|  |  |  |  |
| --- | --- | --- | --- |
| Optimized_image_813b5ec2.png | | **دانشگاه تهران**  **پردیس دانشکده­های فنی**  **دانشکده مهندسی برق و کامپیوتر** | Images__Logo_FE.gif |
|  | | | |
|  | | | |
| پردازش تصاویر دیجیتال  **گزارش تمرین سری چهارم-فصل 5** | | | |
| **دانشجو**  سید محمد جواد موسوی | | | |
| **استاد کلاس**  دکتر حمید سلطانیان­زاده | | | |

سوال 2

بخش اول

در این بخش ابتدا با استفاده از کتابخانه cv2 و از طریق کد زیر سه فیلتر میانگین گیری و میانه و گوسی بر روی تصویر اعمال شده است:

1. # Apply noise reduction filters (Mean, Median, and Gaussian)

2. # Mean filter

3. mean = cv2.blur(img, ksize=(3, 3))

4.

5. # Median filter

6. median = cv2.medianBlur(img, ksize=3)

7.

8. # Gaussian filter

9. gaussian = cv2.GaussianBlur(img, ksize=(3, 3), sigmaX=0)

سایز کرنل در هر سه فیلتر برابر 3 در 3 در نظر گرفته شد. همچنینی در فیلتر گوسی مقدار واریانس را بر اساس سایز کرنل به صورت خودکار تنظیم شده است(sigmaX=0). خروجی سه فیلتر به همراه تصویر اصلی در زیر آورده شده است:

فیلتر میانگین



فیلتر میانه



فیلتر گوسی



از لحاظ ظاهری خروجی مربوط به فیلتر میانه بهتر است زیرا نقاط سفید که به صورت نویز نمکی در تصویر اصلی قابل درک هستند را بهتر فیلتر کرده است ولی برای بررسی تاثیر اعمال فیلترینگ لازم است تا دو فاکتور سیگنال به نویز و شباهت ساختاری و ادراکی تصاویر قبل و بعد از فیلترینگ مقایسه بشود. به این منظور از دو معیار PSVR و SSIM استفاده شده است:

1. # Evaluation of Method Effectiveness

2. from skimage.metrics import peak\_signal\_noise\_ratio as psnr, structural\_similarity as ssim

3.

4. print("Mean PSNR:", psnr(img, mean))

5. print("Median PSNR:", psnr(img, median))

6. print("Gaussian PSNR:", psnr(img, gaussian))

7.

8. print("Mean SSIM:", ssim(img, mean))

9. print("Median SSIM:", ssim(img, median))

10. print("Gaussian SSIM:", ssim(img, gaussian))

نتایج به قرار زیر است:

Mean PSNR: 17.78466642374896

Median PSNR: 17.49756737817909

Gaussian PSNR: 19.243479553732016

Mean SSIM: 0.3172768311235811

Median SSIM: 0.25104305171963

Gaussian SSIM: 0.5247483978239877

بر اساس معیارهای PSNR و SSIM، فیلتر گوسی به‌عنوان مؤثرترین روش برای کاهش نویز در تصویر ماهواره‌ای شناخته شد. این فیلتر موفق به کسب PSNR برابر با ۱۹.۲۴ و SSIM برابر با ۰.۵۲۵ شد و عملکرد بهتری نسبت به فیلترهای میانگین و میانه داشت. این موضوع نشان می‌دهد که فیلتر گوسی نه‌تنها نویز را به‌طور مؤثرتری کاهش داده، بلکه جزئیات ساختاری تصویر را نیز بهتر حفظ کرده است که این امر در کاربردهای سنجش از دور بسیار حائز اهمیت است.

بخش دوم

در این بخش برای بررسی تاثیر فیلتر بر نرم کردن لبه ها از الگوریتم canny edge detection استفاده شده است. لبه‌های تصویر با استفاده از این الگوریتم را برای تغییرات پیکسلی که بیشتر از مقدار شدت روشنایی 100 و کمتر از 200 هستند یافته و نمایش داده شده است.کد مورد استفاده و نتایج در زیر آمده است:

1. edges\_original = cv2.Canny(img, 100, 200)

2. edges\_gaussian = cv2.Canny(gaussian, 100, 200)

3. edges\_median = cv2.Canny(median, 100, 200)

4. edges\_mean = cv2.Canny(mean, 100, 200)

5.

6. # Display edges

7. cv2.imshow("Original Edges", edges\_original)

8. cv2.imshow("Gaussian Edges", edges\_gaussian)

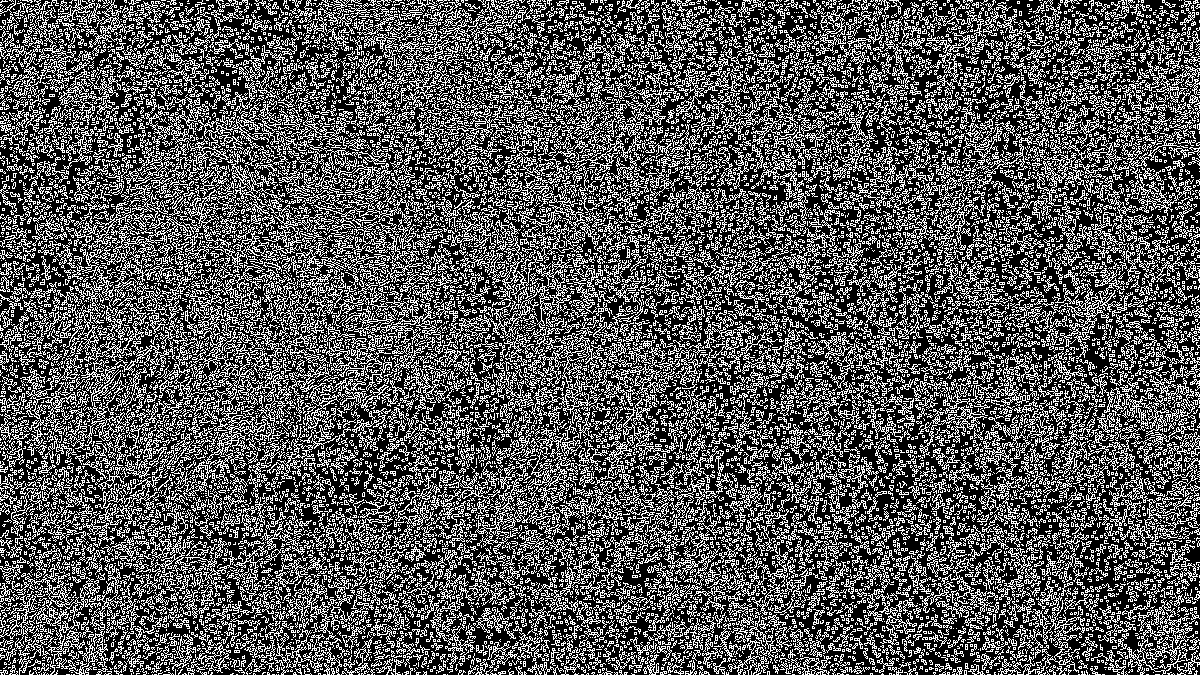
9. cv2.imshow("Median Edges", edges\_median)

10. cv2.imshow("Mean Edges", edges\_mean)

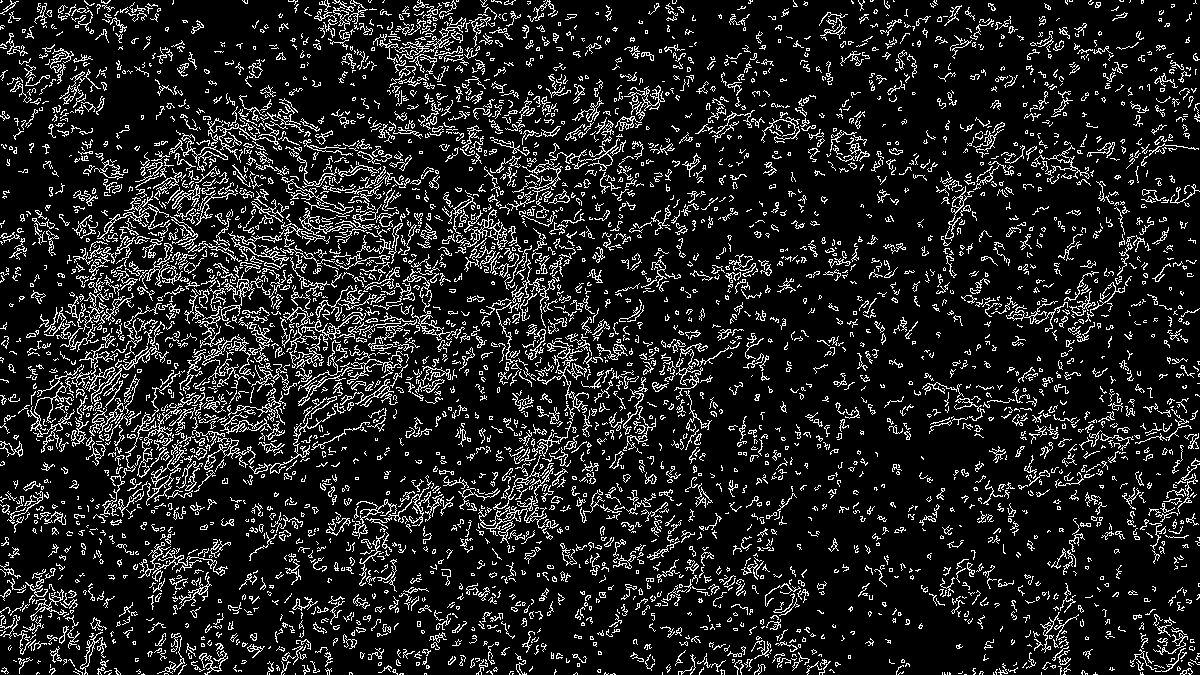
11. cv2.waitKey(0)

12. cv2.destroyAllWindows()

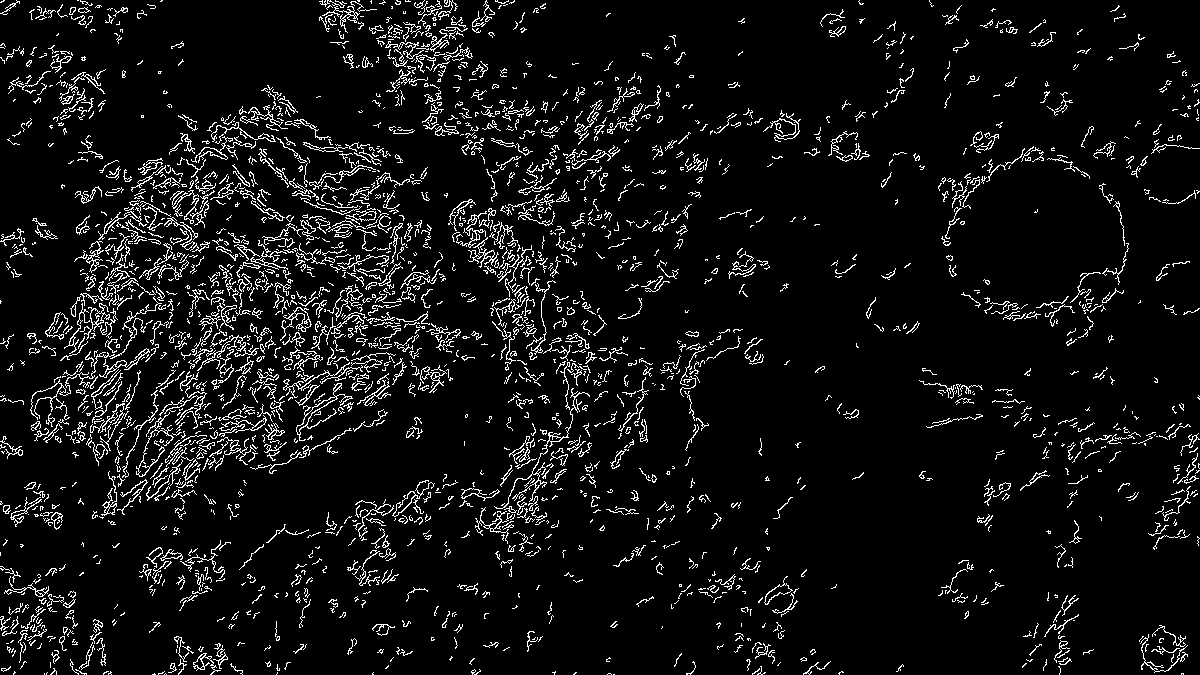
تصویر اصلی



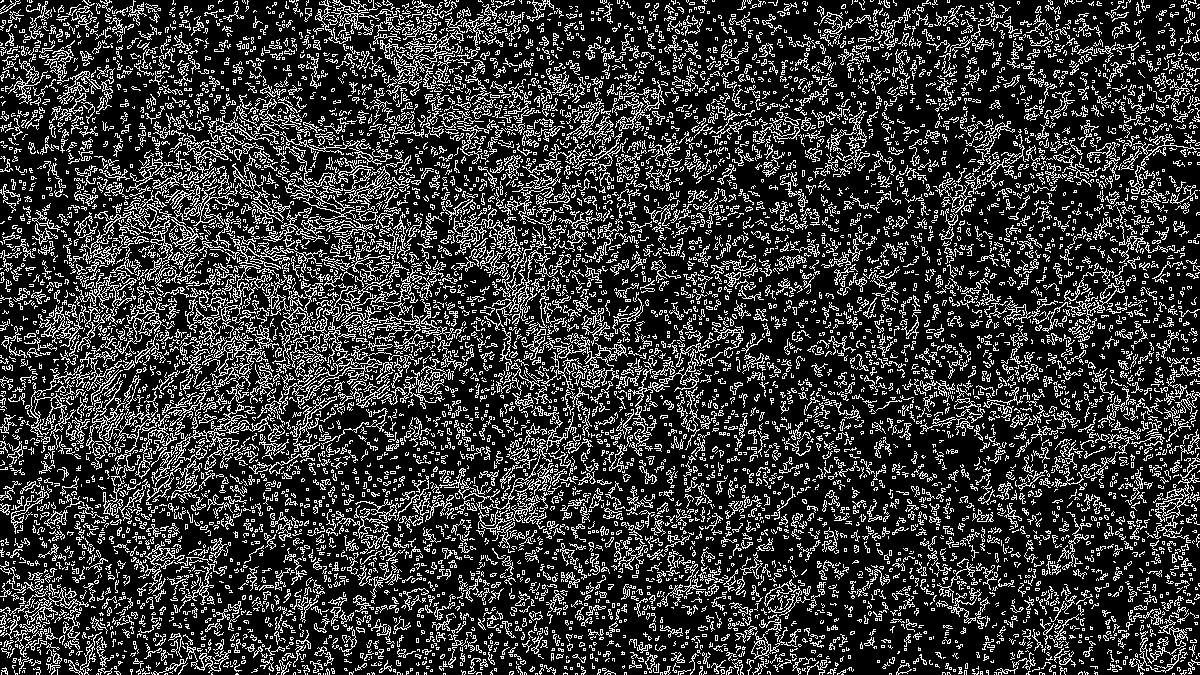
فیلتر میانگین



فیلتر میانه



فیلتر گوسی



مشاهده می­شود که فیلتر گوسی در حفظ لبه های تصویر عملکرد مناسب تری داشته است.

بخش سوم

در این مرحله ابتدا واریانس نویز با کتابخانه skimage محاسبه شد سپس با استفاده از کتابخانه scipy فیلتر واینر پیاده سازی شده است:

1. img = img\_as\_float(img)

2.

3. # Estimate Gaussian noise standard deviation

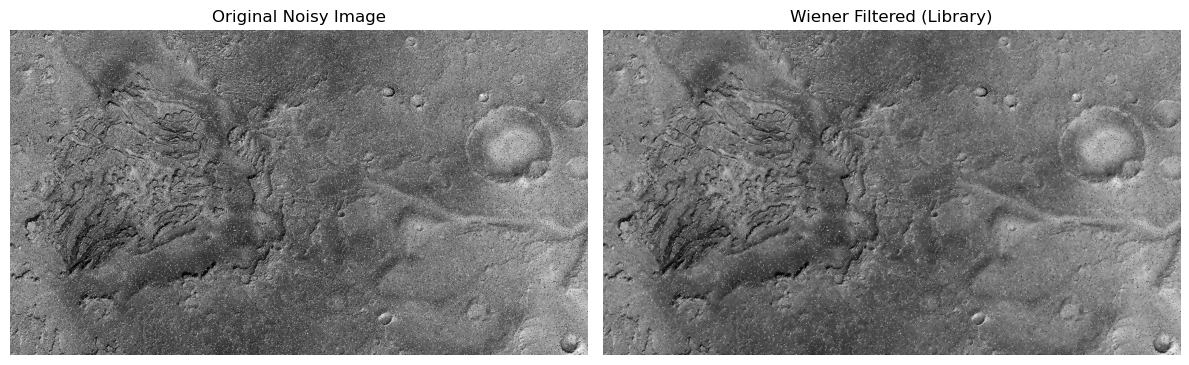
4. sigma\_est = estimate\_sigma(img, channel\_axis=None, average\_sigmas=True)

5.

6. # Apply Wiener filter

7. wiener\_filtered = scipy\_wiener(img, mysize=5, noise=sigma\_est\*\*2)

نتایج به قرار زیر است:



سپس مقادیر PSNR وSSIM برای این فیلتر محاسبه شده است:

PSNR: 26.20 dB

SSIM: 0.878

مشاهده می شود که نتایج به طور قابل ملاحظه ای مطلوب تر شده است.

بخش چهارم

فیلتر میانه تطبیقی علاوه بر اینکه برای نویز نمک فلفلی بسیار مناسب است برای حذف نویز های دیگر نیز عملکرد مطلوبی دارد و به نسبت فیلتر میانه تصویر را کمتر تخریب می کند. پیاده سازی این فیلتر به قرار زیر است:

1. import numpy as np

2. from skimage import io, util

3. import matplotlib.pyplot as plt

4.

5. def adaptive\_median\_filter(image, S\_max):

6.     """

7.     Applies the Adaptive Median Filter to remove impulse noise.

8.

9.     Param:

10.         :image (ndarray): Input grayscale image (uint8).

11.         :S\_max (int): Maximum allowed size of the window (must be odd).

12.

13.     Returns:

14.         :output (ndarray): Filtered image.

15.     """

16.     image = image.astype(np.int32)

17.     pad\_size = S\_max // 2

18.     padded = np.pad(image, pad\_size, mode='edge')

19.     output = np.zeros\_like(image)

20.

21.     rows, cols = image.shape

22.

23.     for i in range(rows):

24.         for j in range(cols):

25.             window\_size = 3

26.             found = False

27.             while window\_size <= S\_max:

28.                 half = window\_size // 2

29.                 x, y = i + pad\_size, j + pad\_size

30.                 window = padded[x - half:x + half + 1, y - half:y + half + 1]

31.

32.                 z\_min = window.min()

33.                 z\_max = window.max()

34.                 z\_med = np.median(window)

35.                 z\_xy = padded[x, y]

36.

37.                 A1 = z\_med - z\_min

38.                 A2 = z\_med - z\_max

39.

40.                 if A1 > 0 and A2 < 0:

41.                     B1 = z\_xy - z\_min

42.                     B2 = z\_xy - z\_max

43.                     if B1 > 0 and B2 < 0:

44.                         output[i, j] = z\_xy

45.                     else:

46.                         output[i, j] = z\_med

47.                     found = True

48.                     break

49.                 else:

50.                     window\_size += 2

51.             if not found:

52.                 output[i, j] = z\_med

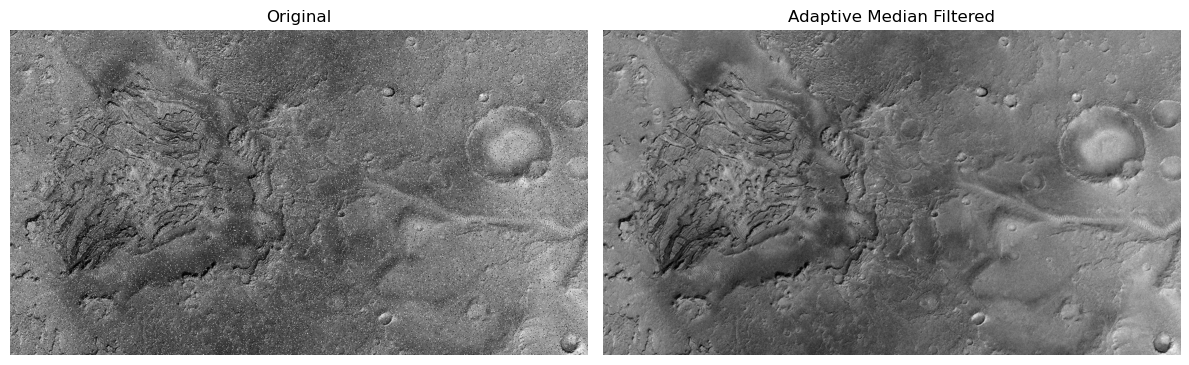
53.     return output.astype(np.uint8)

54.

55. # Apply adaptive median filter

56. adaptive\_median\_filtered = adaptive\_median\_filter(img, S\_max=7)

مقدار حداکثر سایز کرنل برابر 7 در نظر گرفته شده است. خروجی به قرار زیر است:



بسیار واضح است که از نظر بصری تصویر وضوح خیلی خوبی دارد. مقادیر PSNR و SSIM نیز اگرچه از فیلتر واینر کمتر است ولی از فیلتر میانه ساده بهتر است:

PSNR: 17.93 dB

SSIM: 0.376

سوال 3

بخش اول

برای تخمین نویز ابتدا لازم است تا مختصات نویز را از تصویر استخراج کنیم. روشی که در این سوال استفاده شده این است که مختصات سمت چپ بالا از تصویر را که مربوط به پیکسلهای پس زمینه و سیاه هستند را به عنوان اطلاعات نویز بررسی میکند و میانگین واریانس آن ناحیه را به عنوان نویز در نظر می گیر. برای مشخص نمودن مختصات ناحیه مستطیلی از کد زیر استفاده شده است:

1. image\_path = '/mnt/e/Term 4/HW/HW4/DIP-HW4/HW4/images/q3/1.png'

2.

3. # Load the image in grayscale

4. img = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

5. if img is None:

6.     raise ValueError("Image not found!")

7.

8. # Mouse callback function to display pixel info

9. def show\_pixel\_info(event, x, y, flags, param):

10.     if event == cv2.EVENT\_MOUSEMOVE:

11.         pixel\_value = img[y, x]

12.         print(f"x={x}, y={y}, value={pixel\_value}", end="\r")

13.

14. # Create window and set mouse callback

15. cv2.namedWindow('Image')

16. cv2.setMouseCallback('Image', show\_pixel\_info)

17.

18. # Display the image until ESC key is pressed

19. while True:

20.     cv2.imshow('Image', img)

21.     if cv2.waitKey(1) & 0xFF == 27:  # ESC key to exit

22.         break

23.

24. cv2.destroyAllWindows()

در این بخش، هدف اجرای برنامه‌ای است که تصویر سیاه و سفید (grayscale) را بارگذاری کرده و با حرکت موس روی تصویر، مقدار روشنایی (intensity) پیکسل زیر موس را به صورت زنده نمایش دهد. در ابتدا، تصویر مورد نظر با استفاده از تابع cv2.imread از مسیر مشخص‌شده خوانده می‌شود. این تصویر به صورت خاکستری بارگذاری می‌شود تا تنها یک کانال روشنایی داشته باشد. سپس بررسی می‌شود که آیا تصویر به درستی بارگذاری شده است یا خیر؛ در صورتی که تصویر یافت نشود، برنامه با پیام خطا متوقف می‌شود. برای دریافت اطلاعات پیکسل‌ها، یک تابع بازخورد (callback) برای موس تعریف شده است. این تابع هر زمان که موس روی تصویر حرکت کند (cv2.EVENT\_MOUSEMOVE)، مختصات فعلی موس را دریافت کرده و مقدار روشنایی پیکسلی را که موس روی آن قرار دارد از تصویر استخراج کرده و در خروجی چاپ می‌کند. سپس یک پنجره با نام Image ایجاد می‌شود و تابع بازخورد موس به این پنجره اختصاص داده می‌شود. در یک حلقه‌ی بی‌نهایت، تصویر به طور پیوسته نمایش داده می‌شود و تا زمانی که کاربر کلید ESC را فشار نداده باشد، این نمایش ادامه دارد. با فشرده شدن ESC، حلقه متوقف شده و همه‌ی پنجره‌های باز بسته می‌شوند.

ناحیه مورد نظر در راستای محور X 40 پیکسل اول را در نظر می­گیرد و در راستای محور y 50 پیکسل را در نظر می­گیرد بنابراین یک ناحیه با ابعاد 40 در 50 برای محاسبه نویز موزد استفاده قرار گرفته است. محاسبات را برای دو تصویر ایده آل و تخریب ده برای ناحیه متناظر در نظر گرفته شده است.

1. image\_path = '/mnt/e/Term 4/HW/HW4/DIP-HW4/HW4/images/q3/2.png'

2. img = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

3.

4. image\_path\_ideal = '/mnt/e/Term 4/HW/HW4/DIP-HW4/HW4/images/q3/2\_ideal.png'

5. img\_ideal = cv2.imread(image\_path\_ideal, cv2.IMREAD\_GRAYSCALE)

6.

7. y1,y2 = 0, 40

8. x1, x2 = 0, 50

9. region\_1 = img[y1:y2, x1:x2]

10. region\_2 = img\_ideal[y1:y2, x1:x2]

11.

12. mean\_noise1 = np.mean(region\_1)

13. var\_noise1 = np.var(region\_1)

14.

15. mean\_noise2 = np.mean(region\_2)

16. var\_noise2 = np.var(region\_2)

17.

18. print(f"Estimated noise mean(degraded): {mean\_noise1:.2f}")

19. print(f"Estimated noise variance(degraded): {var\_noise1:.2f}")

20.

21. print(f"Estimated noise mean(ideal): {mean\_noise2:.2f}")

22. print(f"Estimated noise variance(ideal): {var\_noise2:.2f}")

برای تصویر اول داریم:

1. Estimated noise mean(degraded): 4.12

2. Estimated noise variance(degraded): 36.80

3. Estimated noise mean(ideal): 0.86

4. Estimated noise variance(ideal): 11.15

برای تصویر دوم مقادیر به قرار زیر است:

Estimated noise mean(degraded): 5.50

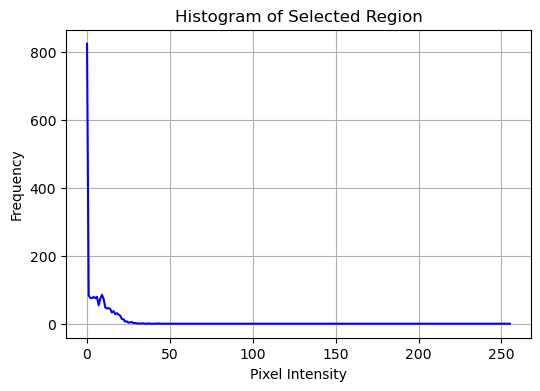
Estimated noise variance(degraded): 44.76

Estimated noise mean(ideal): 2.61

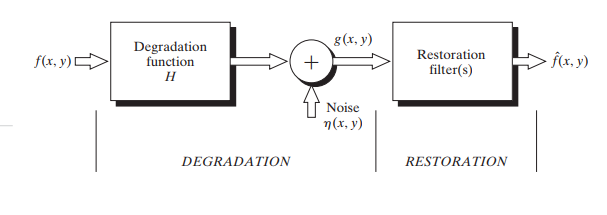
Estimated noise variance(ideal): 6.93

با توجه به نتایج محاسبه شده مقدار میانگین و واریانس نویز برای تصویر اول 4 و 25 و برای تصویر دوم 3 و 36 در نظر گرفته شده است.

هیستوگرام شدت روشنایی برای تصویر نمونه 2 رسم شده که نشان از گوسی بودن نویز دارد:



برای تخمین تابع تبدیل برای دو تصویر با توجه به اینکه تصویر ایده آل و تخریب شده را داشته و نویز را گوسی و مشخص در نظر گرفته ایم لذا با توجه به دیاگرام زیر داریم:



برای تخمین تابع تبدیل ابتدا با دستور زیر n(x,y) تولید شده است. مقادیر میانگین و واریانس ابتدا نرمالیزه شده تا مقادیر خروجی بین 0 و 1 باشد. تصاویر ورودی نیز نرمالیزه شده اند.

1. # Load images

2. ideal = cv2.imread("/mnt/e/Term 4/HW/HW4/DIP-HW4/HW4/images/q3/1\_ideal.png", cv2.IMREAD\_GRAYSCALE)

3. degraded = cv2.imread("/mnt/e/Term 4/HW/HW4/DIP-HW4/HW4/images/q3/1.png", cv2.IMREAD\_GRAYSCALE)

4.

5. # Normalize to [0, 1]

6. ideal\_norm = ideal.astype(np.float32) / 255.0

7. degraded\_norm = degraded.astype(np.float32) / 255.0

8.

9. # Estimated noise info (in 0-255)

10. mu\_n = 4

11. sigma\_n = 5

12.

13. # Normalize to [0,1]

14. mu\_n\_norm = mu\_n / 255.0

15. sigma\_n\_norm = sigma\_n / 255.0

16.

17. # Generate synthetic Gaussian noise matrix

18. np.random.seed(0)  # for reproducibility

19. noise\_matrix = np.random.normal(loc=mu\_n\_norm, scale=sigma\_n\_norm, size=degraded\_norm.shape)

در مرحله بعد مقادیر به حوزه فوریه برده شده و تابع تبدیل در حوزه زمان محاسبه شده است:

1. # Compute FFTs

2. F = fft2(ideal\_norm)

3. G = fft2(degraded\_norm)

4. N = fft2(noise\_matrix)

5.

6. # Compute degradation transfer function H(u,v)

7. # Avoid dividing by zero (add epsilon)

8. H\_estimated1 = (G - N) / F

9.

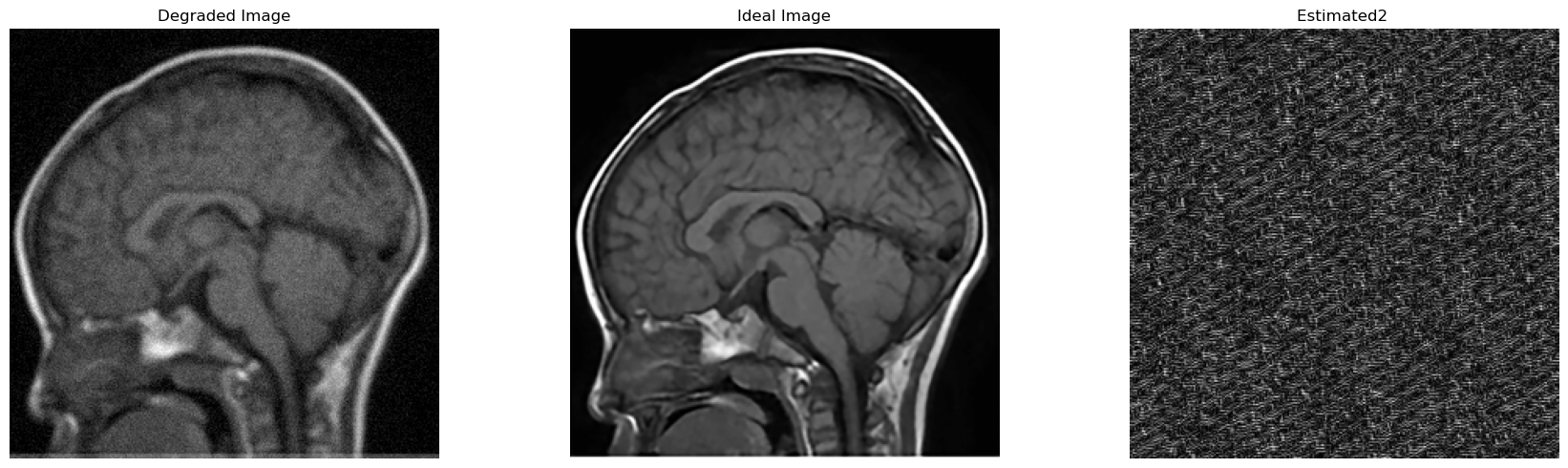
10. # Convert to spatial domain (estimated PSF)

11. psf\_spatial = np.abs(ifft2(H\_estimated1))

نتایج به قرار زیر است:



نتایج برای تصویر دوم نیز به قرار زیر است:

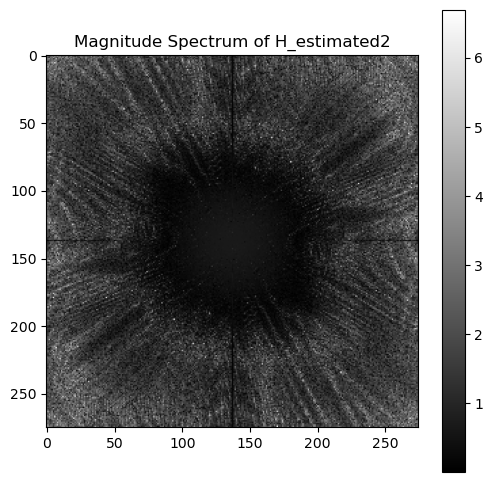


طیف دو تابع تبدیل در زیر آورده شده است:

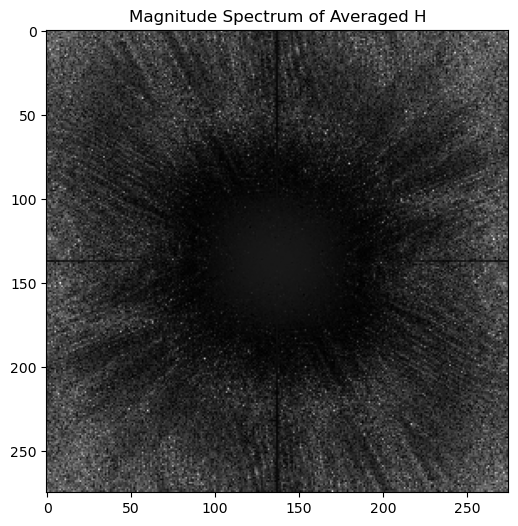
تصویر 1



تصویر 2



یک راه برای ترکیب این دو تابع تبدیل استفاده از میانگین است. علاوه بر این می توان از میانگین وزندار توابع تبدیل نیز برای محاسبه تابع تبدیل سیستم استفاده کرد. در این سوال از میانگین دو تابع تبدیل استفاده شده است.



بخش دوم