DIP HW4 4109029034 蘇柏叡

Code link:
https://colab.research.google.com/drive/1bKqq6SFhWXPiMBJ0qGAFDKFEtO6qxrxi?usp=sharing
Q1、Q2 詳細說明因為有作圖，故在紙本作業上進行繪製。而 Q3-Q7 則會掃描成 pdf 檔案另外繳交。

Q1

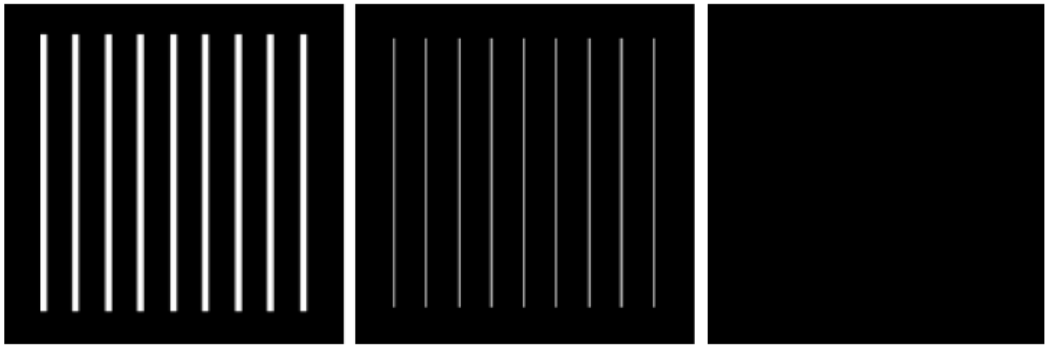


Figure P5.5

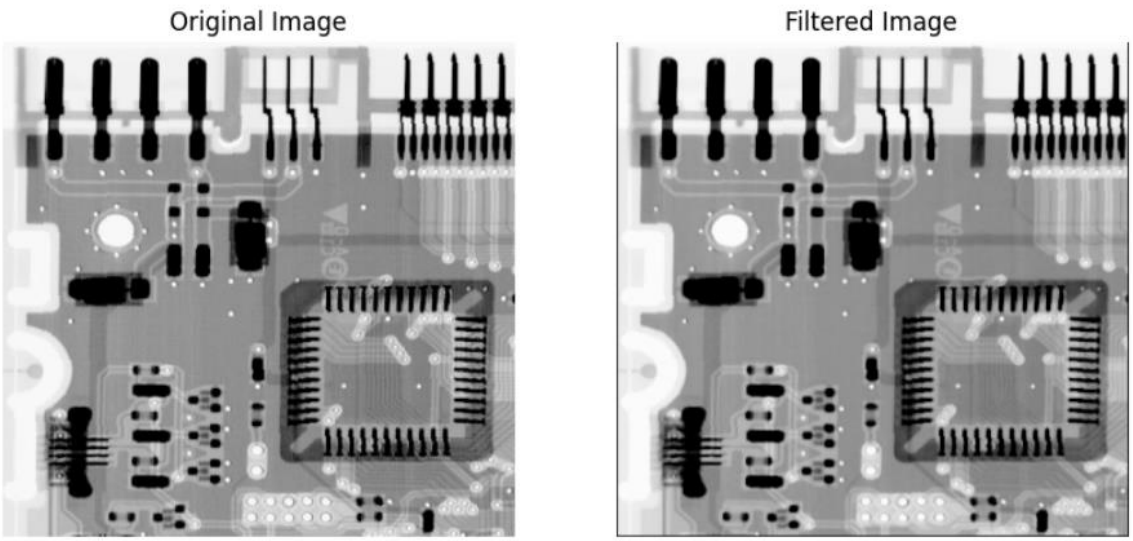
Q2

Figure P5.6

Q8-a

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

def median_filter(image):
    filtered_image = np.zeros_like(image)
    for i in range(1, image.shape[0] - 1):
        for j in range(1, image.shape[1] - 1):
            neighborhood = image[i - 1:i + 2, j - 1:j + 2]
            filtered_image[i, j] = np.median(neighborhood)
    return filtered_image

image = cv2.imread('/content/Fig0507(a)(ckt-board-orig).tif', cv2.IMREAD_GRAYSCALE)

# Apply median filtering
filtered_image = median_filter(image)

plt.figure(figsize=(10, 5))

# Original Image
plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')

# Filtered Image
plt.subplot(1, 2, 2)
plt.imshow(filtered_image, cmap='gray')
plt.title('Filtered Image')
plt.axis('off')
plt.show()
```

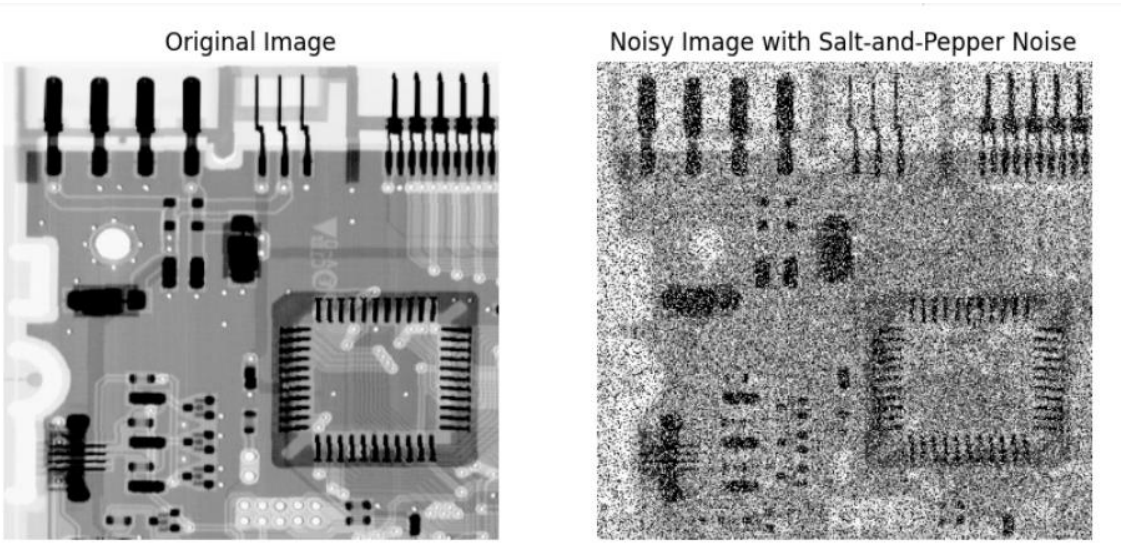
Result:



Q8-b

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

def add_salt_and_pepper_noise(image, pa, pb):
    noisy_image = np.copy(image)
    random_matrix = np.random.rand(image.shape[0], image.shape[1])
    noisy_image[random_matrix < pa] = 0
    noisy_image[random_matrix > 1 - pb] = 255
    return noisy_image

image = cv2.imread('/content/Fig0507(a)(ckt-board-orig).tif', cv2.IMREAD_GRAYSCALE)
# Add salt-and-pepper noise
pa = pb = 0.2
noisy_image = add_salt_and_pepper_noise(image, pa, pb)

# Display the original image and the noisy image
plt.figure(figsize=(10, 5))

# Original Image
plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')

# Noisy Image
plt.subplot(1, 2, 2)
plt.imshow(noisy_image, cmap='gray')
plt.title('Noisy Image with Salt-and-Pepper Noise')
plt.axis('off')

plt.show()
```

Result:

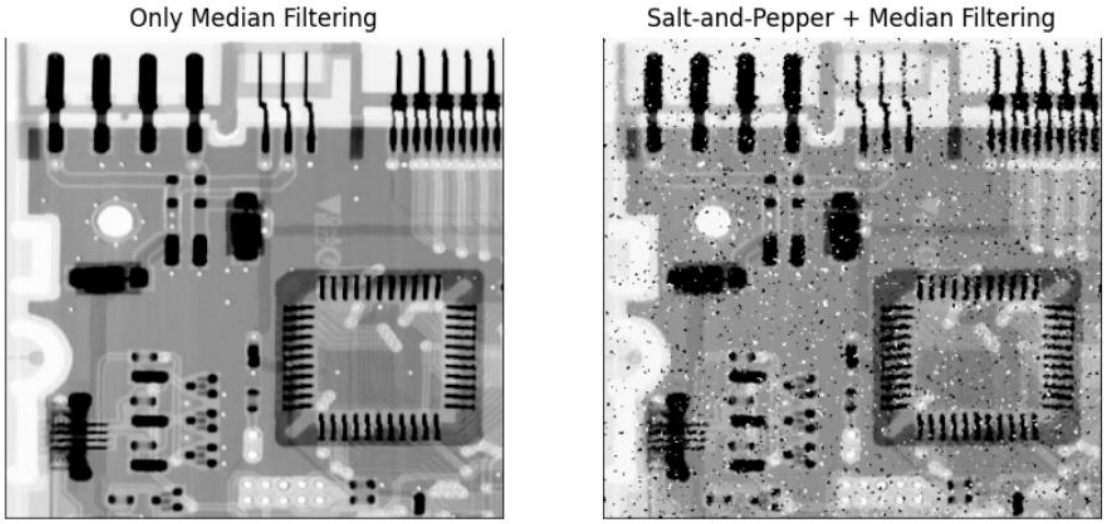


Q8-c

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

def add_salt_and_pepper_noise(image, pa, pb):
    noisy_image = np.copy(image)
    random_matrix = np.random.rand(image.shape[0], image.shape[1])
    noisy_image[random_matrix < pa] = 0
    noisy_image[random_matrix > 1 - pb] = 255
    return noisy_image

def median_filter(image):
    filtered_image = np.zeros_like(image)
    for i in range(1, image.shape[0] - 1):
        for j in range(1, image.shape[1] - 1):
            neighborhood = image[i - 1:i + 2, j - 1:j + 2]
            filtered_image[i, j] = np.median(neighborhood)
    return filtered_image

# Load the original image
image = cv2.imread('/content/Fig0507(a)(ckt-board-orig).tif', cv2.IMREAD_GRAYSCALE)

# Apply median filtering to the original image
filtered_image = median_filter(image)

# Add salt-and-pepper noise to the original image
pa = pb = 0.2
noisy_image = add_salt_and_pepper_noise(image, pa, pb)
```

```python
# Apply median filtering to the noisy image
filtered_image_noisy = median_filter(noisy_image)

# Display the results
plt.figure(figsize=(10, 5))

# Noisy Image after Median Filtering
plt.subplot(1, 2, 1)
plt.imshow(filtered_image, cmap='gray')
plt.title('Only Median Filtering')
plt.axis('off')

# Noisy Image after Adding Salt-and-Pepper Noise and Median Filtering
plt.subplot(1, 2, 2)
plt.imshow(filtered_image_noisy, cmap='gray')
plt.title('Salt-and-Pepper + Median Filtering')
plt.axis('off')

plt.show()
```

Result:



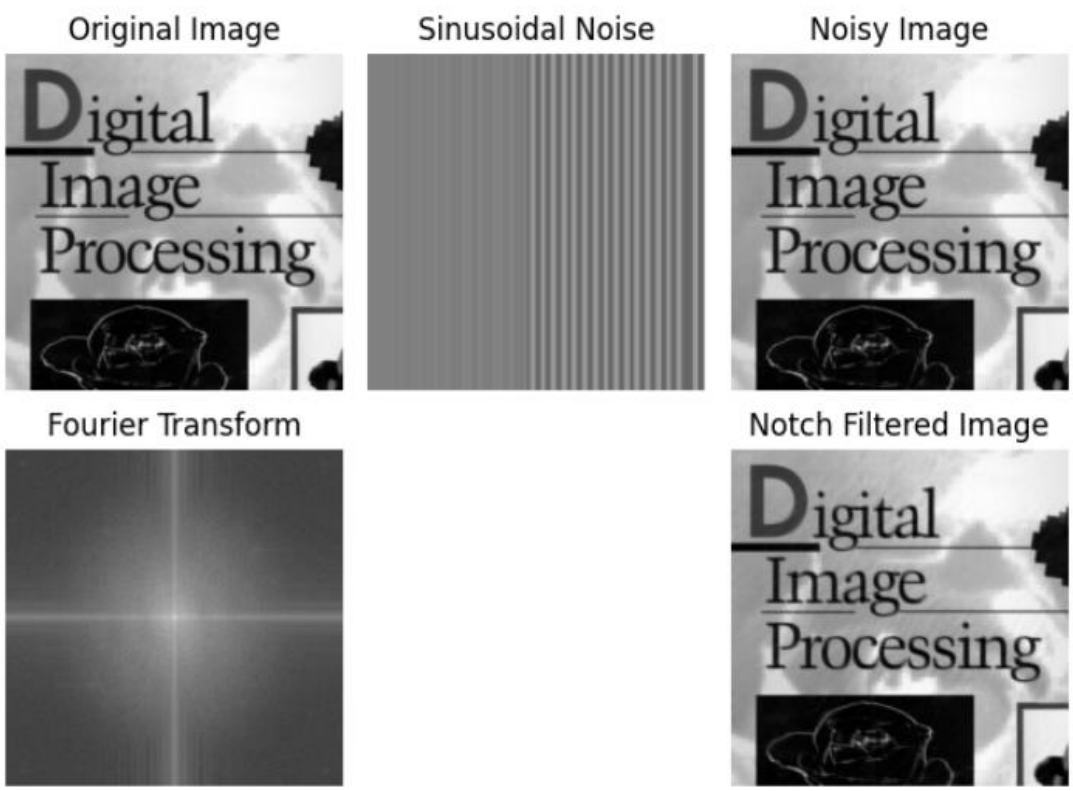
Q9 因本題連貫性，故不分小題

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

def generate_sinusoidal_noise(image_shape, A, u0, v0):
    rows, cols = image_shape
    x = np.arange(cols)
    y = np.arange(rows)
    X, Y = np.meshgrid(x, y)
    noise = A * np.sin(2 * np.pi * (u0 * X / cols + v0 * Y / rows))
    return noise

def add_sinusoidal_noise_to_image(image, A, u0, v0):
    noise = generate_sinusoidal_noise(image.shape[:2], A, u0, v0)
    noisy_image = np.clip(image.astype(np.float32) + noise, 0, 255).astype(np.uint8)
    return noisy_image

def lpfilter(filter_type, M, N, D0):
    u = np.arange(-M // 2, M // 2)
    v = np.arange(-N // 2, N // 2)
    U, V = np.meshgrid(u, v)
    D = np.sqrt(U**2 + V**2)

    if filter_type == 'gaussian':
        H = np.exp(-(D**2) / (2 * (D0**2)))
    else:
        H = np.ones((M, N))

    return H

```

```python
def notch_filter(image, notch_center, notch_radius):
    rows, cols = image.shape[:2]

    mask = np.ones((rows, cols), np.uint8)
    for center in notch_center:
        cx, cy = center
        for i in range(-notch_radius, notch_radius+1):
            for j in range(-notch_radius, notch_radius+1):
                if cx+i >= 0 and cx+i < rows and cy+j >= 0 and cy+j < cols:
                    mask[cx+i, cy+j] = 0

    f = np.fft.fft2(image)
    fshift = np.fft.fftshift(f)
    fshift_filtered = fshift * mask
    f_filtered = np.fft.ifftshift(fshift_filtered)
    filtered_image = np.fft.ifft2(f_filtered).real

    return np.clip(filtered_image, 0, 255).astype(np.uint8)

# Read image
img = cv2.imread('/content/Fig0526(a)(original_DIP).tif', cv2.IMREAD_GRAYSCALE)
M, N = img.shape

# Display original image
plt.subplot(2, 3, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')
plt.axis('off')

# Generate sinusoidal noise
A = 343
u0 = M // 2
v0 = 0
noise = generate_sinusoidal_noise(img.shape, A, u0, v0)

# Display noise
plt.subplot(2, 3, 2)
plt.imshow(noise, cmap='gray')
plt.title('Sinusoidal Noise')
plt.axis('off')

# Add noise to image
img_noisy = np.clip(img.astype(np.float32) + noise, 0, 255).astype(np.uint8)

# Display noisy image
plt.subplot(2, 3, 3)
plt.imshow(img_noisy, cmap='gray')
plt.title('Noisy Image')
plt.axis('off')

# Compute Fourier transform of noisy image
img_fft = np.fft.fft2(img_noisy)
```

```
# Display Fourier transform
plt.subplot(2, 3, 4)
plt.imshow(np.log(np.abs(np.fft.fftshift(img_fft)) + 1), cmap='gray')
plt.title('Fourier Transform')
plt.axis('off')

# Generate notch filter parameters
notch_center = [(256, 256)]    # Example: (256, 256)
notch_radius = 69    # Adjust radius as needed

# Apply notch filter
img_filtered = notch_filter(img_noisy, notch_center, notch_radius)

# Display filtered image
plt.subplot(2, 3, 6)
plt.imshow(img_filtered, cmap='gray')
plt.title('Notch Filtered Image')
plt.axis('off')

plt.tight_layout()
plt.show()
```

Result:



Q10 因本題連貫性，故不分小題

```python
import os
import numpy as np
from numpy.fft import fft2, ifft2
from scipy.signal import convolve2d
import matplotlib.pyplot as plt
import cv2

def blur(img, T=1):
    # Create an empty kernel
    kernel_size = 2 * T + 1
    kernel = np.zeros((kernel_size, kernel_size))

    if T == 1:
        # Set non-zero elements along the +45-degree direction
        for i in range(kernel_size):
            for j in range(kernel_size):
                if i == j:
                    kernel[i, j] = 1 / kernel_size
    else:
        # Generate the kernel according to Eq. (5.6-11)
        a = 0.1
        b = 0.1
        for u in range(-T, T+1):
            for v in range(-T, T+1):
                if u + v == T:
                    kernel[u+T, v+T] = np.abs(np.sin(np.pi * (a * u + b * v))) / (a * u + b * v)

    # Normalize the kernel
    kernel /= np.sum(kernel)

    # Convolve the image with the kernel
    return convolve2d(img, kernel, mode='same')
```

```python
def add_gaussian_noise(img, variance):
    sigma = np.sqrt(variance)
    gauss = np.random.normal(0, sigma, img.shape)
    noisy_img = img + gauss
    noisy_img[noisy_img < 0] = 0
    noisy_img[noisy_img > 255] = 255
    return noisy_img

def wiener_filter(img, kernel, K):
    kernel /= np.sum(kernel)
    dummy = np.copy(img)
    dummy = fft2(dummy)
    kernel = fft2(kernel, s=img.shape)
    kernel = np.conj(kernel) / (np.abs(kernel) ** 2 + K)
    dummy = dummy * kernel
    dummy = np.abs(ifft2(dummy))
    return dummy

def gaussian_kernel(kernel_size=3):
    h = np.outer(np.exp(-np.linspace(-1, 1, kernel_size)**2), np.exp(-np.linspace(-1, 1, kernel_size)**2))
    h /= np.sum(h)
    return h

def rgb2gray(rgb):
    return np.dot(rgb[..., :3], [0.2989, 0.5870, 0.1140])
```

```python
if __name__ == '__main__':
    # Load image and convert it to gray scale
    img = cv2.imread('Fig0526(a)(original_DIP).tif', cv2.IMREAD_GRAYSCALE)

    # Blur the image in the +45-degree direction with T=1
    blurred_img = blur(img, T=1)

    # Add Gaussian noise
    noisy_img = add_gaussian_noise(blurred_img, variance=10)

    # Apply Wiener Filter
    kernel = gaussian_kernel(3)
    filtered_img = wiener_filter(noisy_img, kernel, K=10)

    # Display results
    display = [img, blurred_img, noisy_img, filtered_img]
    label = ['Original Image', 'Blurred Image (T=1 and +45 degree)', 'Blurred + Gaussian Noise', 'Wiener Filtered']

    fig = plt.figure(figsize=(12, 10))

    for i in range(len(display)):
        fig.add_subplot(2, 2, i + 1)
        plt.imshow(display[i], cmap='gray')
        plt.title(label[i])

    plt.show()
```

Result: