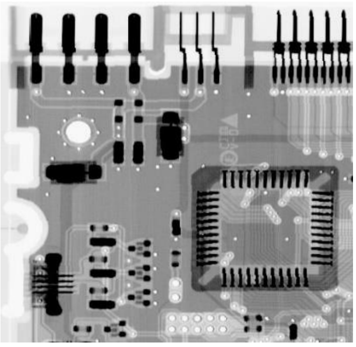
DIP homework 4

4109061012 陳柏翔

Problem 8：

```
# 4109061012 B.S.Chen
import cv2
import numpy as np
from tqdm import tqdm

IMAGE_PATH = "homework_4/src/Fig0507(a)(ckt-board-orig).tif"
TARGET_FOLDER = "homework_4/result"
PA, PB = 0.2, 0.2

def median_filtering(img: np.ndarray, size: int) -> np.ndarray:
    """ Problem (a): Median Filtering """
    s, pad = size, size // 2
    new_img = np.empty_like(img, dtype=np.float32)
    img = np.pad(img, ((pad, pad), (pad, pad)), "symmetric")

    for r in tqdm(range(new_img.shape[0])):
        for c in range(new_img.shape[1]):
            new_img[r, c] = np.median(img[r : r+s, c : c+s])
    return new_img

def salt_and_pepper_noise(img: np.ndarray, Pa: float, Pb: float) -> np.ndarray:
    """ Problem (b): Adding salt and pepper noise on image """
    noise = np.random.choice(3, img.shape, p=[1 - Pa - Pb, Pa, Pb])
    img[noise == 1] = 0
    img[noise == 2] = 255
    return img

if __name__ == "__main__":
    # Problem (c)
    img = np.asarray(cv2.imread(IMAGE_PATH, cv2.IMREAD_GRAYSCALE), dtype=np.uint8)
    img = salt_and_pepper_noise(img, PA, PB)
    cv2.imwrite(f"{TARGET_FOLDER}/P8_Image_with_noise.jpg", img.clip(0, 255).astype(np.uint8))
    img = median_filtering(img, 3)
    cv2.imwrite(f"{TARGET_FOLDER}/P8_Result.jpg", img.clip(0, 255).astype(np.uint8))
```



Original image

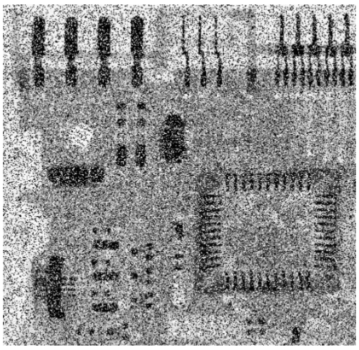
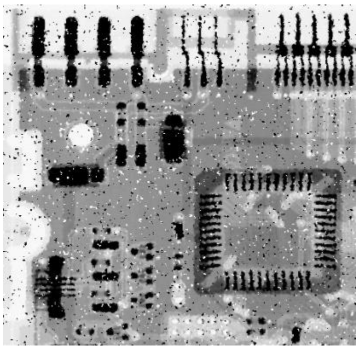


Image with noise

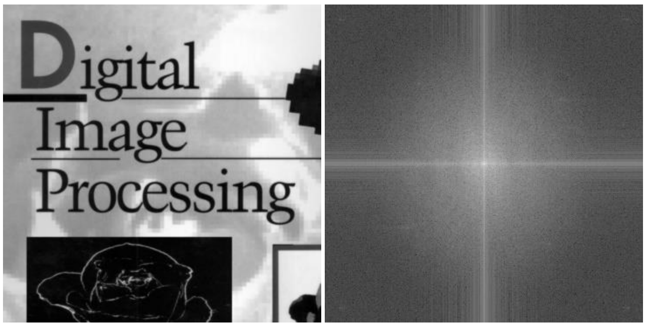


Filtered image

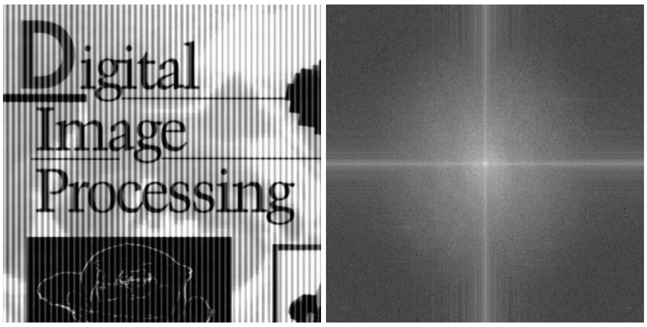
由於 Pa 與 Pb 數值皆較課本 Fig. 5.10 更高，
因此也造成較多的 Noise，看起來更加雜亂。

Problem 9：

```
# 4109061012 B.S.Chen
import cv2
import numpy as np

IMAGE_PATH = "homework_4/src/Fig0526(a)(original_DIP).tif"
TARGET_FOLDER = "homework_4/result"


def sinusoidal_noise(img: np.ndarray, A: float, u0: float, v0: float) -> np.ndarray:
    """ Problem (a): Adding sinusoidal noise (from prob5.14) on image """
    y, x = np.arange(img.shape[0])/img.shape[0], np.arange(img.shape[1])/img.shape[1]
    xv, yv = np.meshgrid(x, y)
    noise = A * np.sin(u0 * xv + v0 * yv)
    return img + noise


def horizontal_notch_filter(shape: tuple, D0: int, mid: int) -> np.ndarray:
    """ Horizontal, ideal notch reject filter """
    notch_pass = np.zeros(shape)
    notch_pass[shape[0]//2 - D0 : shape[0]//2 + D0 + 1, :] = 1
    notch_pass[:, shape[1]//2 - mid//2 : shape[1]//2 + mid//2 + 1] = 0
    display_ft_image(notch_pass, "P9_Notch_pass_filter.jpg")
    return 1 - notch_pass  # retrun is notch "reject" filter


def display_ft_image(ft_img: np.ndarray, name: str):
    save_img = np.log(1 + abs(ft_img))
    save_img = save_img / save_img.max() * 255
    cv2.imshow("Display Image", save_img.astype(np.uint8)), cv2.waitKey(0), cv2.destroyAllWindows()
    saving_image(save_img, name)


def saving_image(img: np.ndarray, name: str):
    cv2.imwrite(f"{TARGET_FOLDER}/{name}", np.clip(img, 0, 255).astype(np.uint8))


if __name__ == "__main__":
    # Problem (b)
    img = np.asarray(cv2.imread(IMAGE_PATH, cv2.IMREAD_GRAYSCALE), dtype=np.uint8)
    img = sinusoidal_noise(img, A=60, u0=img.shape[0]/2, v0=0)
    saving_image(img, "P9_Image_with_noise.jpg")

    # Problem (c)
    ft_img = np.fft.fft2(img)
    ft_img = np.fft.fftshift(ft_img)
    display_ft_image(ft_img, "P9_FT_spectrum.jpg")

    # Problem (d)
    notch_reject = horizontal_notch_filter(ft_img.shape, 0, ft_img.shape[0]//10)
    ft_img = ft_img * notch_reject
    ft_img = np.fft.ifftshift(ft_img)
    img = np.fft.ifft2(ft_img).real
    saving_image(img, "P9_Result.jpg")
```

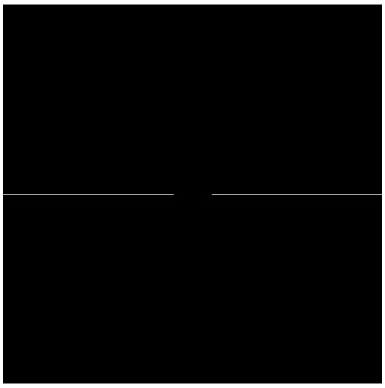


original image

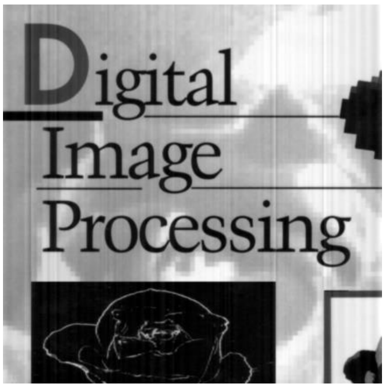


with sinusoidal noise

(橫軸上的中間兩邊有兩點特別亮，由 noise 所造成)



Notch pass filter



Result of notch reject filtering

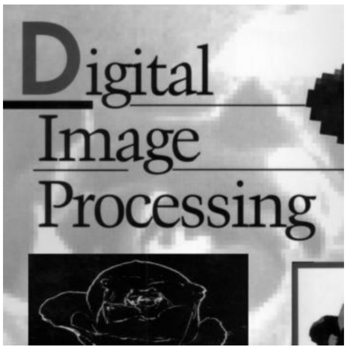
註：

課本 Fig. 5.19 的 noise 在圖片上的方向為橫向的，但這題的 noise 在 image 上造成的方向是縱向的。所以在 Fourier spectrum 上 Fig. 5.19 的 noise 出現在中央的縱軸上，而此題則是中央的橫軸上，所以我將 filter 的方向設為橫向的。

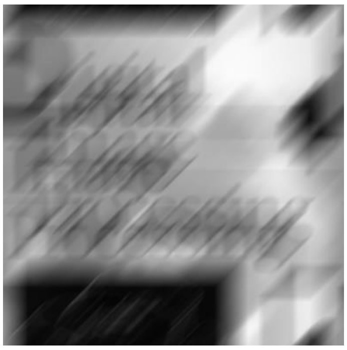
Problem 10：

```
# 4109061012 B.S.Chen
import cv2
import numpy as np

IMAGE_PATH = "homework_4/src/Fig0526(a)(original_DIP).tif"
TARGET_FOLDER = "homework_4/result"
K = 0.0003

def blurring_filter(shape: tuple, a: float, b: float, T: float) -> np.ndarray:
    """ Problem (a): Implement blurring filter as in Eq.(5.6-11) """
    y, x = np.arange(shape[0]), np.arange(shape[1])
    xv, yv = np.meshgrid(x, y)
    tmp = np.pi * (a * xv + b * yv)
    tmp[tmp == 0] = 0.0001

    mask = T * np.sin(tmp) * np.exp(-1j * tmp) / tmp
    return mask

def gaussian_noise(img: np.ndarray, mean: int, var: int) -> np.ndarray:
    """ Problem (a): Adding gaussian noise on the image """
    z_len = 512 - 1  # posible values: pos = 1~255, neg = -1~-255, center = 0, total: 511
    coef = np.sqrt(2 * np.pi * var * np.ones(z_len))
    hist = coef * np.exp(-np.square(np.arange(z_len) - 255 - mean) / (2 * var))

    mask = np.random.choice(z_len, img.shape, p=hist/hist.sum()) - 255
    img = img + mask
    return img.clip(0, 255)

def Wiener_filtering(G: np.ndarray, H: np.ndarray, K: float) -> np.ndarray:
    H_square = H * H.conj()
    F_hat = (G / H) * (H_square / (H_square + K))
    return F_hat

def display_ft_image(ft_img: np.ndarray, name: str):
    """ Display (saving) the image which is in fourier spectrum """
    save_img = np.log(1 + abs(ft_img))
    save_img = save_img / save_img.max() * 255
    cv2.imwrite(f"{TARGET_FOLDER}/{name}", save_img)
```

```
if __name__ == "__main__":
    # Problem (b): Blurring
    img = np.asarray(cv2.imread(IMAGE_PATH, cv2.IMREAD_GRAYSCALE), dtype=np.uint8)
    blur_filter = blurring_filter(img.shape, -0.1, 0.1, 1)
    display_ft_image(blur_filter, "P10_Blurring_filter.jpg")

    ft_img = np.fft.fft2(img)
    ft_img = np.fft.fftshift(ft_img)
    ft_img = ft_img * blur_filter
    ft_img = np.fft.ifftshift(ft_img)
    img = np.fft.ifft2(ft_img).real

    img = img.clip(0, 255)
    cv2.imwrite(f"{TARGET_FOLDER}/P10(b)_Result.jpg", img.astype(np.uint8))


    # Problem (c): Adding noise
    img = gaussian_noise(img, 0, 10)
    cv2.imwrite(f"{TARGET_FOLDER}/P10(c)_Result.jpg", img.astype(np.uint8))

    # Problem (d): Restoring the image
    ft_img = np.fft.fft2(img)
    ft_img = np.fft.fftshift(ft_img)
    ft_img = Wiener_filtering(ft_img, blur_filter, K)
    ft_img = np.fft.ifftshift(ft_img)
    img = np.fft.ifft2(ft_img).real
    cv2.imwrite(f"{TARGET_FOLDER}/P10(d)_Result.jpg", img.clip(0, 255).astype(np.uint8))
```

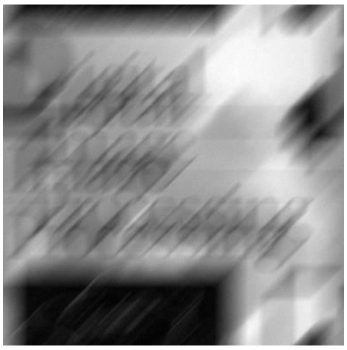


original

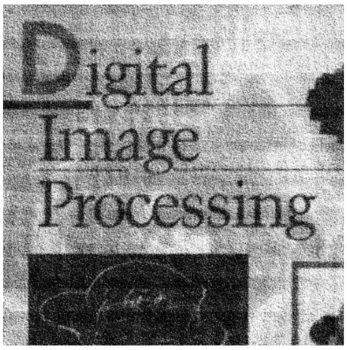


blurred image

(+45-degree direction)



blurred image with noise



filtered image