ข้อ 1

Notch filters

def notch\_filter(rows, cols, r, filter\_type="low-pass"):

    center\_x, center\_y = rows // 2, cols // 2

    filter = np.zeros((rows, cols), np.float32)

    for x in range(rows):

        for y in range(cols):

            distance = np.sqrt((x - center\_x)\*\*2 + (y - center\_y)\*\*2)

            if filter\_type == "low-pass":

                if distance <= r:

                    filter[x, y] = 1

            elif filter\_type == "high-pass":

                if distance > r:

                    filter[x, y] = 1

    return filter

images = ["flower1.jpg", "fruit.jpg"]

for each in images:

    img = cv2.imread(each, cv2.IMREAD\_GRAYSCALE)

    # Discrete Fourier Transform

    dft = cv2.dft(np.float32(img), flags=cv2.DFT\_COMPLEX\_OUTPUT)

    dft\_shifted = np.fft.fftshift(dft)

    # Create the filter masks

    r\_values = [10, 50, 100]

    for r in r\_values:

        low\_pass\_filter = notch\_filter(img.shape[0], img.shape[1], r, "low-pass")

        high\_pass\_filter = notch\_filter(img.shape[0], img.shape[1], r, "high-pass")

        # Apply the filters

        low\_pass = dft\_shifted \* low\_pass\_filter[:, :, np.newaxis]

        high\_pass = dft\_shifted \* high\_pass\_filter[:, :, np.newaxis]

    # Inverse DFT for Low-Pass filtered image

        idft\_low\_pass = np.fft.ifftshift(low\_pass)

        img\_reconstructed\_low = cv2.idft(idft\_low\_pass, flags=cv2.DFT\_REAL\_OUTPUT)

        img\_reconstructed\_low = cv2.normalize(img\_reconstructed\_low, None, 0, 255, cv2.NORM\_MINMAX).astype(np.uint8)

        # Inverse DFT for High-Pass filtered image

        idft\_high\_pass = np.fft.ifftshift(high\_pass)

        img\_reconstructed\_high = cv2.idft(idft\_high\_pass, flags=cv2.DFT\_REAL\_OUTPUT)

        img\_reconstructed\_high = cv2.normalize(img\_reconstructed\_high, None, 0, 255, cv2.NORM\_MINMAX).astype(np.uint8)

        output\_folder = "./notch\_filters"

        if not os.path.exists(output\_folder):

            os.makedirs(output\_folder)

        cv2.imwrite(os.path.join(output\_folder, f"{each} low pass r = {r}.jpg"), img\_reconstructed\_low)

        cv2.imwrite(os.path.join(output\_folder, f"{each} high pass r = {r}.jpg"), img\_reconstructed\_high)

ภาพต้นฉบับ

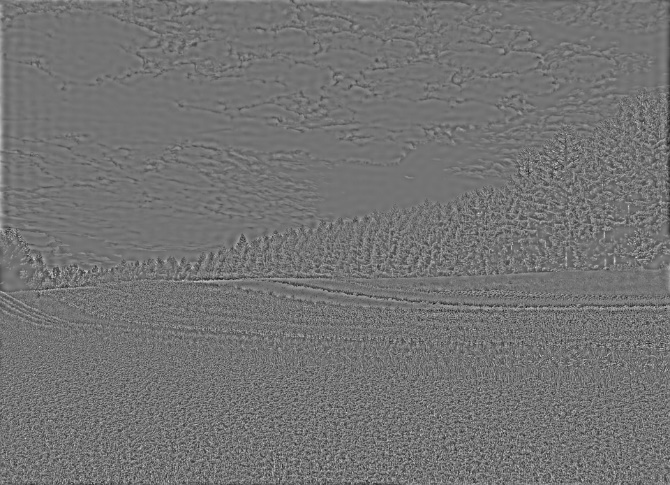
flower1.jpg fruit.jpg



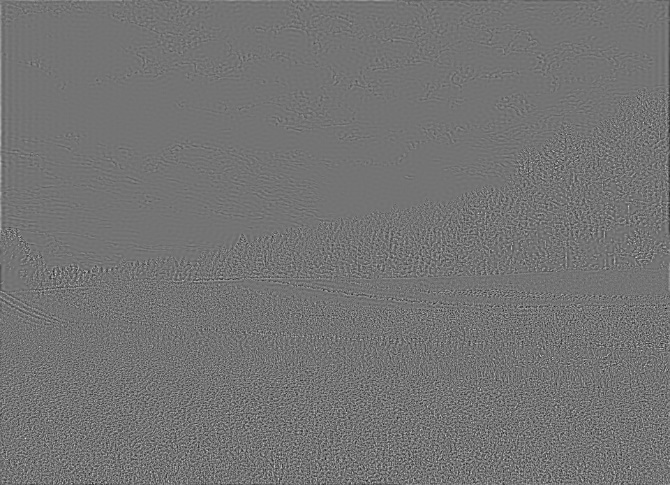
high pass filter low pass filter



r = 10

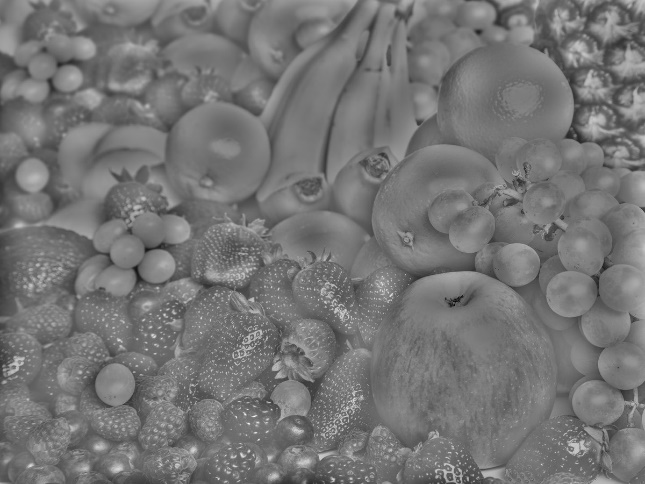


r = 50

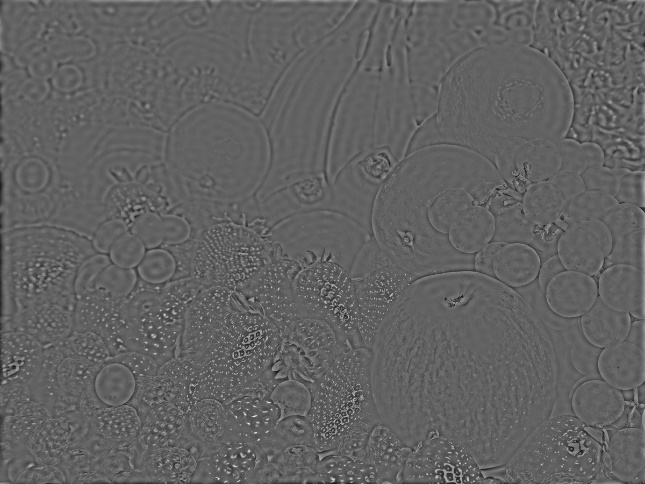


r = 100

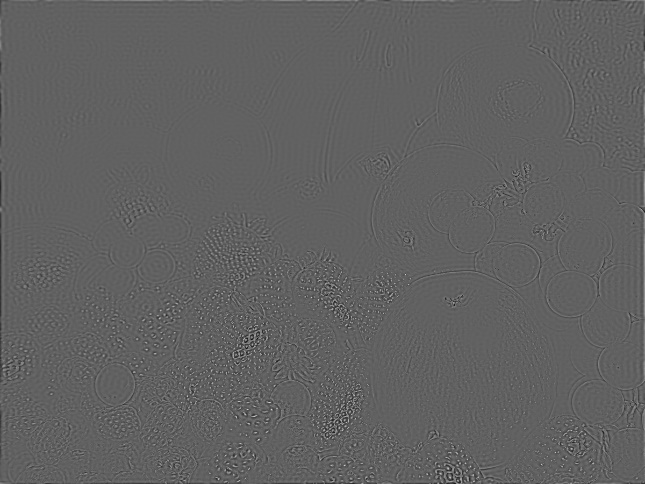
high pass filter low pass filter



r = 10



r = 50



r = 100

อธิบายเพิ่มเติม : การทำ notch filters เป็นการ filter ค่าจากความถึ่ โดย low-pass filter จะทำการ filter ค่าความถึ่ที่มากกว่า r ออกและเก็บค่าที่อยู่ในรัศมี r ไว้ ส่วน high-pass filter จะตรงกันข้ามคือทำการ filter ค่าความถี่ที่น้อยกว่า r ออก และเก็บค่าที่อยู่นอกรัศมี r ไว้ ซึ่งการทำอย่างนี้ส่งผลให้ภาพจาก low-pass filter จะมีลักษณะที่เบลอเพราะส่วนที่ไม่มีการเปลี่ยนแปลง (ส่วนที่ไม่ใช่ขอบ) จะมีค่าความถึ่ที่ต่ำและ low-pass filter เก็บไว้ ส่วนภาพที่ได้จาก high-pass filter จะมีลักษณะเน้นขอบในรูปภาพ เพราะ ส่วนที่มีการเปลี่ยนแปลง (ส่วนที่เป็นขอบๆ) ความถี่จะสูงแล้ว high-pass filter เก็บไว้

ข้อ 2

Gaussian filters

def gaussian\_lpf(rows, cols, D0):

    center\_x, center\_y = rows // 2, cols // 2

    filter = np.zeros((rows, cols, 2), np.float32)

    for x in range(rows):

        for y in range(cols):

            distance = np.sqrt((x - center\_x) \*\* 2 + (y - center\_y) \*\* 2)

            filter[x, y] = np.exp(-(distance \*\* 2) / (2 \* (D0 \*\* 2)))

    return filter

def gaussian\_hpf(rows, cols, D0):

    return 1 - gaussian\_lpf(rows, cols, D0)

images = ["flower1.jpg", "fruit.jpg"]

for each in images:

    # Load the image in grayscale

    img = cv2.imread(each, cv2.IMREAD\_GRAYSCALE)

    # Apply FFT and shift the zero-frequency component to the center

    dft = cv2.dft(np.float32(img), flags=cv2.DFT\_COMPLEX\_OUTPUT)

    dft\_shifted = np.fft.fftshift(dft)

    # Apply Gaussian Filters

    D0\_values = [10, 50, 100]

    for D0 in D0\_values:

        # Low-pass filter

        lpf = gaussian\_lpf(img.shape[0], img.shape[1], D0)

        filtered\_lpf = dft\_shifted \* lpf

        idft\_lpf = np.fft.ifftshift(filtered\_lpf)

        img\_reconstructed\_lpf = cv2.idft(idft\_lpf, flags=cv2.DFT\_REAL\_OUTPUT)

        img\_display\_lpf = cv2.normalize(img\_reconstructed\_lpf, None, 0, 255, cv2.NORM\_MINMAX).astype(np.uint8)

        # High-pass filter

        hpf = gaussian\_hpf(img.shape[0], img.shape[1], D0)

        filtered\_hpf = dft\_shifted \* hpf

        idft\_hpf = np.fft.ifftshift(filtered\_hpf)

        img\_reconstructed\_hpf = cv2.idft(idft\_hpf, flags=cv2.DFT\_REAL\_OUTPUT)

        img\_display\_hpf = cv2.normalize(img\_reconstructed\_hpf, None, 0, 255, cv2.NORM\_MINMAX).astype(np.uint8)

    #     cv2.imshow(f'Gaussian HPF D0={D0}', img\_display\_hpf)

        output\_folder = "./gaussian\_filters"

        if not os.path.exists(output\_folder):

            os.makedirs(output\_folder)

        cv2.imwrite(os.path.join(output\_folder, f"{each} low pass d0 = {D0}.jpg"), img\_display\_lpf)

        cv2.imwrite(os.path.join(output\_folder, f"{each} high pass d0 = {D0}.jpg"), img\_display\_hpf)

ภาพต้นฉบับ

flower1.jpg fruit.jpg

A field with trees in the background

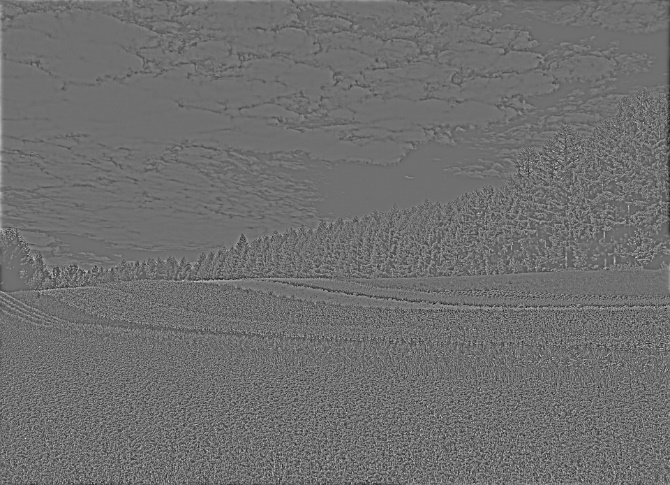
Description automatically generatedA close-up of various fruits

Description automatically generated

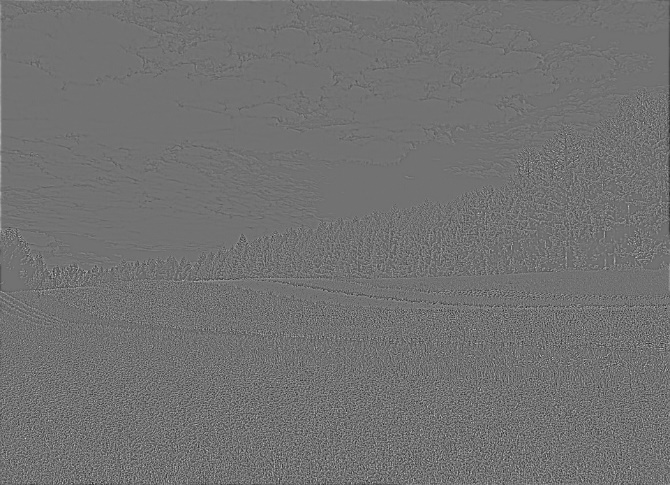
high pass filter low pass filter



r = 10



r = 50



r = 100

อธิบายเพิ่มเติม : การทำ gaussian filter จะคล้ายๆ notch filter โดย low-pass filter จะทำการ filter ค่าความถึ่ที่มากกว่า r ออกและเก็บค่าที่อยู่ในรัศมี r ไว้ ส่วน high-pass filter จะตรงกันข้ามคือทำการ filter ค่าความถี่ที่น้อยกว่า r ออก และเก็บค่าที่อยู่นอกรัศมี r ไว้ แต่จะแตกต่างกันตรงที่ gaussian filter จะไม่ทำการ filter แบบสีดำทันที แต่จะค่อยเปลี่ยนเป็นสีดำ หรือ ค่อยๆเปลี่ยนเป็นสีขาว (low-pass กับ high-pass ตามลำดับ) แต่คิดว่าที่ภาพ high-pass r = 50 และ r = 100 มองเห็นได้ไม่ชัดเพราะ r มีขนาดที่ใหญ่เกินไปทำให้ภาพใน frequency domain ส่วนใหญ่ถูกตัดทิ้งออกไป

3. Remove periodic noise from the given images using any of the filters that you think are the most suitable ones to be used.

noisy\_flower1\_horizontal.jpg

import os

import cv2

import numpy as np

import matplotlib.pyplot as plt

def gaussian\_notch\_reject(shape, d0=30, width=10):

    rows, cols = shape

    crow, ccol = rows // 2, cols // 2

    y, x = np.ogrid[:rows, :cols]

    d2\_from\_center = (x - ccol)\*\*2 + (y - crow)\*\*2

    d2\_from\_d0\_above = (x - ccol)\*\*2 + (y - (crow-d0))\*\*2

    d2\_from\_d0\_below = (x - ccol)\*\*2 + (y - (crow+d0))\*\*2

    # Gaussian masks for the two dots

    mask\_above = np.exp(-d2\_from\_d0\_above / (2\*width\*\*2))

    mask\_below = np.exp(-d2\_from\_d0\_below / (2\*width\*\*2))

    # Final combined mask (1 - mask) to reject the specific frequencies

    mask\_gaussian = 1 - (mask\_above + mask\_below)

    # Extend the shape for real and imaginary parts

    mask\_gaussian = mask\_gaussian[:, :, np.newaxis]

    return mask\_gaussian

img = cv2.imread('noisy\_flower1\_horizontal.jpg', cv2.IMREAD\_GRAYSCALE)

dft = cv2.dft(np.float32(img), flags=cv2.DFT\_COMPLEX\_OUTPUT)

dft\_shifted = np.fft.fftshift(dft)

notch70 = gaussian\_notch\_reject(img.shape, 70, 25)

notch140 = gaussian\_notch\_reject(img.shape, 140, 25)

notch = notch70 \* notch140

# Apply the notch filter

fshift = dft\_shifted \* notch

# Inverse FFT to get the image back

f\_ishift = np.fft.ifftshift(fshift)

img\_back = cv2.idft(f\_ishift, flags=cv2.DFT\_REAL\_OUTPUT)

img\_back = cv2.normalize(img\_back, None, 0, 255, cv2.NORM\_MINMAX).astype(np.uint8)

magnitude\_spectrum = 20 \* np.log(cv2.magnitude(dft\_shifted[:, :, 0], dft\_shifted[:, :, 1]))

magnitude\_spectrum\_normalized = cv2.normalize(magnitude\_spectrum, None, 0, 255, cv2.NORM\_MINMAX)

# Compute the magnitude of the filtered frequency domain

magnitude\_values\_filtered = cv2.magnitude(fshift[:, :, 0], fshift[:, :, 1])

magnitude\_spectrum\_filtered = 20 \* np.log(magnitude\_values\_filtered + 1)

magnitude\_spectrum\_filtered\_normalized = cv2.normalize(magnitude\_spectrum\_filtered, None, 0, 255, cv2.NORM\_MINMAX)

cv2.imshow('Original Magnitude Spectrum', magnitude\_spectrum\_normalized.astype(np.uint8))

cv2.imshow('Filtered Magnitude Spectrum', magnitude\_spectrum\_filtered\_normalized.astype(np.uint8))

output\_folder = "./noisy"

if not os.path.exists(output\_folder):

    os.makedirs(output\_folder)

cv2.imwrite(os.path.join(output\_folder, f"horizontal image d0 = 95,190 width = 25.jpg"), img\_back)

cv2.imwrite(os.path.join(output\_folder, f"horizontal magnitude\_spectrum.jpg"), magnitude\_spectrum\_normalized)

cv2.imwrite(os.path.join(output\_folder, f"horizontal magnitude\_spectrum\_filtered d0 = 95,190 width = 25.jpg"), magnitude\_spectrum\_filtered\_normalized)

noisy\_flower1\_vertical.jpg

import os

import cv2

import numpy as np

import matplotlib.pyplot as plt

def gaussian\_notch\_reject(shape, d0=30, width=10):

    rows, cols = shape

    crow, ccol = rows // 2, cols // 2

    y, x = np.ogrid[:rows, :cols]

    d2\_from\_center = (x - ccol)\*\*2 + (y - crow)\*\*2

    d2\_from\_d0\_right = (x - (ccol-d0))\*\*2 + (y - (crow))\*\*2

    d2\_from\_d0\_left = (x - (ccol+d0))\*\*2 + (y - (crow))\*\*2

    # Gaussian masks for the two dots

    mask\_above = np.exp(-d2\_from\_d0\_right / (2\*width\*\*2))

    mask\_below = np.exp(-d2\_from\_d0\_left / (2\*width\*\*2))

    # Final combined mask (1 - mask) to reject the specific frequencies

    mask\_gaussian = 1 - (mask\_above + mask\_below)

    # Extend the shape for real and imaginary parts

    mask\_gaussian = mask\_gaussian[:, :, np.newaxis]

    return mask\_gaussian

img = cv2.imread('noisy\_flower1\_vertical.jpg', cv2.IMREAD\_GRAYSCALE)

dft = cv2.dft(np.float32(img), flags=cv2.DFT\_COMPLEX\_OUTPUT)

dft\_shifted = np.fft.fftshift(dft)

notch1 = gaussian\_notch\_reject(img.shape,95, 25)

notch2 = gaussian\_notch\_reject(img.shape, 190, 25)

notch = notch1 \* notch2

# Apply the notch filter

fshift = dft\_shifted \* notch

# Inverse FFT to get the image back

f\_ishift = np.fft.ifftshift(fshift)

img\_back = cv2.idft(f\_ishift, flags=cv2.DFT\_REAL\_OUTPUT)

img\_back = cv2.normalize(img\_back, None, 0, 255, cv2.NORM\_MINMAX).astype(np.uint8)

magnitude\_spectrum = 20 \* np.log(cv2.magnitude(dft\_shifted[:, :, 0], dft\_shifted[:, :, 1]))

magnitude\_spectrum\_normalized = cv2.normalize(magnitude\_spectrum, None, 0, 255, cv2.NORM\_MINMAX)

# Compute the magnitude of the filtered frequency domain

magnitude\_values\_filtered = cv2.magnitude(fshift[:, :, 0], fshift[:, :, 1])

magnitude\_spectrum\_filtered = 20 \* np.log(magnitude\_values\_filtered + 1)

magnitude\_spectrum\_filtered\_normalized = cv2.normalize(magnitude\_spectrum\_filtered, None, 0, 255, cv2.NORM\_MINMAX)

output\_folder = "./noisy"

if not os.path.exists(output\_folder):

    os.makedirs(output\_folder)

cv2.imwrite(os.path.join(output\_folder, f"vertical image d0 = 95,190 width = 25.jpg"), img\_back)

cv2.imwrite(os.path.join(output\_folder, f"vertical magnitude\_spectrum.jpg"), magnitude\_spectrum\_normalized)

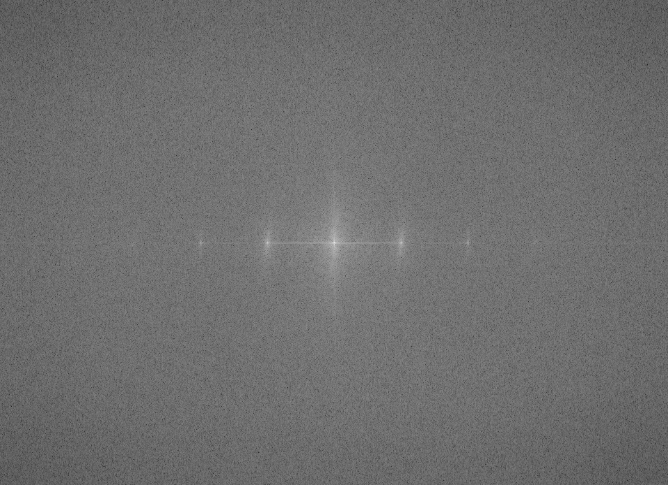
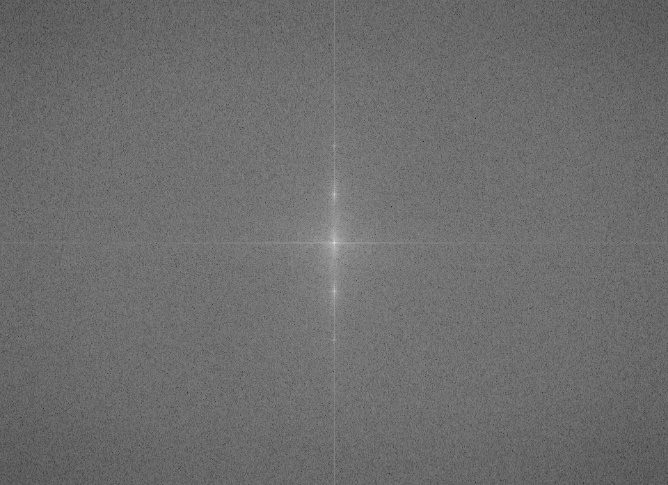
cv2.imwrite(os.path.join(output\_folder, f"vertical magnitude\_spectrum\_filtered d0 = 95,190 width = 25.jpg"), magnitude\_spectrum\_filtered\_normalized)

ภาพต้นฉบับ

noisy\_flower1\_horizontal.jpg noisy\_flower1\_vertical.jpg



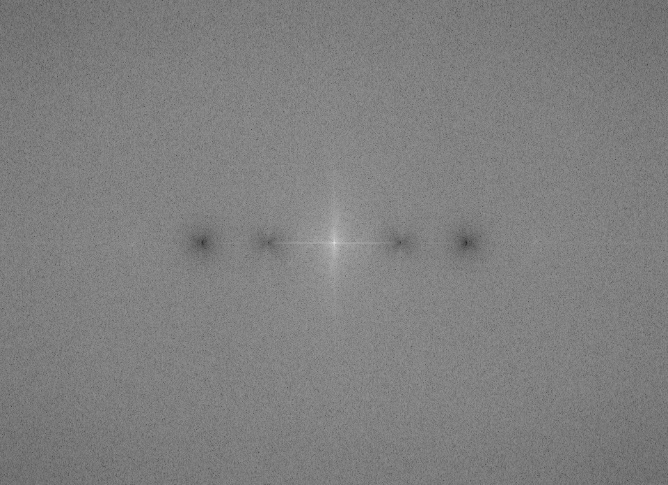
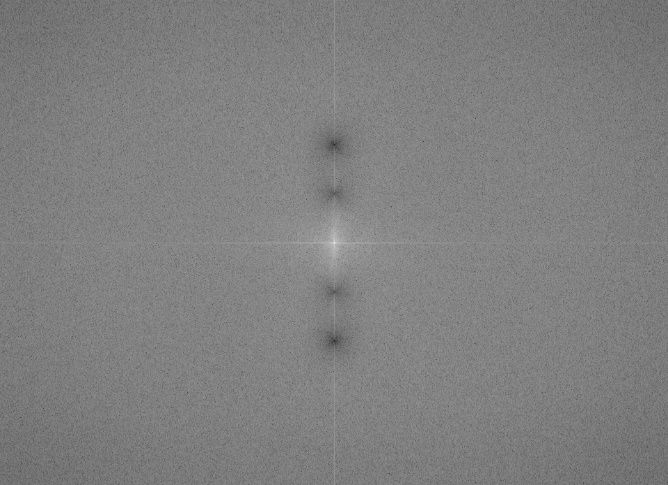
Horizontal magnitude spectrum Vertical magnitude spectrum



Gaussian filter horizontal d0 = 70, 140 width = 25 Gaussian filter vertical d0 = 95, 190 width = 25



Filtered horizontal magnitude spectrum Filtered vertical magnitude spectrum



อธิบายเพิ่มเติม : ภาพที่กำหนดมามี noise ที่มีลักษณะเป็นเส้นในแนวนอนและแนวตั้งที่ห่างด้วยความยาวเท่าๆกัน ทำให้การใช้ notch filter มีความเหมาะสมมากในการลบ noise ประเภทนี้ และเหตุผลที่ใช้ gaussian เนื่องจากลักษณะจุด noise บน magnitude spectrum มีลักษณะเป็นวงกลม ซึ่งหลังจากลบ noise ออกจะทำให้ภาพมีลักษณะที่ชัดเจนมากยิ่งขึ้นเส้น noise มองเห็นได้ยากขึ้น แต่อาจมีหลงเหลือบางจากการใช้ notch filter ที่ลบออกไม่หมด