符昕宇 202364810311

**To access my code more easily, you can find them at my github repository:**

**https://github.com/Fluorine-Brian/Code-for-Digital-Image-Processing**

## Homework1: Implement Figure 4.40 by programming with Python

### Python Code:

```
import numpy as np
import imageio.v2 as imageio
import matplotlib
import matplotlib.pyplot as plt
import os


def pad_image_for_dft(image):
    """
    Pads the image to a size suitable for DFT
    """
    M, N = image.shape
    P, Q = 2 * M, 2 * N
    padded_image = np.zeros((P, Q), dtype=image.dtype)
    padded_image[0:M, 0:N] = image
    return padded_image, (M, N)


def calculate_power_percentage(centered_dft_spectrum, D0):
    """
    Calculates the percentage of total power contained within a circle of radius D0
    """
    P, Q = centered_dft_spectrum.shape
    power_spectrum = np.abs(centered_dft_spectrum) ** 2
    total_power = np.sum(power_spectrum)
    power_within_D0 = 0.0
    center_u, center_v = P / 2, Q / 2

    u_coords = np.arange(P) - P / 2
    v_coords = np.arange(Q) - Q / 2
    U, V = np.meshgrid(u_coords, v_coords, indexing='ij')
    D_uv = np.sqrt(U ** 2 + V ** 2)

    power_within_D0 = np.sum(power_spectrum[D_uv <= D0])
    if total_power > 0:
        return (power_within_D0 / total_power) * 100
    else:
        return 0.0


if __name__ == "__main__":
```

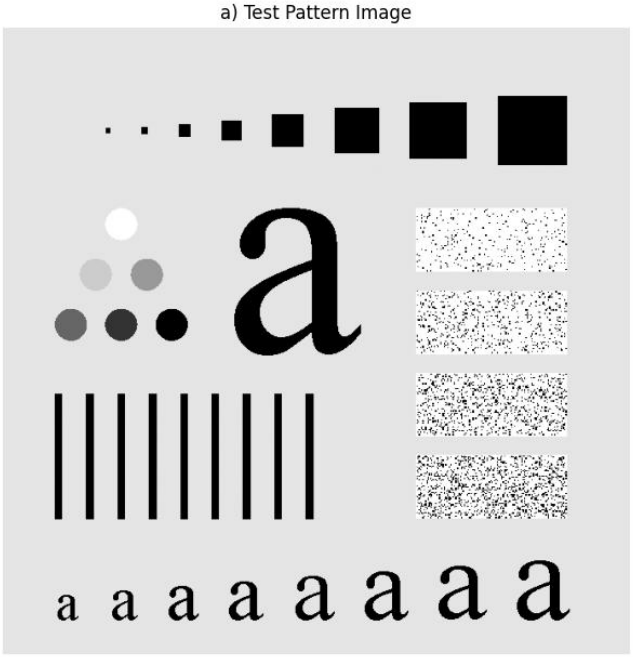
```
    input_dir = "original_image"
    output_dir = "output_image"
    os.makedirs(output_dir, exist_ok=True)

    image_filename = "Fig0440.tif"
    image_path = os.path.join(input_dir, image_filename)
    base_name = os.path.splitext(image_filename)[0]

    original_image = imageio.imread(image_path)
    original_M, original_N = original_image.shape

    padded_image, _ = pad_image_for_dft(original_image)
    P, Q = padded_image.shape

    dft_original = np.fft.fft2(padded_image.astype(float))
    centered_dft = np.fft.fftshift(dft_original)

    spectrum_log_magnitude = 20 * np.log(np.abs(centered_dft) + 1e-9)

    spectrum_display = (spectrum_log_magnitude - np.min(spectrum_log_magnitude)) /
                       (np.max(spectrum_log_magnitude) - np.min(spectrum_log_magnitude)) * 255
    spectrum_display = spectrum_display.astype(np.uint8)

    D0_values = [10, 30, 60, 160, 460]
    power_percentages = {}
    for D0 in D0_values:
        power_percent = calculate_power_percentage(centered_dft, D0)
        power_percentages[D0] = power_percent

    plt.figure(figsize=(14, 7))
    plt.suptitle(f'Figure 4.40: Test Pattern and its Frequency Spectrum for {image_filename}',
fontsize=16)

    plt.subplot(1, 2, 1)
    plt.imshow(original_image, cmap='gray', vmin=0, vmax=255)
    plt.title('a) Test Pattern Image')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(spectrum_display, cmap='gray', vmin=0, vmax=255)
    plt.title('b) Frequency Spectrum with D0 Circles')
    plt.axis('off')
    center_x, center_y = Q / 2, P / 2
    for D0 in D0_values:
        circle = plt.Circle((center_x, center_y), D0, color='red', fill=False, linewidth=1, alpha=0.7)
        plt.gca().add_patch(circle)
        text_x = center_x + D0 * np.cos(np.deg2rad(45))center
        text_y = center_y + D0 * np.sin(np.deg2rad(45))
```

```
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
combined_output_path = os.path.join(output_dir, f"{base_name}_fig4_40_reproduction.png")
plt.savefig(combined_output_path, bbox_inches='tight')
plt.close()
```

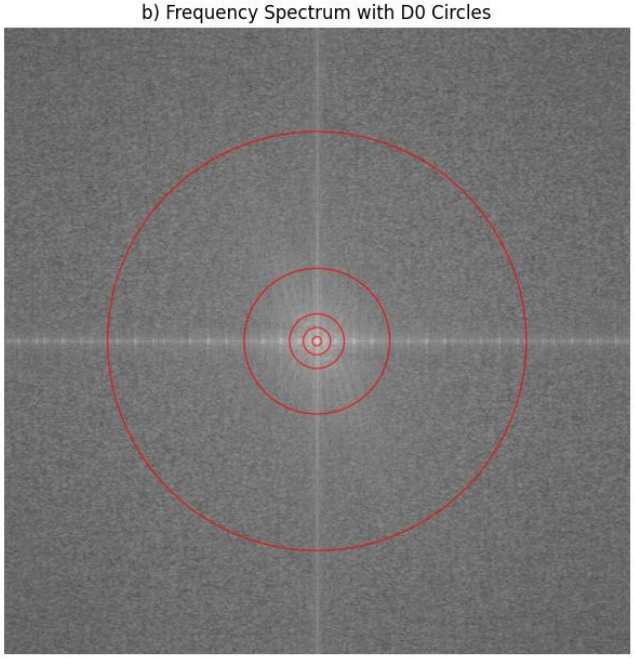
## Processed Images:

Figure 4.40: Test Pattern and its Frequency Spectrum for Fig0440.tif

a) Test Pattern Image



b) Frequency Spectrum with D0 Circles



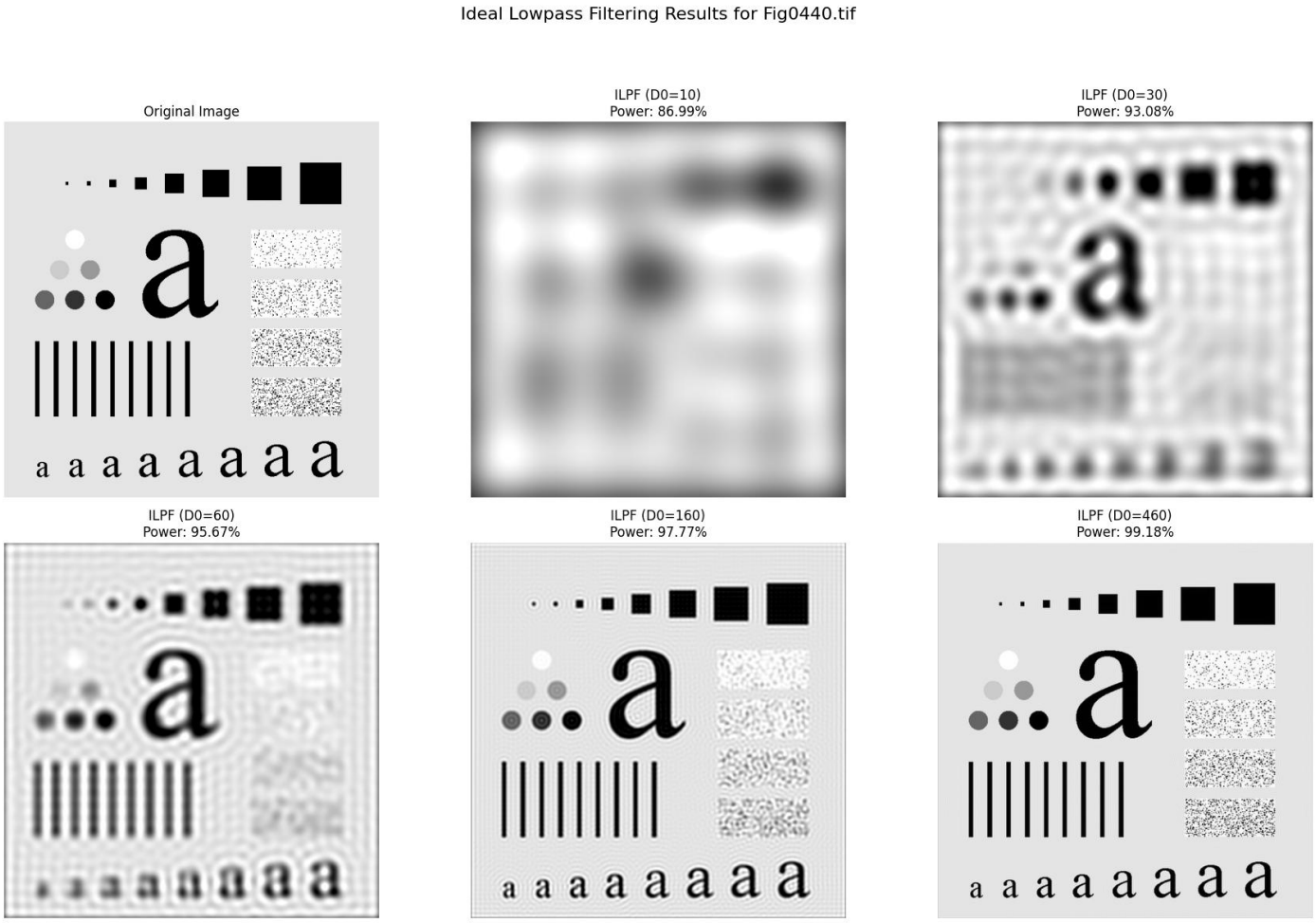
## Homework 2: Implement Figure 4.41 by programming with Python

## Python Code:

```
import numpy as np
import imageio.v2 as imageio
import matplotlib.pyplot as plt
import os


def pad_image_for_dft(image):
    """
    Pads the image to a size suitable for DFT (e.g., 2*M x 2*N)
    """
    M, N = image.shape
    P, Q = 2 * M, 2 * N  # Double the dimensions
    padded_image = np.zeros((P, Q), dtype=image.dtype)
    padded_image[0:M, 0:N] = image

    return padded_image, (M, N)


def calculate_power_percentage(centered_dft_spectrum, D0):
    """
    Calculates the percentage of total power contained within a circle of radius D0
    """
    P, Q = centered_dft_spectrum.shape
    power_spectrum = np.abs(centered_dft_spectrum) ** 2
    total_power = np.sum(power_spectrum)
    power_within_D0 = 0.0
    center_u, center_v = P / 2, Q / 2

    u_coords = np.arange(P) - P / 2
    v_coords = np.arange(Q) - Q / 2
    U, V = np.meshgrid(u_coords, v_coords, indexing='ij')
    D_uv = np.sqrt(U ** 2 + V ** 2)

    power_within_D0 = np.sum(power_spectrum[D_uv <= D0])

    if total_power > 0:
        return (power_within_D0 / total_power) * 100
    else:
        return 0.0


if __name__ == "__main__":
    input_dir = "original_image"
    output_dir = "output_image"
    os.makedirs(output_dir, exist_ok=True)
    image_filename = "Fig0440.tif"
```

```
    image_path = os.path.join(input_dir, image_filename)
    base_name = os.path.splitext(image_filename)[0]

    original_image = imageio.imread(image_path)
    original_M, original_N = original_image.shape

    padded_image, _ = pad_image_for_dft(original_image)
    P, Q = padded_image.shape

    dft_original = np.fft.fft2(padded_image.astype(float))
    centered_dft = np.fft.fftshift(dft_original)
    spectrum_log_magnitude = 20 * np.log(np.abs(centered_dft) + 1e-9)
    spectrum_display = (spectrum_log_magnitude - np.min(spectrum_log_magnitude)) / \
                       (np.max(spectrum_log_magnitude) - np.min(spectrum_log_magnitude)) * 255
    spectrum_display = spectrum_display.astype(np.uint8)
    D0_values = [10, 30, 60, 160, 460]
    power_percentages = {}

    for D0 in D0_values:
        power_percent = calculate_power_percentage(centered_dft, D0)
        power_percentages[D0] = power_percent

    plt.figure(figsize=(14, 7))
    plt.suptitle(f'Figure 4.40: Test Pattern and its Frequency Spectrum for {image_filename}',
fontsize=16)

    plt.subplot(1, 2, 1)
    plt.imshow(original_image, cmap='gray', vmin=0, vmax=255)
    plt.title('a) Test Pattern Image')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(spectrum_display, cmap='gray', vmin=0, vmax=255)
    plt.title('b) Frequency Spectrum with D0 Circles')
    plt.axis('off')

    center_x, center_y = Q / 2, P / 2
    for D0 in D0_values:
        circle = plt.Circle((center_x, center_y), D0, color='red', fill=False, linewidth=1, alpha=0.7)
        plt.gca().add_patch(circle)

        text_x = center_x + D0 * np.cos(np.deg2rad(45))
        text_y = center_y + D0 * np.sin(np.deg2rad(45))

    plt.tight_layout(rect=[0, 0.03, 1, 0.95])

    combined_output_path = os.path.join(output_dir, f"{base_name}_fig4_40_reproduction.png")
    plt.savefig(combined_output_path, bbox_inches='tight')
    plt.close()
```

## Processed Images:



## Homework3: Implement Figure 4.44 by programming with Python

## Python Code:

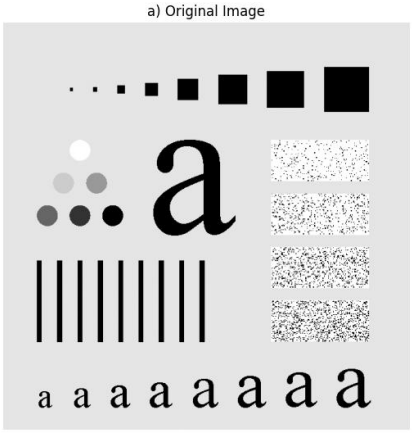
```
import numpy as np
import imageio.v2 as imageio
import matplotlib
import matplotlib.pyplot as plt
import os


def pad_image_for_dft(image, padding_mode='reflect'):
    """
    Pads the image to a size suitable for DFT (e.g., 2*M x 2*N)
    """
    M, N = image.shape
    P, Q = 2 * M, 2 * N
    pad_h = P - M
    pad_w = Q - N
    padded_image = np.pad(image, ((0, pad_h), (0, pad_w)), mode=padding_mode)

    return padded_image, (M, N)


def create_gaussian_lowpass_filter(shape, D0):
    """
    Creates a Gaussian Lowpass Filter (GLPF) in the frequency domain
    """
    P, Q = shape
    H = np.zeros((P, Q), dtype=float)
    center_u, center_v = P / 2, Q / 2

    u_coords = np.arange(P) - center_u
    v_coords = np.arange(Q) - center_v
    U, V = np.meshgrid(u_coords, v_coords, indexing='ij')
    D_uv_squared = U ** 2 + V ** 2

    if D0 == 0:
        H = np.ones(shape, dtype=float)
    else:
        H = np.exp(-D_uv_squared / (2 * D0 ** 2))
    return H


if __name__ == "__main__":
    input_dir = "original_image"
    output_dir = "output_image"
    os.makedirs(output_dir, exist_ok=True)
    image_filename = "Fig0440.tif"
    image_path = os.path.join(input_dir, image_filename)
```

```
base_name = os.path.splitext(image_filename)[0]

original_image = imageio.imread(image_path)
original_M, original_N = original_image.shape
padded_image, _ = pad_image_for_dft(original_image, padding_mode='reflect')
P, Q = padded_image.shape

dft_original = np.fft.fft2(padded_image.astype(float))
centered_dft = np.fft.fftshift(dft_original)
D0_values = [10, 30, 60, 160, 460]
filtered_images = {}

for D0 in D0_values:
    H = create_gaussian_lowpass_filter((P, Q), D0)
    filtered_dft = centered_dft * H
    idft_shifted = np.fft.ifftshift(filtered_dft)
    filtered_image_complex = np.fft.ifft2(idft_shifted)

    filtered_image = np.real(filtered_image_complex)[0:original_M, 0:original_N]
    filtered_image = np.clip(filtered_image, 0, 255).astype(np.uint8)
    filtered_images[D0] = filtered_image

plt.figure(figsize=(18, 12))
plt.suptitle(f'Gaussian Lowpass Filtering Results for {image_filename}', fontsize=16)

plt.subplot(2, 3, 1)
plt.imshow(original_image, cmap='gray', vmin=0, vmax=255)
plt.title('a) Original Image')
plt.axis('off')
subplot_labels = ['b', 'c', 'd', 'e', 'f']
for i, D0 in enumerate(D0_values):
    plt.subplot(2, 3, i + 2)
    plt.imshow(filtered_images[D0], cmap='gray', vmin=0, vmax=255)
    plt.title(f'{subplot_labels[i]}) GLPF (D0={D0})')
    plt.axis('off')

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
combined_output_path = os.path.join(output_dir, f"{base_name}_glpf_results_combined.png")
plt.savefig(combined_output_path, bbox_inches='tight')
plt.close()
```

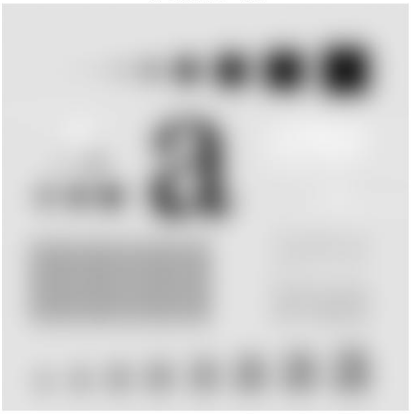
**Processed Images:**

Gaussian Lowpass Filtering Results for Fig0440.tif

a) Original Image



b) GLPF (D0=10)

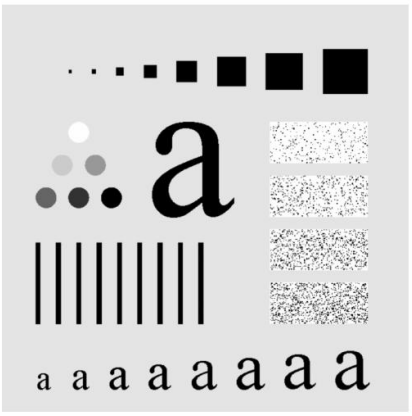


c) GLPF (D0=30)

d) GLPF (D0=60)

e) GLPF (D0=160)

f) GLPF (D0=460)



## Homework4: Implement Figure 4.46 by programming with Python

## Python Code:

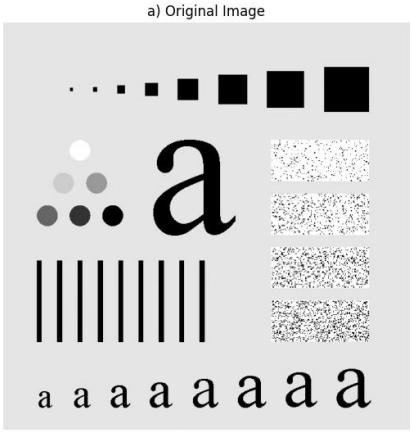
```
import numpy as np
import imageio.v2 as imageio
import matplotlib
import matplotlib.pyplot as plt
import os


def pad_image_for_dft(image, padding_mode='reflect'):
    """
    Pads the image to a size suitable for DFT (e.g., 2*M x 2*N)
    """
    M, N = image.shape
    P, Q = 2 * M, 2 * N
    pad_h = P - M
    pad_w = Q - N
    padded_image = np.pad(image, ((0, pad_h), (0, pad_w)), mode=padding_mode)
    return padded_image, (M, N)


def create_butterworth_lowpass_filter(shape, D0, n):
    """
    Creates a Butterworth Lowpass Filter (BLPF) in the frequency domain
    """
    P, Q = shape
    H = np.zeros((P, Q), dtype=float)
    center_u, center_v = P / 2, Q / 2
    u_coords = np.arange(P) - center_u
    v_coords = np.arange(Q) - center_v
    U, V = np.meshgrid(u_coords, v_coords, indexing='ij')
    D_uv = np.sqrt(U ** 2 + V ** 2)

    if D0 == 0:
        H = np.ones(shape, dtype=float)  # Pass all frequencies if D0 is 0
    else:
        H = 1 / (1 + (D_uv / D0) ** (2 * n))
        H[int(center_u), int(center_v)] = 1.0
    return H


if __name__ == "__main__":
    input_dir = "original_image"
    output_dir = "output_image"
    os.makedirs(output_dir, exist_ok=True)
    image_filename = "Fig0440.tif"
    image_path = os.path.join(input_dir, image_filename)
    base_name = os.path.splitext(image_filename)[0]
```

```
    original_image
    original_M, original_N = original_image.shape

    padded_image, _ = pad_image_for_dft(original_image, padding_mode='reflect')
    P, Q = padded_image.shape
    dft_original = np.fft.fft2(padded_image.astype(float))
    centered_dft = np.fft.fftshift(dft_original)
    D0_values = [10, 30, 60, 160, 460]
    n_order = 2.25
    filtered_images = {}

    for D0 in D0_values:
        H = create_butterworth_lowpass_filter((P, Q), D0, n_order)
        filtered_dft = centered_dft * H
        idft_shifted = np.fft.ifftshift(filtered_dft)
        filtered_image_complex = np.fft.ifft2(idft_shifted)
        filtered_image = np.real(filtered_image_complex)[0:original_M, 0:original_N]
        filtered_image = np.clip(filtered_image, 0, 255).astype(np.uint8)
        filtered_images[D0] = filtered_image

    plt.figure(figsize=(18, 12))
    plt.suptitle(f'Butterworth Lowpass Filtering Results for {image_filename} (n={n_order})',
fontsize=16)
    plt.subplot(2, 3, 1)
    plt.imshow(original_image, cmap='gray', vmin=0, vmax=255)
    plt.title('a) Original Image')
    plt.axis('off')

    subplot_labels = ['b', 'c', 'd', 'e', 'f']
    for i, D0 in enumerate(D0_values):
        plt.subplot(2, 3, i + 2)
        plt.imshow(filtered_images[D0], cmap='gray', vmin=0, vmax=255)
        plt.title(f'{subplot_labels[i]}) BLPF (D0={D0}, n={n_order})')
        plt.axis('off')
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])

    combined_output_path = os.path.join(output_dir, f"{base_name}_blpf_results_combined_n{n_order}.png")
    plt.savefig(combined_output_path, bbox_inches='tight')
    plt.close()
```

## Processed Image

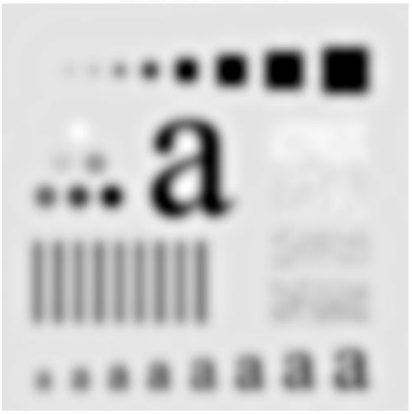
Butterworth Lowpass Filtering Results for Fig0440.tif (n=2.25)

a) Original Image



b) BLPF (D0=10, n=2.25)

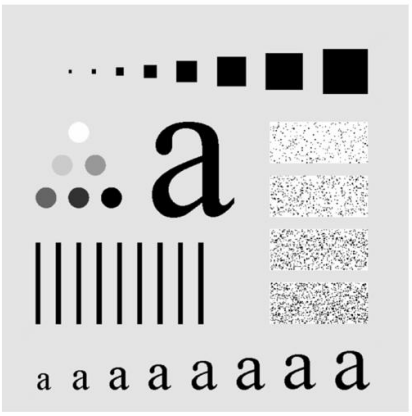
c) BLPF (D0=30, n=2.25)



d) BLPF (D0=60, n=2.25)

e) BLPF (D0=160, n=2.25)

f) BLPF (D0=460, n=2.25)



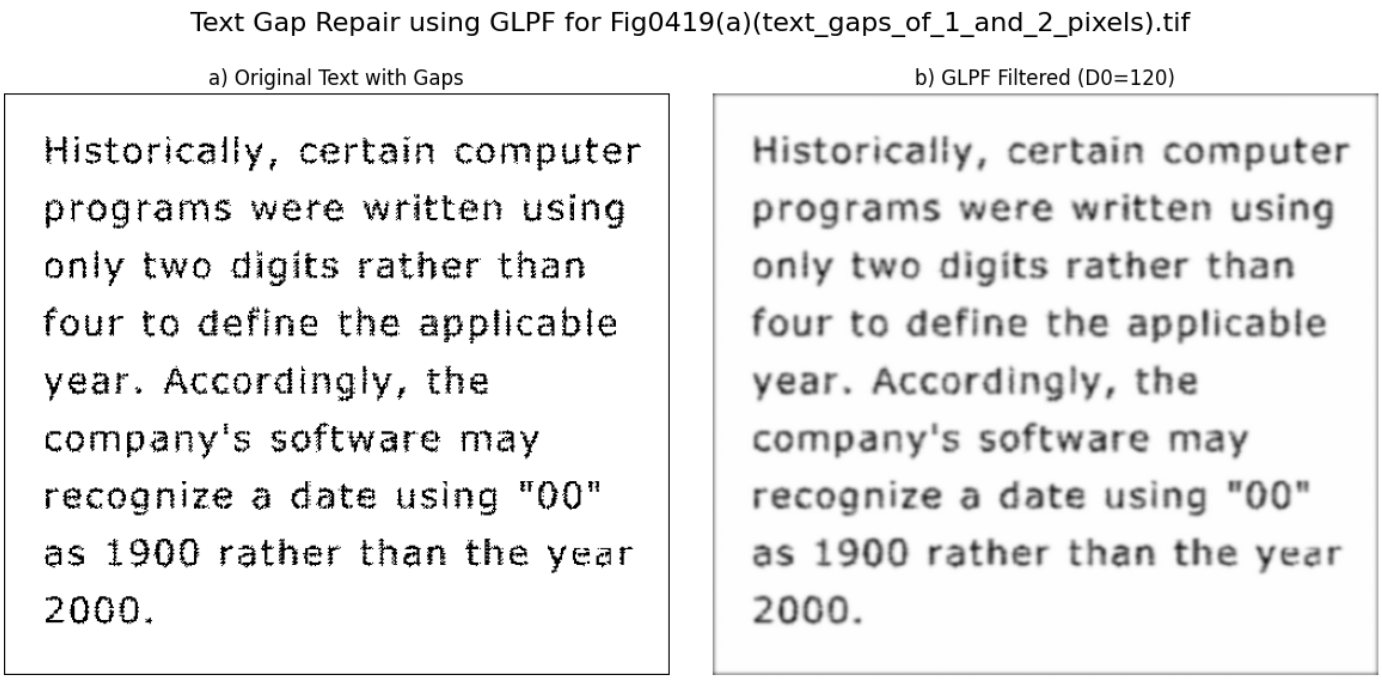
## Homework5: Implement Figure 4.48 by programming with Python

## Python Code:

```
import numpy as np
import imageio.v2 as imageio
import matplotlib.pyplot as plt
import os
def pad_image_for_dft(image, padding_mode='reflect'):
    """
    Pads the image to a size suitable for DFT (e.g., 2*M x 2*N)
    """
    M, N = image.shape
    P, Q = 2 * M, 2 * N
    pad_h = P - M
    pad_w = Q - N
    padded_image = np.pad(image, ((0, pad_h), (0, pad_w)), mode=padding_mode)
    return padded_image, (M, N)
def create_gaussian_lowpass_filter(shape, D0):
    """
    Creates a Gaussian Lowpass Filter (GLPF) in the frequency domain
    """
    P, Q = shape
    H = np.zeros((P, Q), dtype=float)
    center_u, center_v = P / 2, Q / 2
    u_coords = np.arange(P) - center_u
    v_coords = np.arange(Q) - center_v
    U, V = np.meshgrid(u_coords, v_coords, indexing='ij')
    D_uv_squared = U ** 2 + V ** 2

    if D0 == 0:
        H = np.ones(shape, dtype=float)
    else:
        H = np.exp(-D_uv_squared / (2 * D0 ** 2))

    return H

if __name__ == "__main__":
    input_dir = "original_image"
    output_dir = "output_image"
    os.makedirs(output_dir, exist_ok=True)
    image_filename = "Fig0419(a)(text_gaps_of_1_and_2_pixels).tif"
    image_path = os.path.join(input_dir, image_filename)
    base_name = os.path.splitext(image_filename)[0]
    original_image = imageio.imread(image_path)
    original_M, original_N = original_image.shape

    padded_image, _ = pad_image_for_dft(original_image, padding_mode='reflect')
    P, Q = padded_image.shape
    dft_original = np.fft.fft2(padded_image.astype(float))
```

```
centered_dft = np.fft.fftshift(dft_original)
D0_value = 120
H = create_gaussian_lowpass_filter((P, Q), D0_value)

filtered_dft = centered_dft * H
idft_shifted = np.fft.ifftshift(filtered_dft)
filtered_image_complex = np.fft.ifft2(idft_shifted)
filtered_image = np.real(filtered_image_complex)[0:original_M, 0:original_N]
filtered_image = np.clip(filtered_image, 0, 255).astype(np.uint8)
print(f"Finished GLPF with D0={D0_value}.")

plt.figure(figsize=(12, 6))
plt.suptitle(f'Text Gap Repair using GLPF for {image_filename}', fontsize=16)
plt.subplot(1, 2, 1)
plt.imshow(original_image, cmap='gray', vmin=0, vmax=255)
plt.title('a) Original Text with Gaps')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(filtered_image, cmap='gray', vmin=0, vmax=255)
plt.title(f'b) GLPF Filtered (D0={D0_value})')
plt.axis('off')
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
combined_output_path = os.path.join(output_dir, f"{base_name}_glpf_text_repair_results.png")
plt.savefig(combined_output_path, bbox_inches='tight')
plt.close()
```

**Processed Images:**



## Homework6: Implement Figure 4.49 by programming with Python

## Python Code:

```
import numpy as np
import imageio.v2 as imageio
import matplotlib.pyplot as plt
import os

def pad_image_for_dft(image, padding_mode='reflect'):
    """
    Pads the image to a size suitable for DFT (e.g., 2*M x 2*N)
    """
    M, N = image.shape
    P, Q = 2 * M, 2 * N
    pad_h = P - M
    pad_w = Q - N
    padded_image = np.pad(image, ((0, pad_h), (0, pad_w)), mode=padding_mode)
    return padded_image, (M, N)

def create_gaussian_lowpass_filter(shape, D0):
    """
    Creates a Gaussian Lowpass Filter (GLPF) in the frequency domain
    """
    P, Q = shape
    H = np.zeros((P, Q), dtype=float)
    center_u, center_v = P / 2, Q / 2
    u_coords = np.arange(P) - center_u
    v_coords = np.arange(Q) - center_v
    U, V = np.meshgrid(u_coords, v_coords, indexing='ij')
    D_uv_squared = U ** 2 + V ** 2
    if D0 == 0:
        H = np.ones(shape, dtype=float)
    else:
        H = np.exp(-D_uv_squared / (2 * D0 ** 2))
    return H

if __name__ == "__main__":
    input_dir = "original_image"
    output_dir = "output_image"
    os.makedirs(output_dir, exist_ok=True)exist
    image_filename = "Fig0450(a)(woman_original).tif"
    image_path = os.path.join(input_dir, image_filename)
    base_name = os.path.splitext(image_filename)[0]
    original_image = imageio.imread(image_path)
    original_M, original_N = original_image.shape

    padded_image, _ = pad_image_for_dft(original_image, padding_mode='reflect')
    P, Q = padded_image.shape
    dft_original = np.fft.fft2(padded_image.astype(float))
```

```
centered_dft = np.fft.fftshift(dft_original)
D0_values = [150, 130]
filtered_images = {}

for D0 in D0_values:
    H = create_gaussian_lowpass_filter((P, Q), D0)
    filtered_dft = centered_dft * H
    idft_shifted = np.fft.ifftshift(filtered_dft)
    filtered_image_complex = np.fft.ifft2(idft_shifted)
    filtered_image = np.real(filtered_image_complex)[0:original_M, 0:original_N]
    filtered_image = np.clip(filtered_image, 0, 255).astype(np.uint8)
    filtered_images[D0] = filtered_image

plt.figure(figsize=(18, 6))
plt.suptitle(f'Image Beautification using GLPF for {image_filename}', fontsize=16)
plt.subplot(1, 3, 1)
plt.imshow(original_image, cmap='gray', vmin=0, vmax=255)
plt.title('a) Original Image')
plt.axis('off')
plt.subplot(1, 3, 2)
plt.imshow(filtered_images[150], cmap='gray', vmin=0, vmax=255)
plt.title(f'b) GLPF Filtered (D0=150)')
plt.axis('off')
plt.subplot(1, 3, 3)
plt.imshow(filtered_images[130], cmap='gray', vmin=0, vmax=255)
plt.title(f'c) GLPF Filtered (D0=130)')
plt.axis('off')
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
combined_output_path = os.path.join(output_dir, f"{base_name}_glpf_beautification_results.png")
plt.savefig(combined_output_path, bbox_inches='tight')
print(f"Combined visualization saved to: {combined_output_path}")
plt.close()
```

## Processed Images:

## Homework7: Implement Figure 4.50 by programming with Python

## Python Code:

```
import numpy as np
import imageio.v2 as imageio
import matplotlib.pyplot as plt
import os


def pad_image_for_dft(image, padding_mode='reflect'):
    """Pads the image to a size suitable for DFT (e.g., 2*M x 2*N)"""
    M, N = image.shape
    P, Q = 2 * M, 2 * N
    pad_h = P - M
    pad_w = Q - N
    padded_image = np.pad(image, ((0, pad_h), (0, pad_w)), mode=padding_mode)
    return padded_image, (M, N)


def create_gaussian_lowpass_filter(shape, D0):
    """Creates a Gaussian Lowpass Filter (GLPF) in the frequency domain"""
    P, Q = shape
    H = np.zeros((P, Q), dtype=float)
    center_u, center_v = P / 2, Q / 2
    u_coords = np.arange(P) - center_u
    v_coords = np.arange(Q) - center_v
    U, V = np.meshgrid(u_coords, v_coords, indexing='ij')
    D_uv_squared = U ** 2 + V ** 2
    if D0 == 0:
        H = np.ones(shape, dtype=float)
    else:
        H = np.exp(-D_uv_squared / (2 * D0 ** 2))
    return H


if __name__ == "__main__":
    input_dir = "original_image"
    output_dir = "output_image"
    os.makedirs(output_dir, exist_ok=True)
    image_filename = "Fig0451(a)(satellite_original).tif"
    image_path = os.path.join(input_dir, image_filename)
    base_name = os.path.splitext(image_filename)[0]
    original_image = imageio.imread(image_path)
    original_M, original_N = original_image.shape

    padded_image, _ = pad_image_for_dft(original_image, padding_mode='reflect')
    P, Q = padded_image.shape
    dft_original = np.fft.fft2(padded_image.astype(float))
    centered_dft = np.fft.fftshift(dft_original)
```

```
D0_values = [50, 20]
filtered_images = {}

for D0 in D0_values:
    H = create_gaussian_lowpass_filter((P, Q), D0)
    filtered_dft = centered_dft * H
    idft_shifted = np.fft.ifftshift(filtered_dft)
    filtered_image_complex = np.fft.ifft2(idft_shifted)
    filtered_image = np.real(filtered_image_complex)[0:original_M, 0:original_N]
    filtered_image = np.clip(filtered_image, 0, 255).astype(np.uint8)
    filtered_images[D0] = filtered_image

plt.figure(figsize=(18, 6))
plt.suptitle(f'GLPF for Satellite Image (Fig 4.50) - {image_filename}', fontsize=16)

plt.subplot(1, 3, 1)
plt.imshow(original_image, cmap='gray', vmin=0, vmax=255)
plt.title('a) Original Satellite Image')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(filtered_images[50], cmap='gray', vmin=0, vmax=255)
plt.title(f'b) GLPF Filtered (D0=50)')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(filtered_images[20], cmap='gray', vmin=0, vmax=255)
plt.title(f'c) GLPF Filtered (D0=20)')
plt.axis('off')

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
combined_output_path = os.path.join(output_dir, f"{base_name}_glpf_satellite_results.png")
plt.savefig(combined_output_path, bbox_inches='tight')
plt.close()
```

## Processed Images: