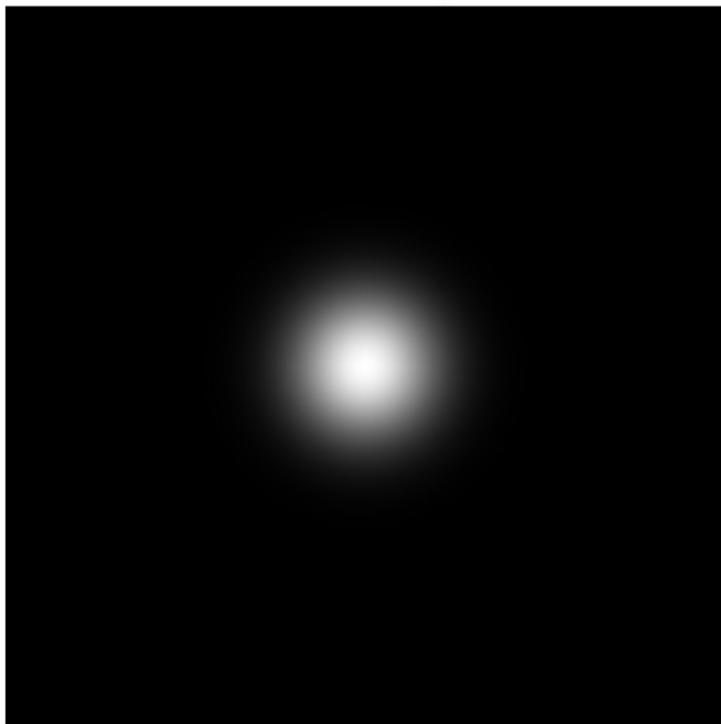
Original (gris)



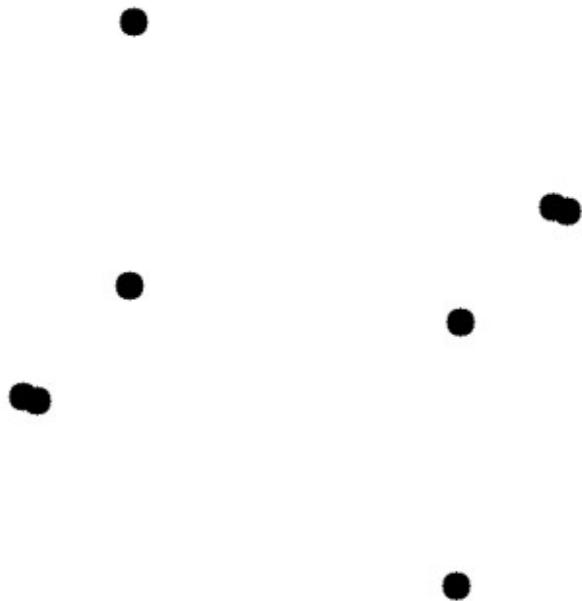
FFT Magnitud (log)



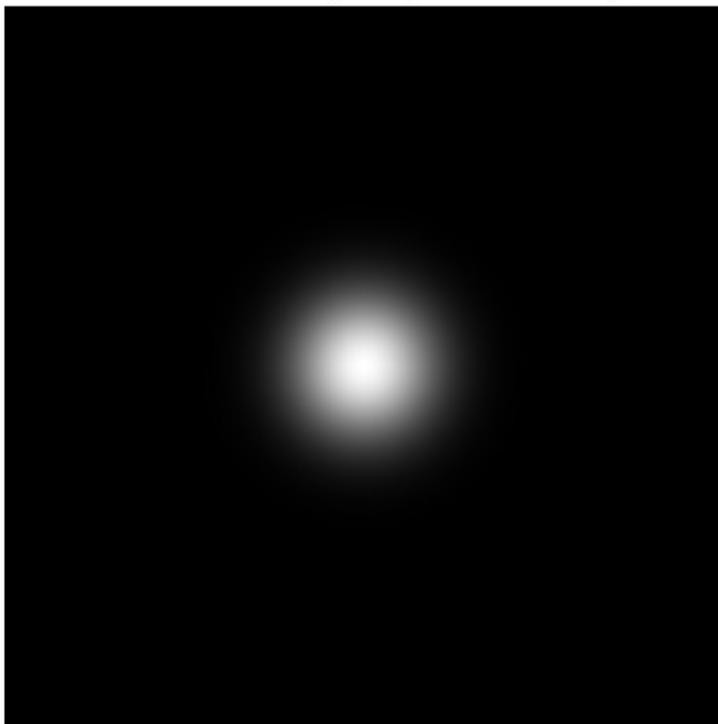
Máscara Low-pass



Máscara Notch (picos apagados)



Máscara final (Low-pass * Notch)



Resultado IFFT (textura suprimida)



1. Aplique un umbralizado (thresholding) a la imagen resultante del paso 1 para obtener una máscara binaria preliminar. (Segmentación)

```

# Thresholding

smooth.blur = cv2.GaussianBlur(smooth, (5, 5), 0)

_, th = cv2.threshold(smooth.blur, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)
_, th_inv = cv2.threshold(smooth.blur, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)

def white_ratio(mask):
    return np.mean(mask == 255)

r1 = white_ratio(th)
r2 = white_ratio(th_inv)

def score(r):
    if 0.01 <= r <= 0.40:
        return 1.0 - abs(r - 0.10)
    return -1.0

mask_pre = th if score(r1) >= score(r2) else th_inv

plt.figure()
plt.title("IFFT (textura suprimida)")
plt.imshow(smooth, cmap="gray")
plt.axis("off")

plt.figure()
plt.title("Blur (antes de threshold)")
plt.imshow(smooth.blur, cmap="gray")
plt.axis("off")

plt.figure()
plt.title("Otsu normal")
plt.imshow(th, cmap="gray")
plt.axis("off")

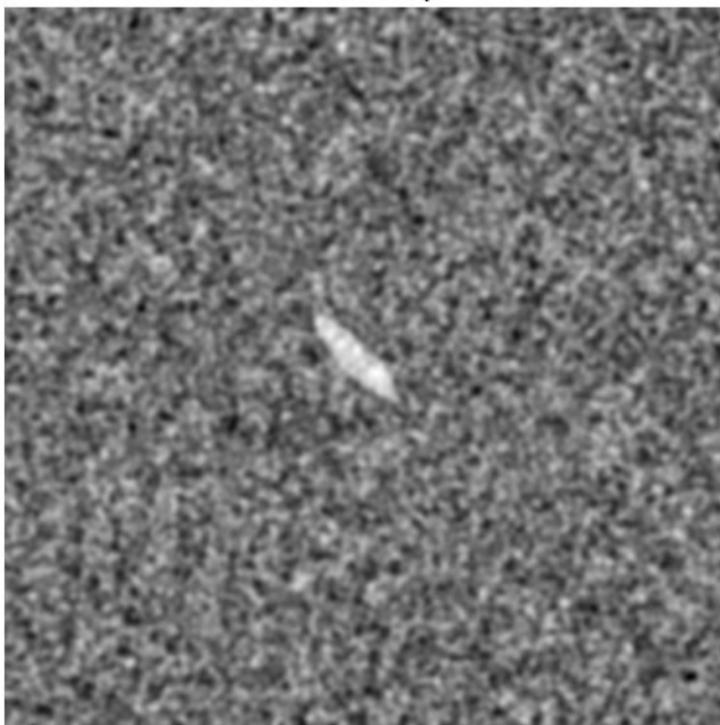
plt.figure()
plt.title("Otsu invertido")
plt.imshow(th_inv, cmap="gray")
plt.axis("off")

plt.figure()
plt.title("Máscara preliminar elegida (punto 2)")
plt.imshow(mask_pre, cmap="gray")
plt.axis("off")

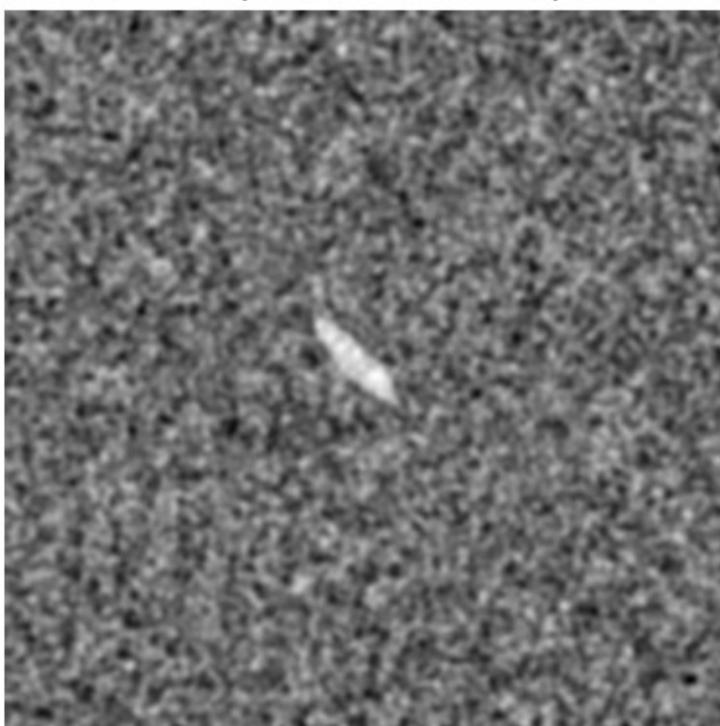
(np.float64(-0.5), np.float64(511.5), np.float64(511.5), np.float64(-
0.5))

```

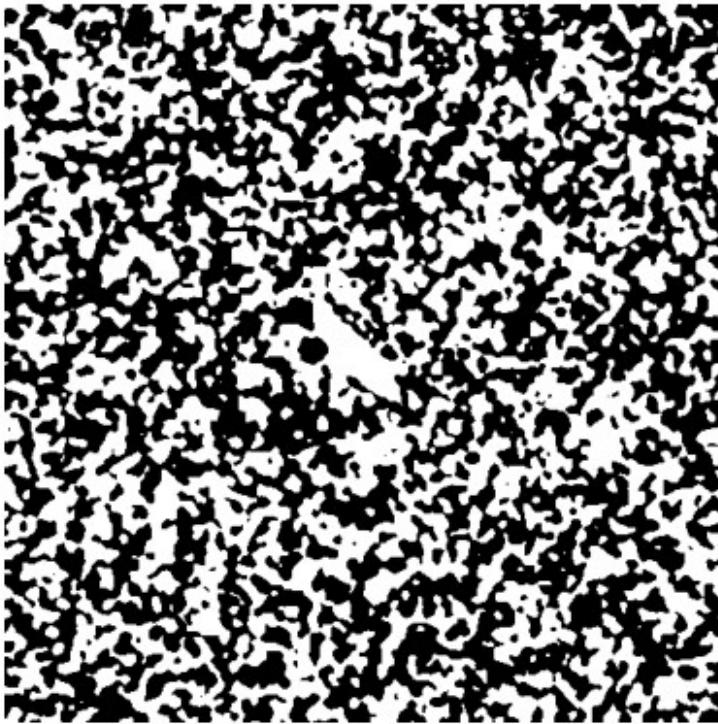
IFFT (textura suprimida)



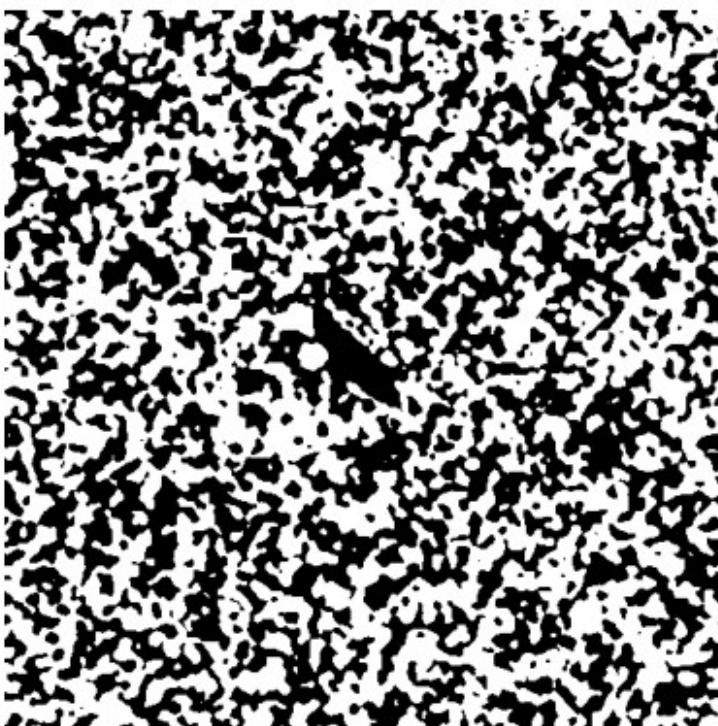
Blur (antes de threshold)



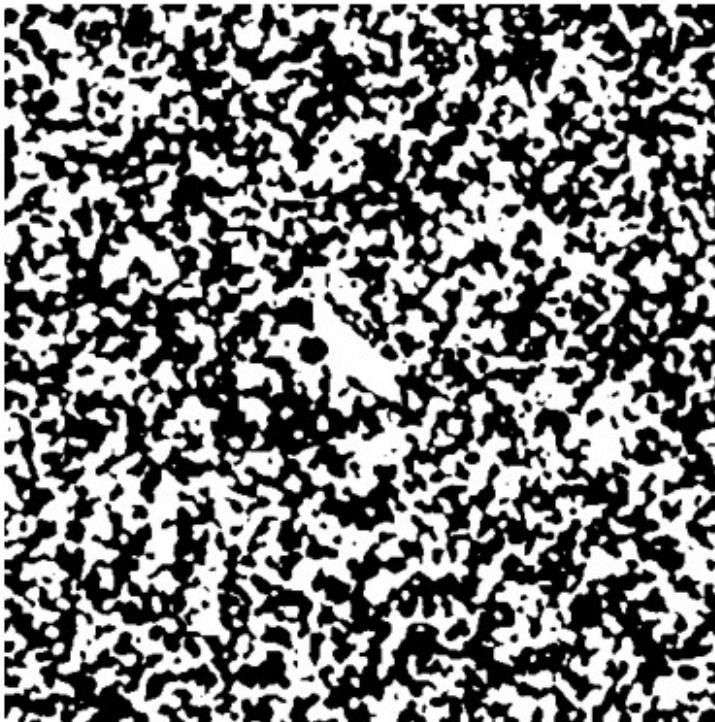
Otsu normal



Otsu invertido



Máscara preliminar elegida (punto 2)



1. La máscara seguramente tendrá ruido residual. Utilice operaciones morfológicas para limpiar la máscara y dejar únicamente la silueta de la rasgadura. (Refinamiento)

```
# Morfología:
```

```
# Quitar puntos/ruido pequeños
k_open = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (7, 7))
mask_open = cv2.morphologyEx(mask_pre, cv2.MORPH_OPEN, k_open,
iterations=3)

# Unir cortes y llenar huecos pequeños en la rasgadura
k_close = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (11, 11))
mask_close = cv2.morphologyEx(mask_open, cv2.MORPH_CLOSE, k_close,
iterations=3)

num, labels, stats, _ = cv2.connectedComponentsWithStats(mask_close,
connectivity=8)
mask_final = np.zeros_like(mask_close)

if num > 1:
    areas = stats[1:, cv2.CC_STAT_AREA]
    best = 1 + int(np.argmax(areas))
    mask_final[labels == best] = 255
else:
    mask_final = mask_close.copy()
```

```
plt.figure()
plt.title("Máscara preliminar (punto 2)")
plt.imshow(mask_pre, cmap="gray")
plt.axis("off")

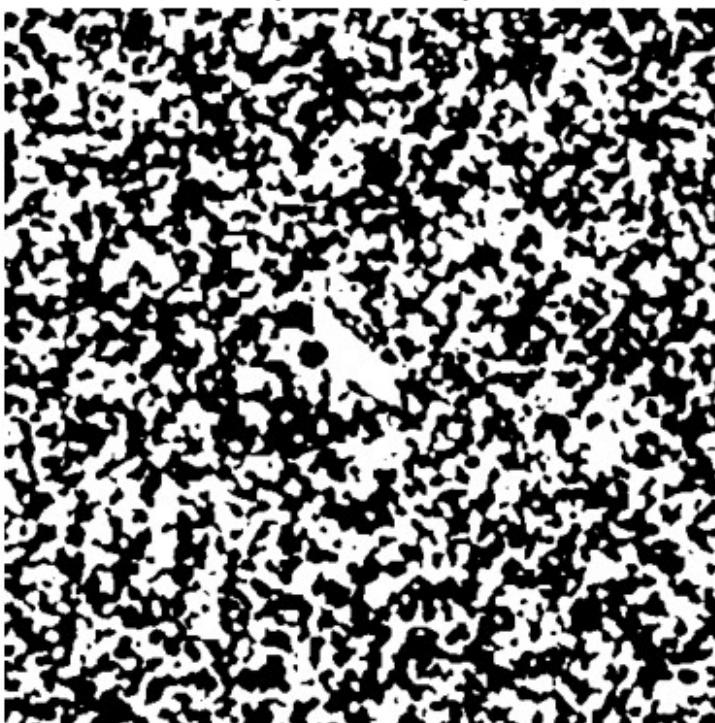
plt.figure()
plt.title("OPEN (quita ruido)")
plt.imshow(mask_open, cmap="gray")
plt.axis("off")

plt.figure()
plt.title("CLOSE (une / rellena)")
plt.imshow(mask_close, cmap="gray")
plt.axis("off")

plt.figure()
plt.title("Máscara final (rasgadura blanca)")
plt.imshow(mask_final, cmap="gray")
plt.axis("off")

(np.float64(-0.5), np.float64(511.5), np.float64(511.5), np.float64(-0.5))
```

Máscara preliminar (punto 2)



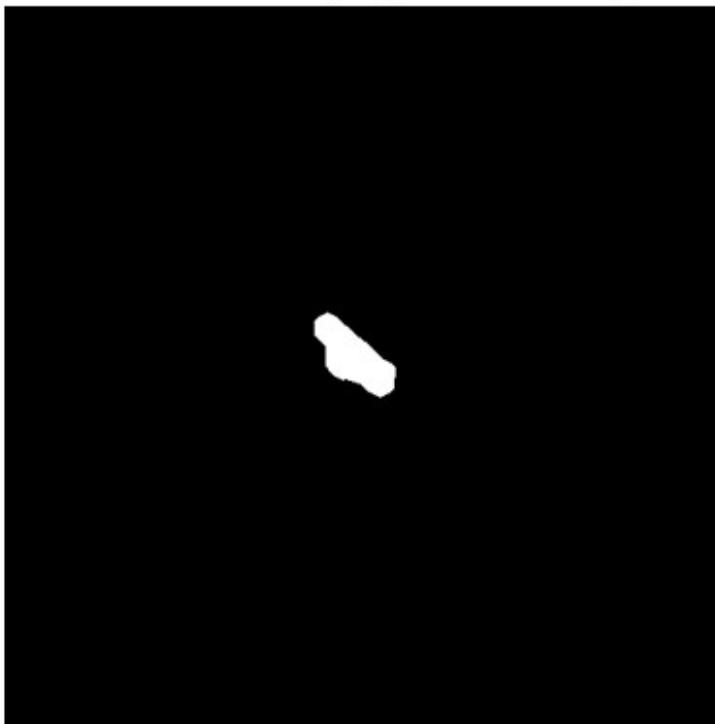
OPEN (quita ruido)



CLOSE (une / rellena)



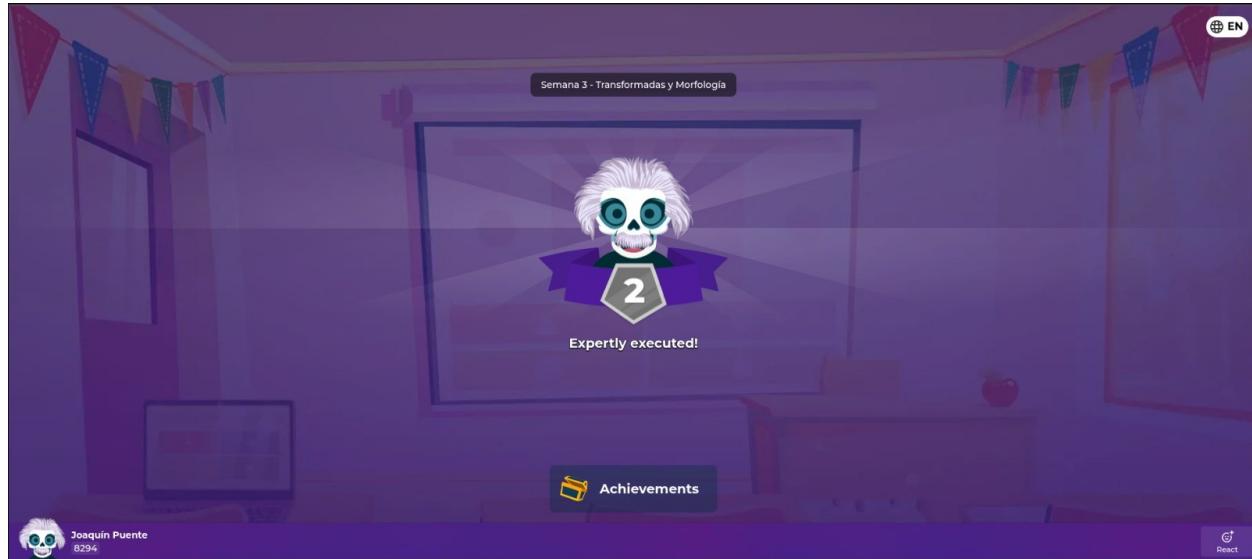
Máscara final (rasgadura blanca)



Escriba un párrafo (aprox. 100 palabras) describiendo los trade-offs de su solución. ¿Qué pasa si la rasgadura es muy pequeña? ¿Qué pasa si cambiamos el tipo de tela? ¿Es su solución robusta o específica para esta imagen?

Al suprimir la textura mediante Fourier se asume que el tejido genera patrones repetitivos y bien definidos en frecuencia; esto funciona bien para mezclilla, pero si la rasgadura es muy pequeña puede quedar atenuada junto con la textura y perderse en el filtrado. Además, los parámetros del filtro (low-pass y notch) están ajustados a una escala y patrón específicos; al cambiar el tipo de tela, por ejemplo algodón liso o telas con patrones no periódicos, el espectro en frecuencia cambia y el filtro deja de ser óptimo. Por ello, la solución no es completamente robusta: es efectiva para este tipo de imagen y condiciones similares, pero requiere reajuste para otros materiales, tamaños de defecto o condiciones de iluminación.

Puntos Extras por Kahoot de la semana:





Kahoot!

Modo clásico

Semana 3 - Transformadas y
Morfología



5.º lugar