

Procesamiento Digital de Imágenes

Práctica 6

“Filtros de Wiener”

Galante Federico, Gutiérrez David, Velasco Ricardo

*Universidad Nacional Autónoma de México
Ciudad Universitaria, Coyoacán 04510, Ciudad de México, México*

I. OBJETIVOS

De acuerdo al modelo de degradación que sufre una serie de imágenes, encontrar y aplicar el filtro de Wiener adecuado para la restauración óptima de cada imagen.

II. INTRODUCCIÓN

Considerando a $x[n]$ como un proceso discreto y estacionario en sentido amplio (Wide-Sense Stationary). Queremos determinar la respuesta al impulso unitario o la respuesta en frecuencia del sistema LTI mencionado anteriormente, de forma tal que la salida del filtro $\hat{y}[n]$ constituya el error cuadrático medio mínimo (MMSE) de un proceso objetivo $y[n]$ que es conjuntamente WSS con $x[n]$. Definiendo el error, $e[n]$, como:

$$e[n] \triangleq \hat{y}[n] - y[n], \quad (11.1)$$

Se requiere realizar la siguiente minimización:

$$\min_{h[\cdot]} \epsilon = E\{e^2[n]\}. \quad (11.2)$$

El filtro resultante, $h[n]$ se denomina filtro de Wiener para estimar $y[n]$ a partir de $x[n]$.

Para determinar la elección óptima para $h[n]$, expandimos el criterio de error en (11.2):

$$\epsilon = E \left\{ \left(\sum_{k=-\infty}^{+\infty} h[k]x[n-k] - y[n] \right)^2 \right\}. \quad (11.3)$$

Los valores de respuesta al impulso que minimizan el error pueden ser obtenidos al derivar (11.3) con respecto al filtro $h[m]$ e igualando a 0, para todos los valores de m para los que $h[m]$ no está restringido a ser cero:

$$\frac{\partial \epsilon}{\partial h[m]} = E \left\{ 2 \left(\underbrace{\sum_k h[k]x[n-k] - y[n]}_{e[n]} \right) x[n-m] \right\} = 0 . \quad (11.4)$$

La ecuación (11.4) implica que

$$E\{e[n]x[n-m]\} = 0, \text{ o } R_{ex}[m] = 0 \quad (11.5)$$

En la ecuación (11.5) se hace visible el principio de ortogonalidad: para el filtro óptimo, el error es ortogonal a todos los datos empleados para formar el estimado. Bajo nuestra asunción de $x[n]$ con media cero, la ortogonalidad es equivalente a no correlación.

Nótese que

$$\begin{aligned} R_{ex}[m] &= E\{e[n]x[n-m]\} \\ &= E\{(\hat{y}[n] - y[n])x[n-m]\} \\ &= R_{yx}^{\wedge}[m] - R_{yx}[m] . \end{aligned} \quad (11.6)$$

Por tanto, una forma alternativa de expresar el principio de ortogonalidad (11.5) es

$$R_{yx}^{\wedge}[m] = R_{yx}[m] \text{ for all appropriate } m . \quad (11.7)$$

En otras palabras, para el sistema óptimo, la correlación cruzada entre la entrada y la salida del estimador es igual a la correlación cruzada entre la entrada y la salida objetivo. Para encontrar realmente los valores de la respuesta al impulso, observe que, dado que $y[n]$ se obtiene al filtrar $x[n]$ a través de un sistema LTI con respuesta al impulso $h[n]$, la siguiente relación es válida:

$$R_{yx}^{\wedge}[m] = h[m] * R_{xx}[m] . \quad (11.8)$$

Combinando esto con la expresión alternativa de la condición de ortogonalidad, se obtiene

$$h[m] * R_{xx}[m] = R_{yx}[m] , \quad (11.9)$$

O, de forma equivalente

$$\sum_k h[k] R_{xx}[m-k] = R_{yx}[m] \quad (11.10)$$

La ecuación (11.10) representa un conjunto de ecuaciones lineales que deben resolverse para los valores de respuesta al impulso:

$$\begin{bmatrix} R_{xx}[0] & R_{xx}[-1] & \cdots & R_{xx}[1-N] \\ R_{xx}[1] & R_{xx}[0] & \cdots & R_{xx}[2-N] \\ \vdots & \vdots & \ddots & \vdots \\ R_{xx}[N-1] & R_{xx}[N-2] & \cdots & R_{xx}[0] \end{bmatrix} \begin{bmatrix} h[0] \\ h[1] \\ \vdots \\ h[N-1] \end{bmatrix} = \begin{bmatrix} R_{yx}[0] \\ R_{yx}[1] \\ \vdots \\ R_{yx}[N-1] \end{bmatrix} \quad (11.11)$$

Al obtener la transformada z a la ecuación (11.10) se obtiene:

$$H(z) S_{xx}(z) = S_{yx}(z) \quad (11.12)$$

Con lo que, la función de transferencia óptima, i.e. la función de transferencia del filtro de Wiener resultantes es entonces:

$$H(z) = S_{yx}(z)/S_{xx}(z) \quad (11.13)$$

III. DESARROLLO

1.- Filtro de Wiener con ruido añadido

```
% Cargar imagen sin ruido
ImagenSinRuido_Espacial = imread('Glaciar512.jpg');

% Generar ruido en el dominio de frecuencias (aumentar la varianza del ruido)
ImagenConRuido_Espacial = imnoise(ImagenSinRuido_Espacial, 'gaussian', 0.5);
ImagenConRuido_Frec = fft2(double(ImagenConRuido_Espacial));

% Transformada de Fourier
ImagenSinRuido_Frec = fft2(double(ImagenSinRuido_Espacial));

% Densidad espectral de la imagen original
PImagenSinRuido = abs(ImagenSinRuido_Frec).^2;

% Generar ruido en el dominio de frecuencias (aumentar la varianza del ruido)
ImagenConRuido_Espacial = imnoise(ImagenSinRuido_Espacial, 'gaussian', 0.5);
ImagenConRuido_Frec = fft2(double(ImagenConRuido_Espacial));

% Calcular la nueva varianza del ruido, evitar 0
Sn = var(double(ImagenConRuido_Espacial(:))) + 1e-10;

% Densidad espectral de la imagen con ruido (Para filtro wiener)
PImagenConRuido = abs(ImagenConRuido_Frec).^2;

% Filtro de Wiener
WienerFiltro = PImagenSinRuido ./ (PImagenSinRuido + Sn); % Filtro de Wiener

% Aplicar el filtro Wiener a la imagen con ruido
ImagenFiltradaWiener_Frec = ImagenConRuido_Frec .* WienerFiltro;

% Transformada inversa para volver al dominio espacial
ImagenFiltradaWiener_Espacial = ifft2(ImagenFiltradaWiener_Frec);

% Tomar solo la parte real
ImagenFiltradaWiener_Espacial = real(ImagenFiltradaWiener_Espacial);
%Asegurar dentro de rango
ImagenFiltradaWiener_Espacial = uint8(255 * mat2gray(ImagenFiltradaWiener_Espacial));
```

Fig 1. Código para añadir ruido a imagen y colocar filtro de mejora de ruido

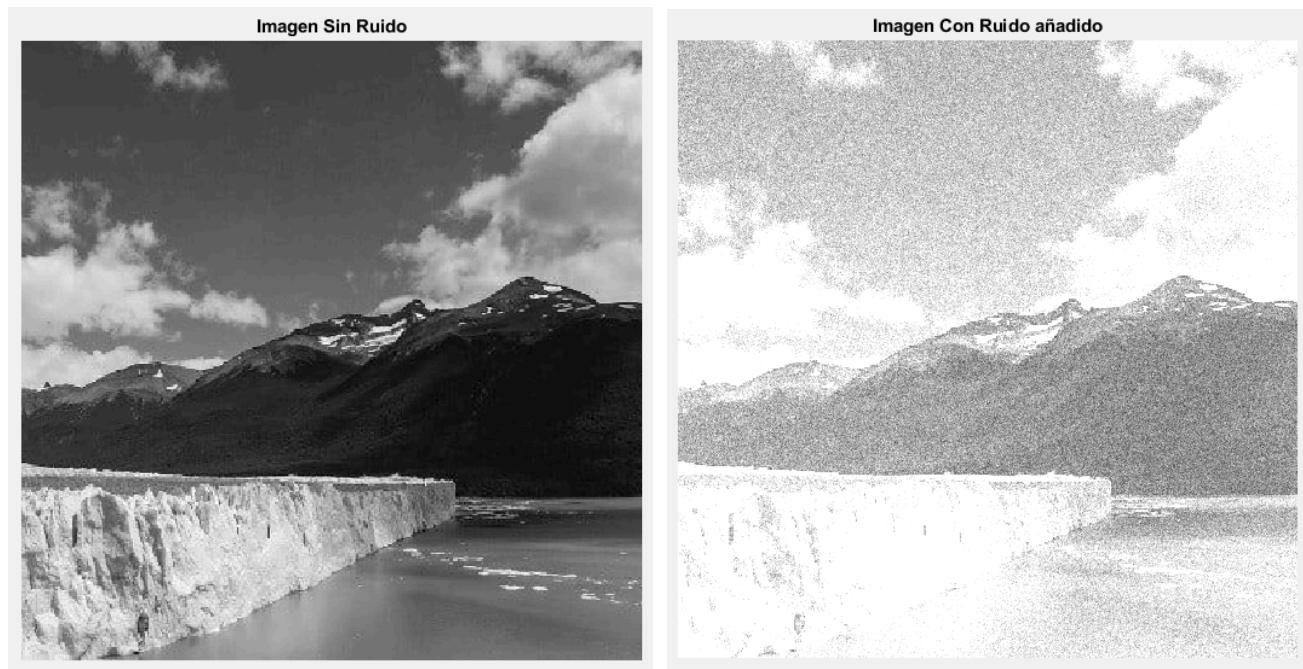


Fig 2. Comparación de imagen original e imagen con ruido añadido (0.5)

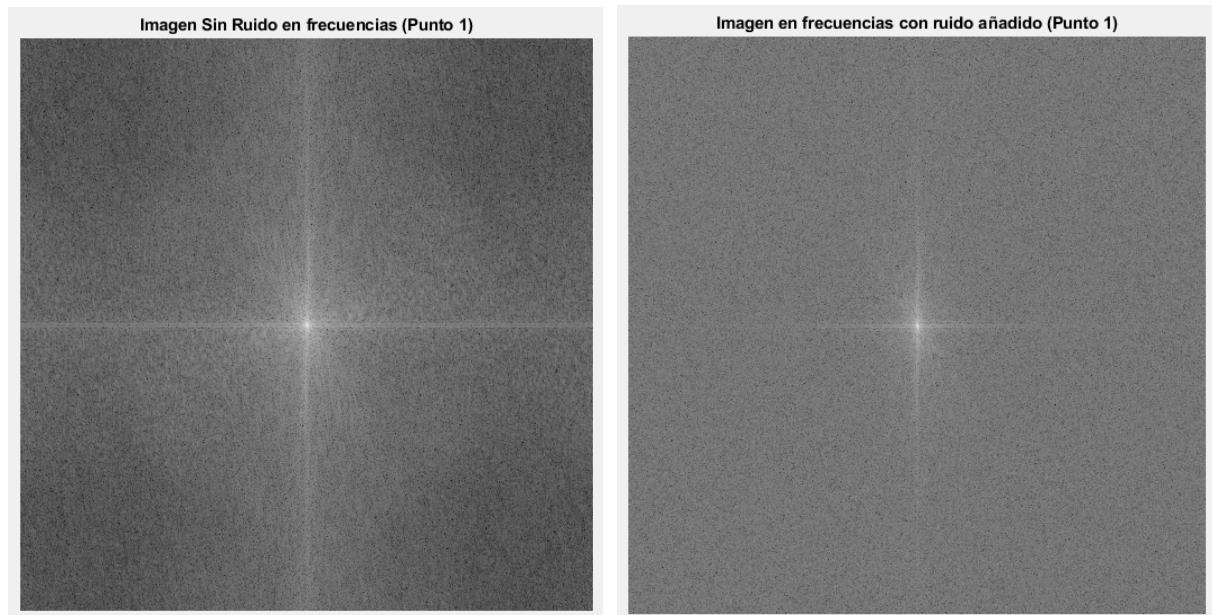


Fig 3. Comparación en el campo de las frecuencias de imagen original e imagen con ruido añadido

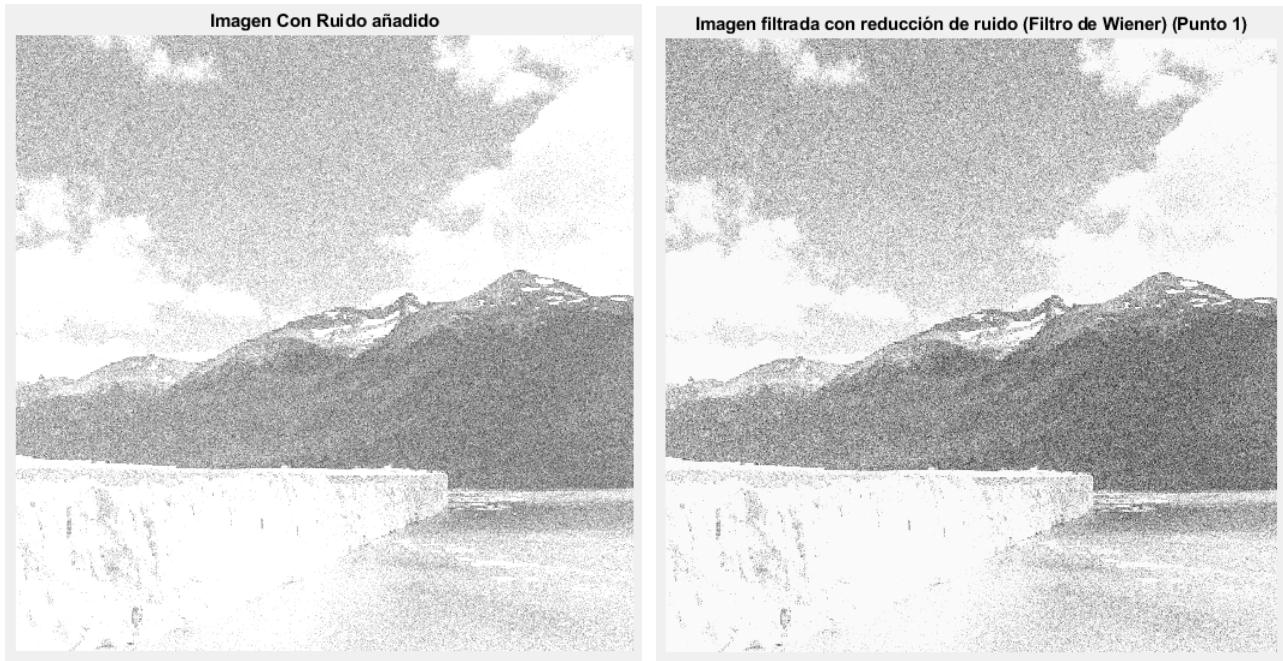


Fig 4. Comparación de imágenes tras aplicar el filtro Wiener

2.- Filtro de Wiener con Nitidez añadida

```
%Punto 2
% G = F . H (En dominio de frecuencias)

% Filtro binomial 9x9 (Matriz H)
FiltroBinomial_Espacial = [1 8 28 56 70 56 28 8 1]' * [1 8 28 56 70 56 28 8 1];
FiltroBinomial_Espacial = FiltroBinomial_Espacial / sum(FiltroBinomial_Espacial(:)); % Normalizar

% Transformada de Fourier del filtro (Matriz H)
Filtro_Frec = fft2(FiltroBinomial_Espacial, 512, 512); % Padding del filtro

% Multiplicacion en dominio de frecuencias (Matriz G)
ImagenFiltrada_Frec = ImagenSinRuido_Frec .* Filtro_Frec;

% Transformada inversa para volver al dominio espacial
ImagenFiltrada_Espacial = ifft2(ImagenFiltrada_Frec);

% Calcular la matriz inversa del filtro
% en el dominio de la frecuencia (Matriz H^-1)
FiltroFrecInverso = zeros(size(Filtro_Frec));

for i = 1: numel(Filtro_Frec) % Evitar div entre 0
    if Filtro_Frec(i) ~= 0
        FiltroFrecInverso(i) = 1 / Filtro_Frec(i);
    end
end

% Producto de las matrices G y H^-1
ImagenFiltroPasoBajasHInversa_Frec = ImagenFiltrada_Frec .* FiltroFrecInverso;

% Transformada inversa para volver al dominio espacial
ImagenFiltroPasoBajasHInversa_Espacial = ifft2(ImagenFiltroPasoBajasHInversa_Frec);
```

Fig 5. Código para añadir nitidez a la imagen y colocar filtro de mejora de ruido

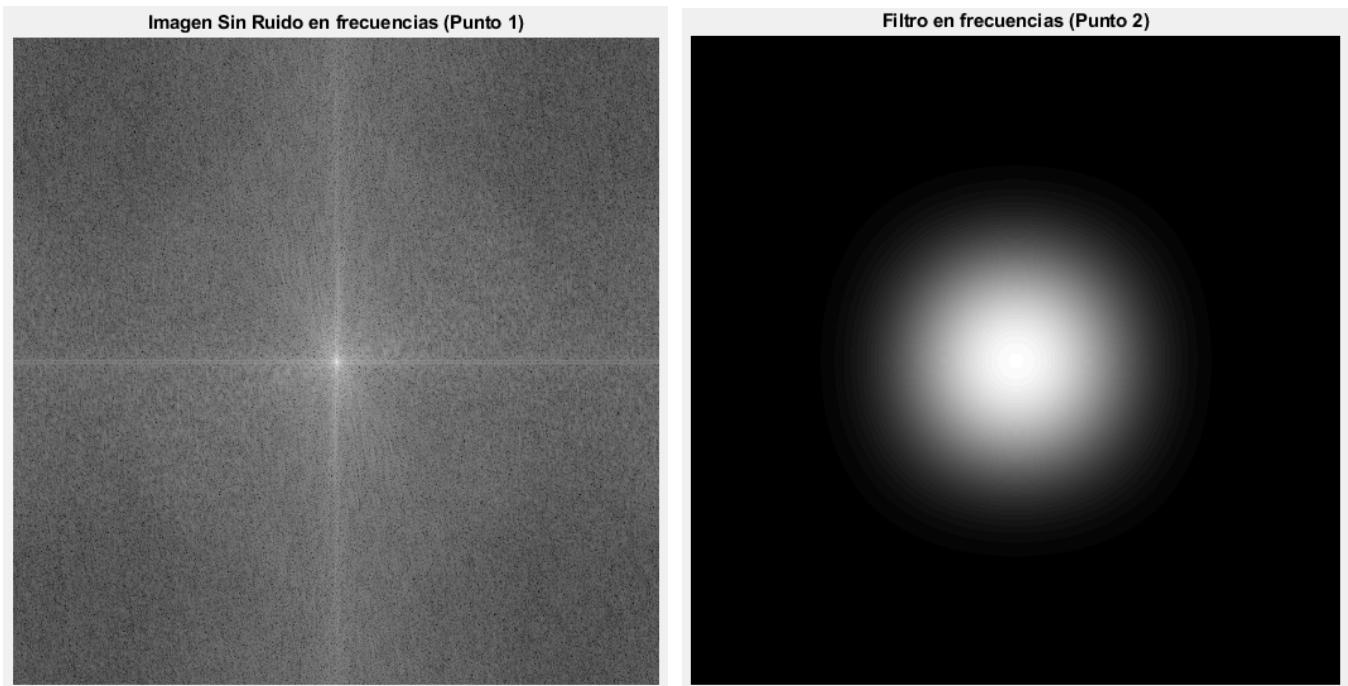


Fig 6. Comparación en el campo de las frecuencias de imagen original y del filtro paso bajas 9x9

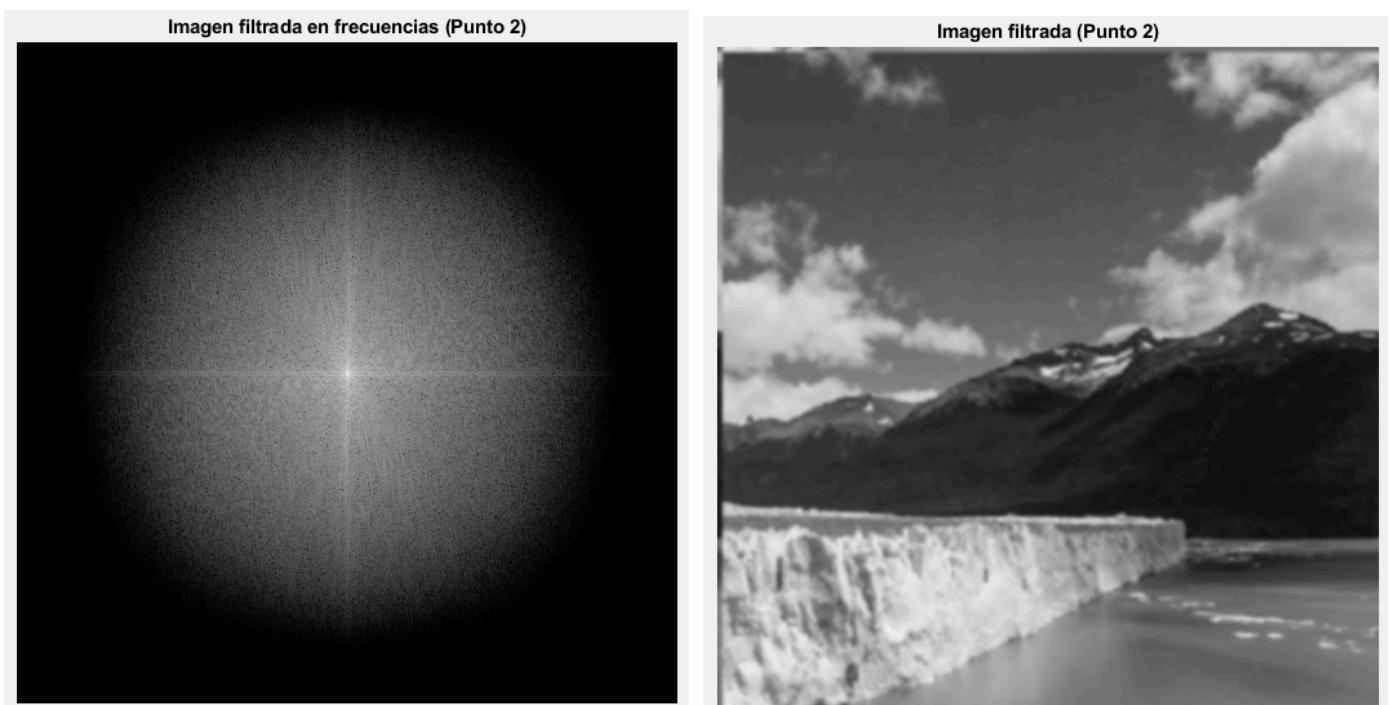


Fig 7. Comparación de convolución en el campo de las frecuencias y resultado en el campo espacial

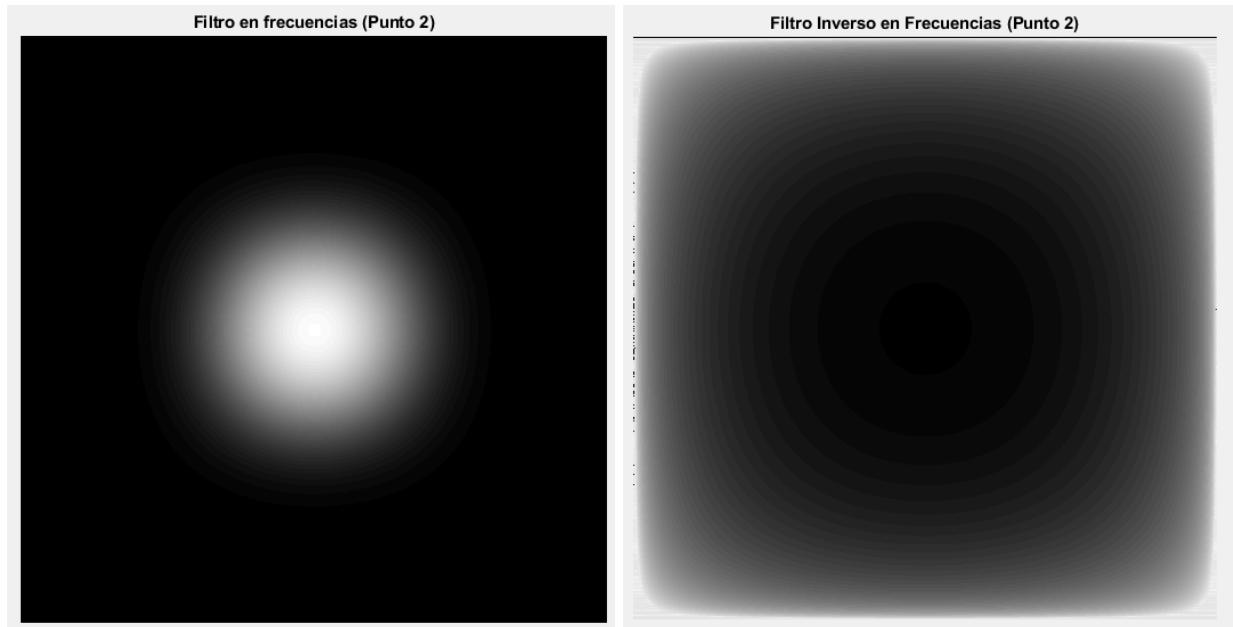


Fig 8. Comparación en el campo de las frecuencias de filtro paso bajas y su inversa

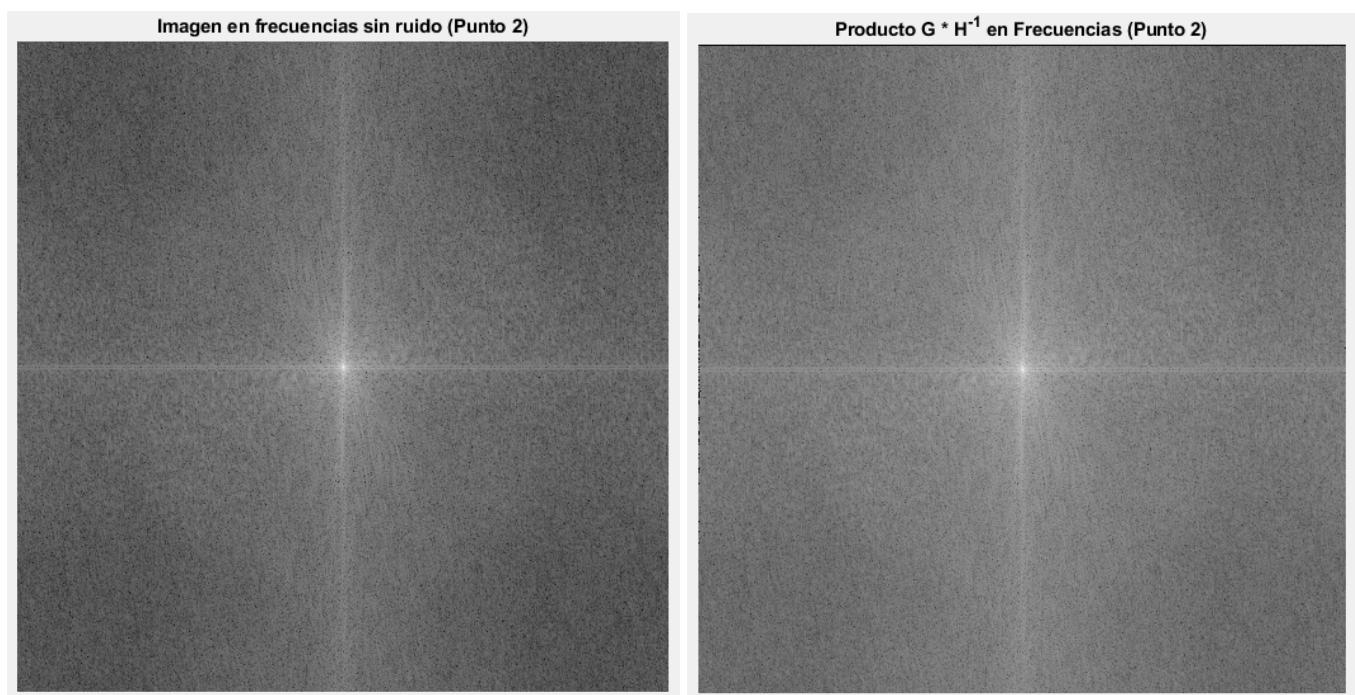


Fig 9. Comparación en el campo de las frecuencias de imagen original y su proceso de convolución con el filtro

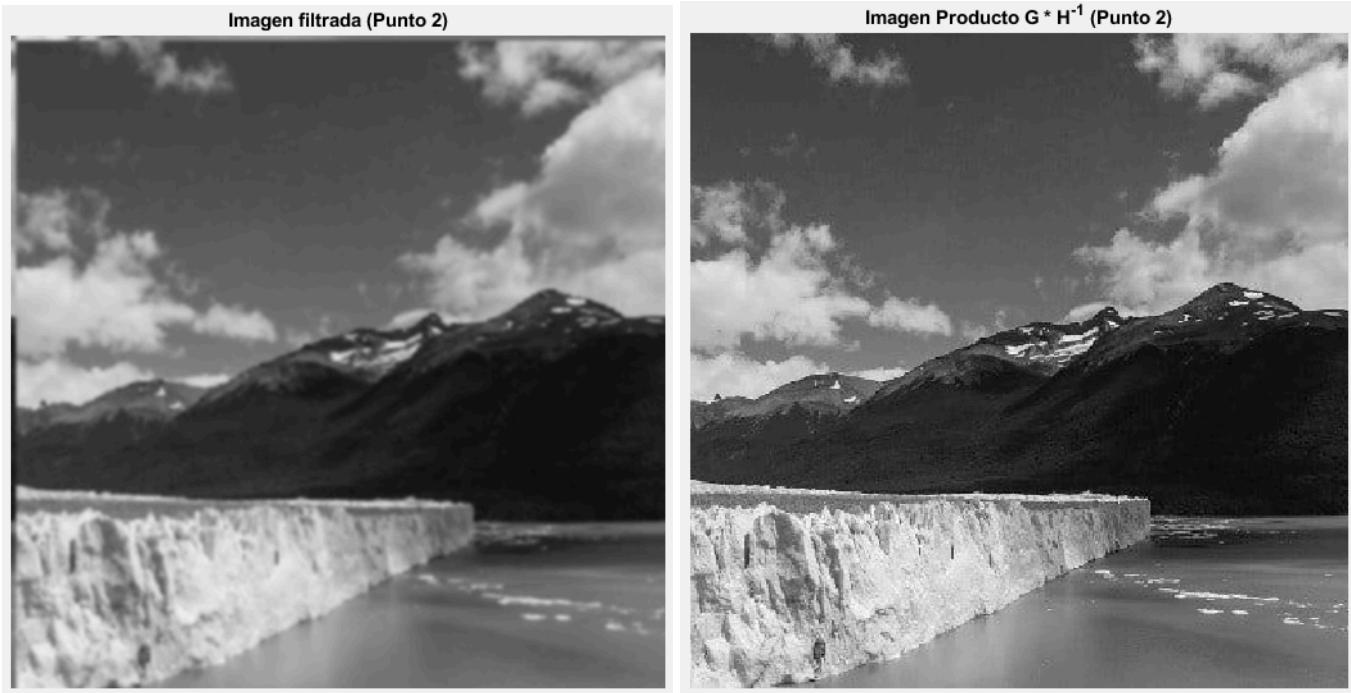


Fig 10. Comparación en el campo espacial de imagen filtrada y su resultado tras aplicar filtro de Wiener

3.- Restauración de imagen que se le aplicó ruido gaussiano y pérdida de nitidez

```

def add_gaussian_noise(image, mean=0, variance=0.01):
    """Aregar ruido gaussiano a una imagen."""
    row, col = image.shape
    sigma = np.sqrt(variance)
    gauss = np.random.normal(mean, sigma, (row, col))
    noisy = image + gauss * 255
    return np.clip(noisy, 0, 255).astype(np.uint8)

def create_lowpass_filter(size=9):
    """Crear un filtro paso bajas normalizado de tamaño size x size."""
    kernel = np.ones((size, size)) / (size * size)
    return kernel

def degradation_function(image, kernel):
    """Aplicar la función de degradación en el dominio de la frecuencia con shift correcto."""
    image_float = image.astype(float) / 255.0
    kernel_padded = np.zeros_like(image_float)
    kh, kw = kernel.shape
    kernel_padded[:kh, :kw] = kernel
    kernel_padded = np.roll(kernel_padded, -(kh//2), axis=0)
    kernel_padded = np.roll(kernel_padded, -(kw//2), axis=1)
    image_fft = fftpack.fft2(image_float)
    kernel_fft = fftpack.fft2(kernel_padded)
    degraded_fft = image_fft * kernel_fft
    degraded = np.real(fftpack.ifft2(degraded_fft))
    return np.clip(degraded * 255, 0, 255).astype(np.uint8)

```

```

def wiener_filter_restore(degraded_image, kernel, K=0.02):
    """Restauración usando el filtro de Wiener con manejo correcto del shift."""
    image_float = degraded_image.astype(float) / 255.0
    kernel_padded = np.zeros_like(image_float)
    kh, kw = kernel.shape
    kernel_padded[:kh, :kw] = kernel
    kernel_padded = np.roll(kernel_padded, -(kh//2), axis=0)
    kernel_padded = np.roll(kernel_padded, -(kw//2), axis=1)
    degraded_fft = fftpack.fft2(image_float)
    kernel_fft = fftpack.fft2(kernel_padded)
    kernel_power_spectrum = np.abs(kernel_fft) ** 2
    wiener = np.conj(kernel_fft) / (kernel_power_spectrum + K)
    restored_fft = degraded_fft * wiener
    restored = np.real(fftpack.ifft2(restored_fft))
    return np.clip(restored * 255, 0, 255).astype(np.uint8), kernel_power_spectrum

```

Fig 11. Funciones usadas de apoyo en python para la aplicación de los filtros de ruido y degradado

```

# Cargar imagen
image = cv2.imread('Glaciar512.jpg', cv2.IMREAD_GRAYSCALE)
if image is None:
    raise ValueError("No se pudo cargar la imagen")

# Parámetros
NOISE_MEAN = 0
NOISE_VAR = 0.01
K_CASE3 = 0.005
K_CASE4 = 0.03
LOWPASS_KERNEL_SIZE_CASE3 = 9
LOWPASS_KERNEL_SIZE_CASE4 = 9

# Crear filtros paso bajas específicos para cada caso
lowpass_kernel_case3 = create_lowpass_filter(LOWPASS_KERNEL_SIZE_CASE3)
lowpass_kernel_case4 = create_lowpass_filter(LOWPASS_KERNEL_SIZE_CASE4)

# Guardar imágenes
output_dir = "output_images"
os.makedirs(output_dir, exist_ok=True)

cv2.imwrite(os.path.join(output_dir, 'original_image.jpg'), image)
cv2.imwrite(os.path.join(output_dir, 'case3_degraded.jpg'), degraded_case3)
cv2.imwrite(os.path.join(output_dir, 'case3_restored.jpg'), restored_case3)
cv2.imwrite(os.path.join(output_dir, 'case4_degraded.jpg'), degraded_case4)
cv2.imwrite(os.path.join(output_dir, 'case4_restored.jpg'), restored_case4)

# Guardar FFT de imágenes como imágenes JPG
def save_fft_as_image(fft_data, filename):
    """Guardar el espectro FFT como imagen JPG."""
    # Calcular la magnitud y escalar
    fft_magnitude = np.abs(fft_data)
    # Escalar a 0-255 para guardar como imagen
    fft_scaled = np.log1p(fft_magnitude) # Usar log para mejorar la visualización
    fft_scaled = (fft_scaled / np.max(fft_scaled) * 255).astype(np.uint8)
    plt.imsave(filename, fft_scaled, cmap='gray')

```

Fig 12. Código de apoyo para factores iniciales e impresión de imágenes

```

# Caso 3: Ruido gaussiano + pérdida de nitidez
print("Procesando Caso 3: Ruido gaussiano + pérdida de nitidez")
noisy_image = add_gaussian_noise(image, NOISE_MEAN, NOISE_VAR)
degraded_case3 = degradation_function(noisy_image, lowpass_kernel_case3)
restored_case3, kernel_power_spectrum_case3 = wiener_filter_restore(degraded_case3, lowpass_kernel_case3, K_CASE3)

```

Fig 13. Código de apoyo para aplicar filtro de Wiener creado para el punto 3

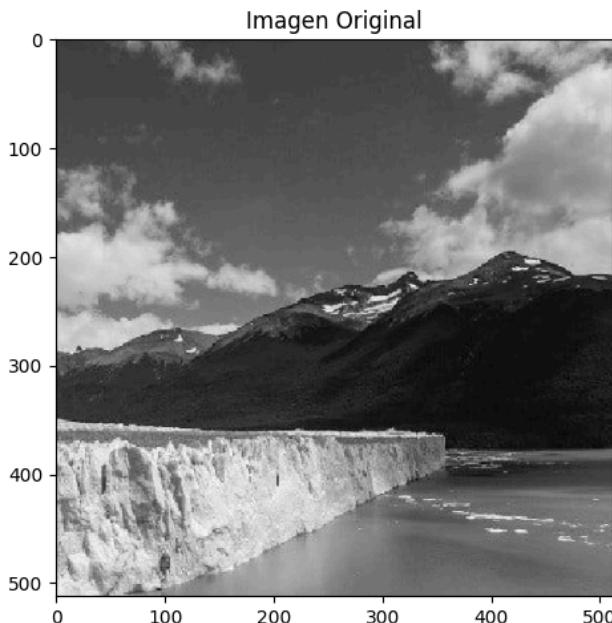


Fig 14. Muestra de imagen original usada

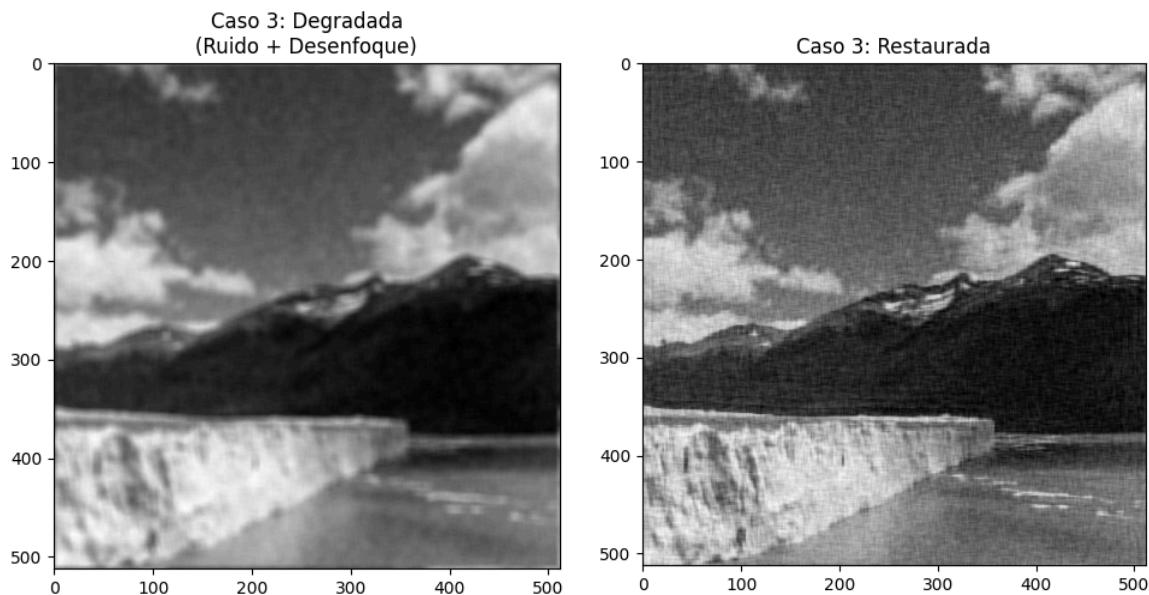
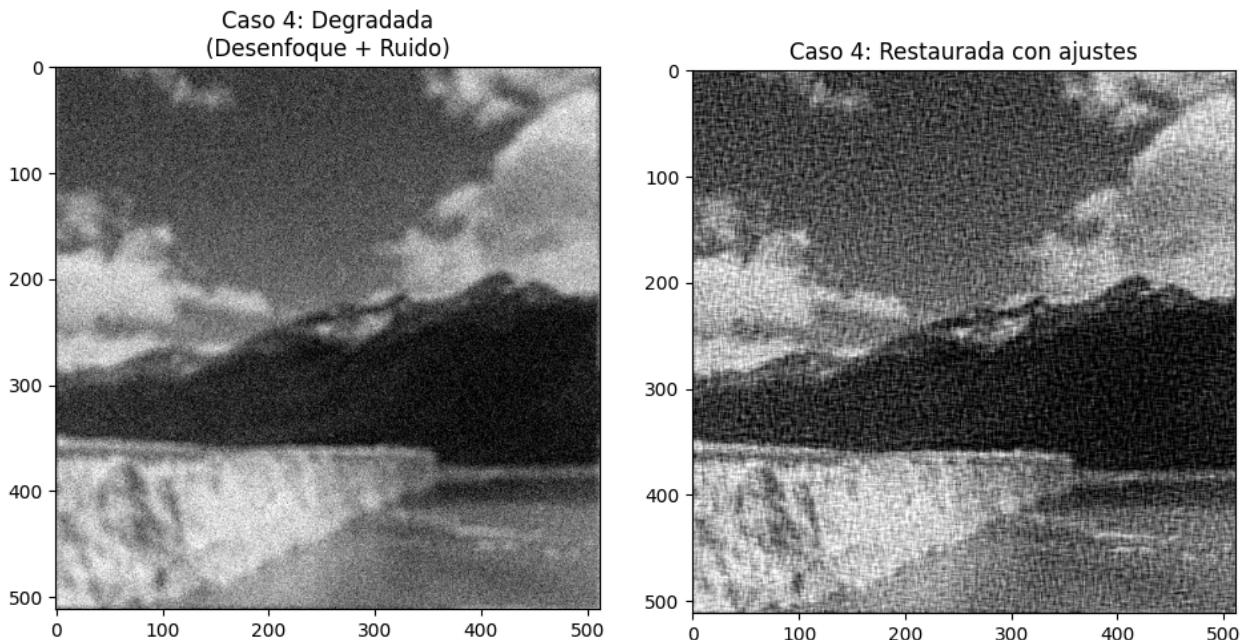


Fig 15. Comparación de resultados tras degradación del paso 3 y su restauración

4.- Restauración de imagen que se le aplicó pérdida de nitidez y ruido

```
# Caso 4: Pérdida de nitidez + ruido gaussiano (con ajustes adicionales)
print("Procesando Caso 4: Pérdida de nitidez + ruido gaussiano con ajustes")
blurred_image = degradation_function(image, lowpass_kernel_case4)
degraded_case4 = add_gaussian_noise(blurred_image, NOISE_MEAN, NOISE_VAR)
restored_case4, kernel_power_spectrum_case4 = wiener_filter_restore(degraded_case4, lowpass_kernel_case4, K_CASE4)
```

Fig 16. Funciones usadas de apoyo en python para la aplicación de los filtros de degradado y ruido



IV. RESULTADOS

En esta práctica se observó cómo varía una imagen al aplicar diferentes tipos de filtros utilizando el software de MatLab, enfocándonos en las transformadas discretas de Fourier, uso de padding y sus inversas (Aplicándolo con imágenes tanto filtradas como no filtradas, usando versiones con ruido y sin ruido).

Se pudo observar como en las imágenes en frecuencia se puede alcanzar a contar las divisiones de acuerdo al tamaño de la matriz usada para la transformada. Además de lo anterior, se nota como en las imágenes que cuentan con ruido el difuminado es menos marcado.

Se notó como en las inversas directas y con zero padding había una diferencia menor entre los bordes, y cuando esto se usaba con las imágenes previamente filtradas por el filtro paso bajas de bloque los bordes tienden a ser ligeramente más oscuros.

V. CONCLUSIONES

En conclusión, se considera que la práctica fue realizada de manera exitosa en su mayor parte. Se logró codificar y visualizar el funcionamiento de los filtros en el procesamiento de imágenes utilizando la Transformada Discreta de Fourier (DFT). Se observaron los efectos de los filtros de paso bajo, que suavizan las imágenes, y los filtros de paso alto, que resaltan los bordes, confirmando los conceptos teóricos aprendidos, aunque cabe añadir que algunos de los filtros de restauración no presentaban la fuerza necesaria para restaurar la imagen cuando el ruido añadido era más bajo. Añadiendo a lo anterior, se pudo notar como al realizar el distinto orden de los filtros de degradación y de ruido añadido se presentaron resultados distintos durante la restauración, indicando que el orden de manipulación de la imagen es importante.

Además, se evidenció la importancia del zero-padding en la convolución, mejorando la representación de las imágenes filtradas. Estos resultados resaltan la relevancia de la DFT en la mejora de la calidad de las imágenes, aportando una comprensión práctica del filtrado en el dominio de la frecuencia.

VI. REFERENCIAS

- [1] RAFAEL C. GONZALEZ, RICHARD E. WOODS, DIGITAL IMAGE PROCESSING (3E), PEARSON INTERNATIONAL EDITION, 2008.
- [2] CONCEPTOS FUNDAMENTALES EN EL DESARROLLO DE FILTROS, FERNANDEZ F.F., TEMA 2, CONCEPTOS FUNDAMENTALES EN EL DISEÑO DE FILTROS
- [3] SIGMUR, UNIVERSIDAD DE MURCIA, TELEDETECCIÓN, TEMA 6: TÉCNICAS DE FILTRADO