Image and Video Processing Lab
Lab 8:Image Restoration

Aruna Shaju Kollannur (SC21M111)

Sub: Image and Video Processing Lab
Department: Avionics(DSP)
Date of Lab sheet: December 7, 2021
Date of Submission: December 23, 2021

Question 1 Restoration of image using Inverse Filtering

Aim

Restoration of blurred image using inverse filtering with and without noise.

Discussion

Inverse Filtering is directly utilizing the notion that the overall SPF of blurring and deblurring filter should be delta(m,n) and thus we have G(w1,w2) = 1/H(w1,w2)

The Image blurring model and deblurring models are given below.

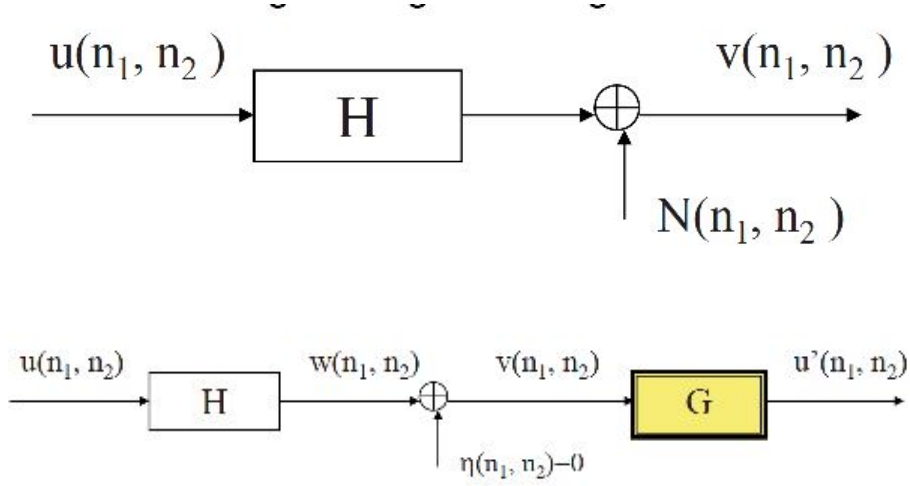

Figure 1: (a) Image Blurring Model (b) Image Deblurring Model

Algorithm

```
Step 1: Start
Step 2: Read the image and convert in into gray-scale.
Step 3: Blur the image and add noise to it
Step 4: Transform the original and noisy image into frequency domain
Step 5: Divide the noisy image output with original image in frequency domain to obtain SPF(H).
Step 6: Invert SPF to obtain G.
Step 7: Perform deblurring and denoisisng using G matrix.
Step 8: Transform Output of Step 7 in spatial domain.
Step 9: Compute mean square error of original and reconstructed image and observe the differences.
```

Program Code - without noise

```
import numpy as np
import cv2

import matplotlib.pyplot as plt

Image_Original = cv2.imread("Cameraman.png", 0)
Image_Blur = cv2.GaussianBlur(Image_Original, (5, 5), 0)

Image_Original_Freq = np.fft.fft2(Image_Original)
Image_Blur_Freq = np.fft.fft2(Image_Blur)

H_SPF = Image_Blur_Freq/Image_Original_Freq
Inverse_H_SPF = 1/H_SPF

Inverse_Filtering_Output_Freq = Image_Blur_Freq*Inverse_H_SPF

Inverse_Filtering_Output = np.fft.ifft2(Inverse_Filtering_Output_Freq)

Inverse_Filtering_Output = np.clip(Inverse_Filtering_Output, 0, 255)
Inverse_Filtering_Output = Inverse_Filtering_Output.astype('uint8')

cv2.imshow('Image_Original', abs(Image_Original))
cv2.waitKey(0)

cv2.imshow('Image_Blur', abs(Image_Blur))
cv2.imwrite('Image_Blur.png',Image_Blur)
cv2.waitKey(0)

plt.figure(num=None, figsize=(8, 6), dpi=80)
plt.imshow(np.log(abs(H_SPF)), cmap='gray');

plt.figure(num=None, figsize=(8, 6), dpi=80)
plt.imshow(np.log(abs(Inverse_H_SPF)), cmap='gray');

cv2.imshow('Inverse_Filtering_Output', abs(Inverse_Filtering_Output))
cv2.imwrite('Inverse_Filtering_Output.png',Inverse_Filtering_Output)
cv2.waitKey(0)

mean_square_error_orig_blur = np.square(np.subtract(Image_Original,Image_Blur)).mean()
mean_square_error_orig_recovered = np.square(np.subtract(Image_Original,
                                  Inverse_Filtering_Output)).mean()

print(f'The Mean Squared error between original and blurred image is
{mean_square_error_orig_blur}')
print(f'The Mean Squared error between original and recovered image
is {mean_square_error_orig_recovered}')
```

Result - without noise

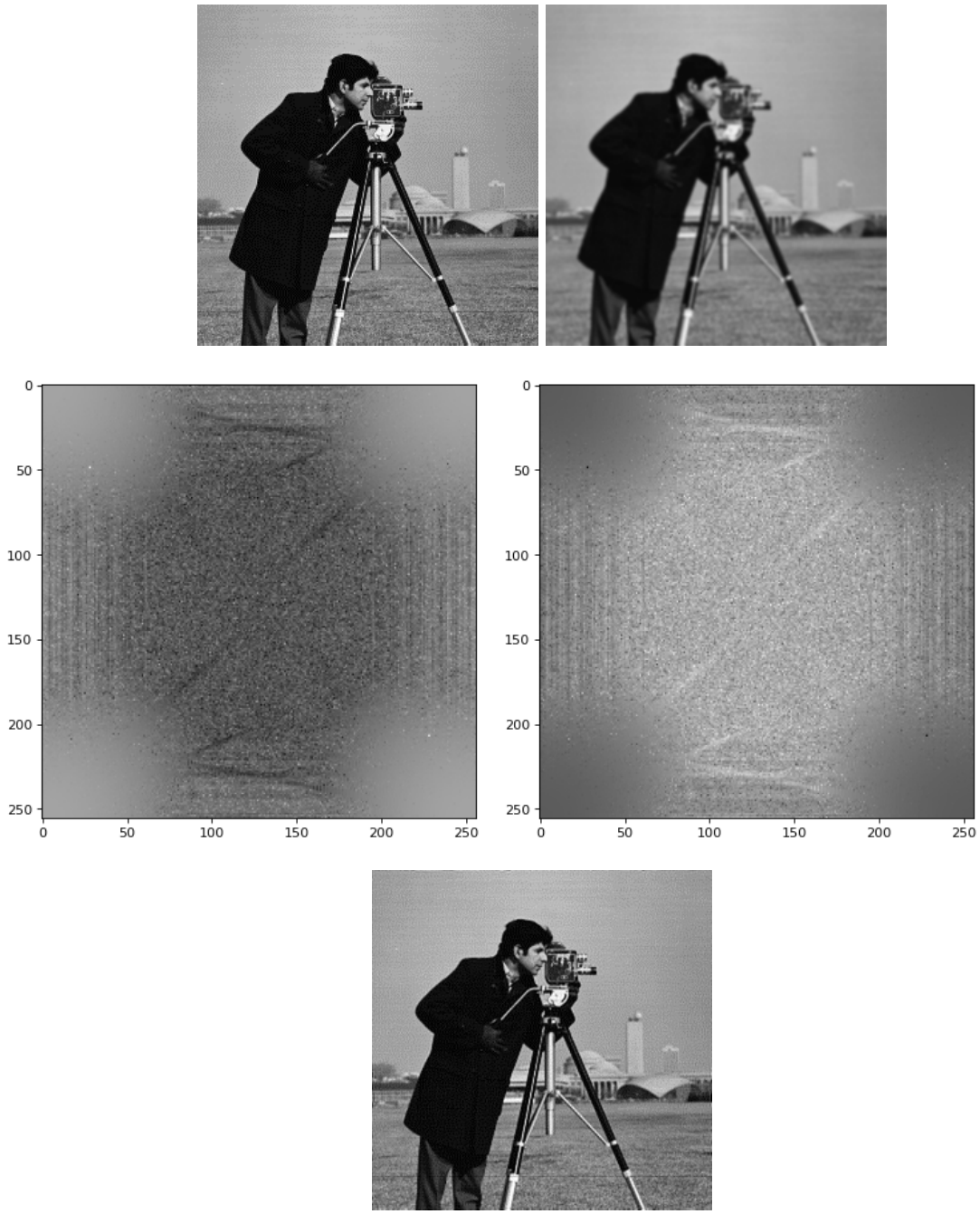

Figure 2: Cameraman (a) Original Image (b) Blurred Image (c)SPF model of blurring (d) Inverse SPF function (deblurring) (e) Restored Image

```
The Mean Squared error between original and blurred image is 60.67596435546875
The Mean Squared error between original and recovered image is 0.0
```

Program Code- With noise

```
import numpy as np
import cv2
```

```
import matplotlib.pyplot as plt

Image_Original = cv2.imread("Cameraman.png", 0)
Image_Blur = cv2.GaussianBlur(Image_Original, (5, 5), 0) + np.random.normal(0,10,(256,256))

Image_Original_Freq = np.fft.fft2(Image_Original)
Image_Blur_Freq = np.fft.fft2(Image_Blur)

H_SPF = Image_Blur_Freq/Image_Original_Freq
Inverse_H_SPF = 1/H_SPF

Inverse_Filtering_Output_Freq = Image_Blur_Freq*Inverse_H_SPF

Inverse_Filtering_Output = np.fft.ifft2(Inverse_Filtering_Output_Freq)

Inverse_Filtering_Output = np.clip(Inverse_Filtering_Output, 0, 255)
Inverse_Filtering_Output = Inverse_Filtering_Output.astype('uint8')

cv2.imshow('Image_Original', abs(Image_Original))
cv2.waitKey(0)

cv2.imshow('Image_Blur_with_noise', abs(Image_Blur))
cv2.imwrite('Image_Blur_with_noise.png',Image_Blur)
cv2.waitKey(0)

plt.figure(num=None, figsize=(8, 6), dpi=80)
plt.imshow(np.log(abs(H_SPF)), cmap='gray');

plt.figure(num=None, figsize=(8, 6), dpi=80)
plt.imshow(np.log(abs(Inverse_H_SPF)), cmap='gray');


cv2.imshow('Inverse_Filtering_Output_noise', abs(Inverse_Filtering_Output))
cv2.imwrite('Inverse_Filtering_Output_noise.png',Inverse_Filtering_Output)
cv2.waitKey(0)

mean_square_error_orig_blur = np.square(np.subtract(Image_Original,Image_Blur)).mean()
mean_square_error_orig_recovered = np.square(np.subtract(Image_Original,
                                  Inverse_Filtering_Output)).mean()

print(f'The Mean Squared error between original and blurred image is
{mean_square_error_orig_blur}')
print(f'The Mean Squared error between original and recovered image
is {mean_square_error_orig_recovered}')
```

Result - With noise

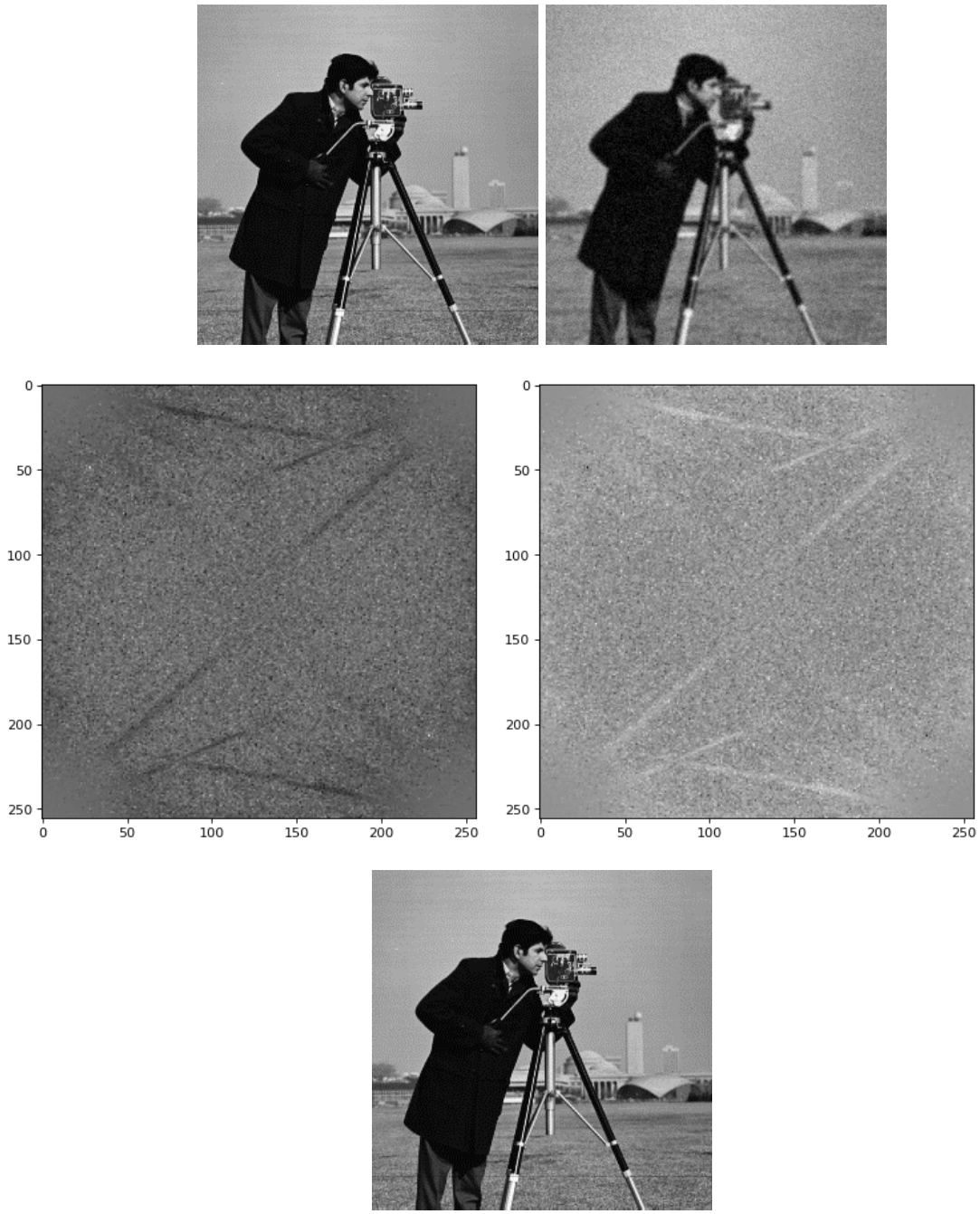

Figure 3: Cameraman (a) Original Image (b) Blurred noisy Image (c)SPF model of blurring (d) Inverse SPF function (deblurring) (e) Restored Image

```
The Mean Squared error between original and blurred image is 330.55841522228417
The Mean Squared error between original and recovered image is 0.184539794921875
```

Inference

The image is restored without any loss in information using inverse filtering when image has no noise. The image is restored almost without any loss in information using inverse filtering when image has with noise. There is some difference between original and reconstructed image observed.

Question 2: Restoration of image using pseudo inverse filtering

Aim

Restoration of blurred image with and without noise using pseudo inverse filtering.

Discussion

Inverse filtering will have problem when there is some noise on the distortion(noise) image. Since G(w1,w2) = 1/H(w1,w2), and if H takes very small values at some frequency, and in order to compensate that supress effect, G will have large values. This could be a big problem when there is some noise, since it will enlarge the noise. Therefore, the pseudo-inverse filter will be applied, and the PSF of G is assigned as :

figure 4

$$G(\omega_1,\omega_2)=\begin{cases}\dfrac{1}{H(\omega_1,\omega_2)}, & \text{if } |H(\omega_1,\omega_2)| \geq \varepsilon \\ 0, & \text{if } |H(\omega_1,\omega_2)| < \varepsilon\end{cases}$$

Algorithm

```
Step 1: Start
Step 2: Read the image and convert in into gray-scale.
Step 3: Blur the image and add noise to it
Step 4: Transform the original and noisy image into frequency domain
Step 5: Declare theta
Step 6: Divide the noisy image output with original image in frequency domain to obtain SPF(H).
Step 7: Invert SPF to obtain G when H < theta, else assign G
as zero.
Step 8: Perform deblurring and denoisisng using G matrix.
Step 9: Transform Output of Step 7 in spatial domain.
Step 10: Compute mean square error of original and
reconstructed image and observe the differences.
```

Program Code - Without Noise

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

Image_Original = cv2.imread("Cameraman.png", 0)
Image_Blur = cv2.GaussianBlur(Image_Original, (5, 5), 0)

Image_Original_Freq = np.fft.fft2(Image_Original)
Image_Blur_Freq = np.fft.fft2(Image_Blur)

Theta = 0
height, width = Image_Original.shape

H_SPF = Image_Blur_Freq/Image_Original_Freq
Inverse_H_SPF = 1/H_SPF
```

```
for i in range(height):
    for j in range(width):

     if(1/Inverse_H_SPF[i,j] < Theta):
        Inverse_H_SPF[i,j] = 0

Inverse_Filtering_Output_Freq = Image_Blur_Freq*Inverse_H_SPF

Inverse_Filtering_Output = np.fft.ifft2(Inverse_Filtering_Output_Freq)

Inverse_Filtering_Output = np.clip(Inverse_Filtering_Output, 0, 255)
Inverse_Filtering_Output = Inverse_Filtering_Output.astype('uint8')

cv2.imshow('Image_Original', abs(Image_Original))
cv2.waitKey(0)

cv2.imshow('Image_Blur', abs(Image_Blur))
cv2.imwrite('Image_Blur.png',Image_Blur)
cv2.waitKey(0)

plt.figure(num=None, figsize=(8, 6), dpi=80)
plt.imshow(np.log(abs(H_SPF)), cmap='gray');

plt.figure(num=None, figsize=(8, 6), dpi=80)
plt.imshow(np.log(abs(Inverse_H_SPF)), cmap='gray');

cv2.imshow('Inverse_Filtering_Output', abs(Inverse_Filtering_Output))
cv2.imwrite('Inverse_Filtering_Output.png',Inverse_Filtering_Output)
cv2.waitKey(0)

mean_square_error_orig_blur = np.square(np.subtract(Image_Original,Image_Blur)).mean()
mean_square_error_orig_recovered = np.square(np.subtract(Image_Original,
                                  Inverse_Filtering_Output)).mean()

print(f'The Mean Squared error between original and blurred image is
{mean_square_error_orig_blur}')
print(f'The Mean Squared error between original and recovered image
is {mean_square_error_orig_recovered}')
```

Result- Without Noise

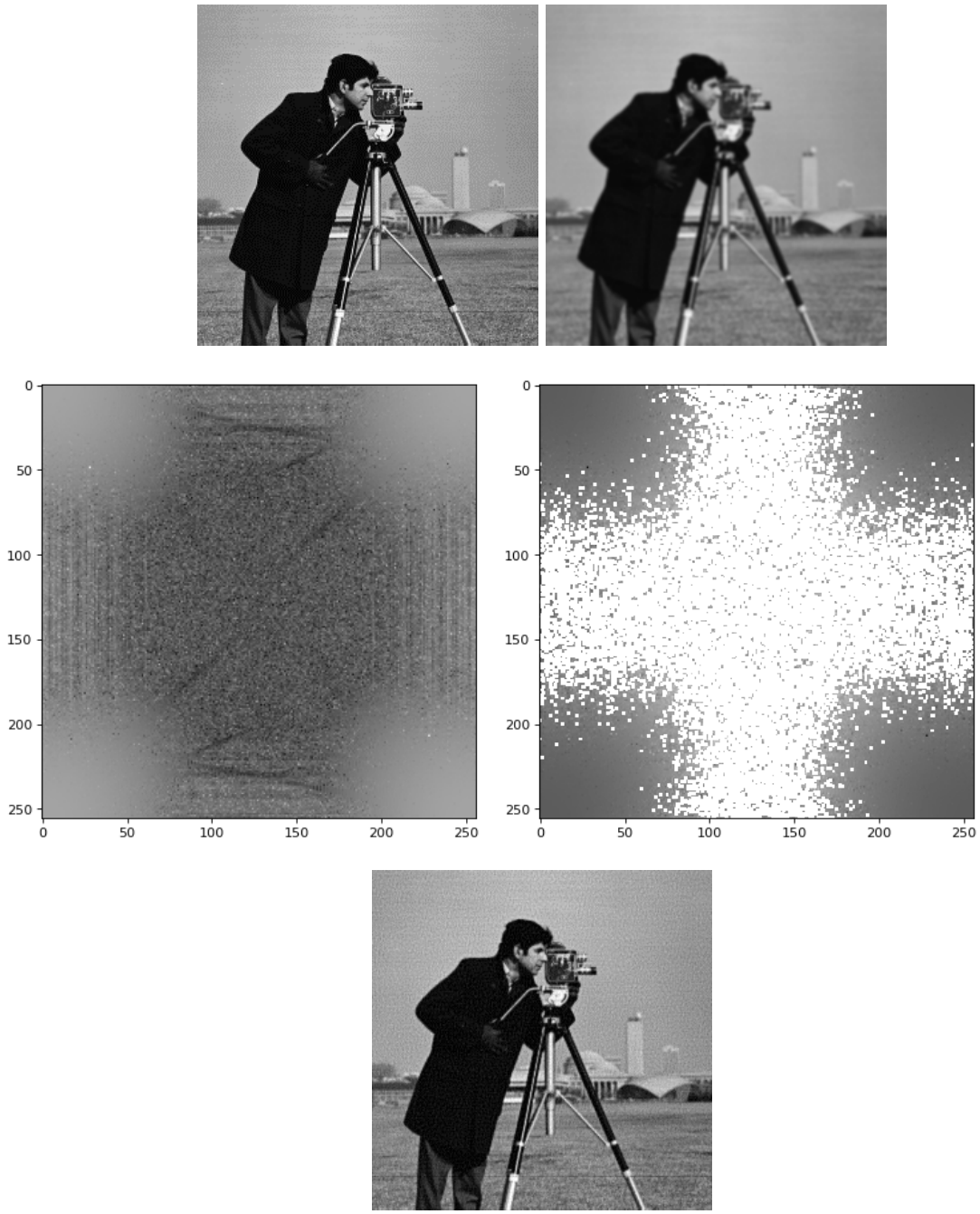

Figure 4: Cameraman (a) Original Image (b) Blurred Image (c)SPF model of blurring (d) Pseudo Inverse SPF function (deblurring) (e) Restored Image

```
The Mean Squared error between original and blurred image is 60.67596435546875
The Mean Squared error between original and recovered image is 33.0185546875
```

Program Code - With Noise

```
import numpy as np
import cv2
```

```
import matplotlib.pyplot as plt

Image_Original = cv2.imread("Cameraman.png", 0)
Image_Blur = cv2.GaussianBlur(Image_Original, (5, 5), 0)
Image_Blur_with_Noise = Image_Blur + np.random.normal(0,10,(256,256))

Theta = 0.01
height, width = Image_Original.shape

Image_Original_Freq = np.fft.fft2(Image_Original)
Image_Blur_Freq = np.fft.fft2(Image_Blur_with_Noise)

H_SPF = Image_Blur_Freq/Image_Original_Freq
Inverse_H_SPF = 1/H_SPF

for i in range(height):
    for j in range(width):

     if(1/Inverse_H_SPF[i,j] < Theta):
        Inverse_H_SPF[i,j] = 0

Inverse_Filtering_Output_Freq = Image_Blur_Freq*Inverse_H_SPF

Inverse_Filtering_Output = np.fft.ifft2(Inverse_Filtering_Output_Freq)

mean_square_error_orig_blur = np.square(np.subtract(Image_Original,Image_Blur_with_Noise)).mean()
mean_square_error_orig_recovered = np.square(np.subtract(Image_Original,
                                  Inverse_Filtering_Output)).mean()

print(f'The Mean Squared error between original and blurred noisy
image is {mean_square_error_orig_blur}')
print(f'The Mean Squared error between original and recovered image
is {mean_square_error_orig_recovered}')

Inverse_Filtering_Output = np.clip(Inverse_Filtering_Output, 0, 255)
Inverse_Filtering_Output = Inverse_Filtering_Output.astype('uint8')


plt.figure(num=None, figsize=(8, 6), dpi=80)
plt.imshow(np.log(abs(H_SPF)), cmap='gray');

plt.figure(num=None, figsize=(8, 6), dpi=80)
plt.imshow(np.log(abs(Inverse_H_SPF)), cmap='gray');

cv2.imshow('Image_Original', abs(Image_Original))
cv2.waitKey(0)

cv2.imshow('Image_Blur', abs(Image_Blur))
cv2.imwrite('Pseudo_Inverse_Image_Blur.png',Image_Blur)
cv2.waitKey(0)

Image_Blur_with_Noise = np.clip(Image_Blur_with_Noise, 0, 255)
Image_Blur_with_Noise = Image_Blur_with_Noise.astype('uint8')
cv2.imshow('Image_Blur_with_Noise', abs(Image_Blur_with_Noise))
cv2.imwrite('Image_Blur_with_Noise.png',Image_Blur)
cv2.waitKey(0)
```

```
cv2.imshow('Psuedo_Inverse_Filtering_Output', abs(Inverse_Filtering_Output))
cv2.imwrite('Pseudo_Inverse_Filtering_Output.png',Inverse_Filtering_Output)
cv2.waitKey(0)
```

Result- With Noise

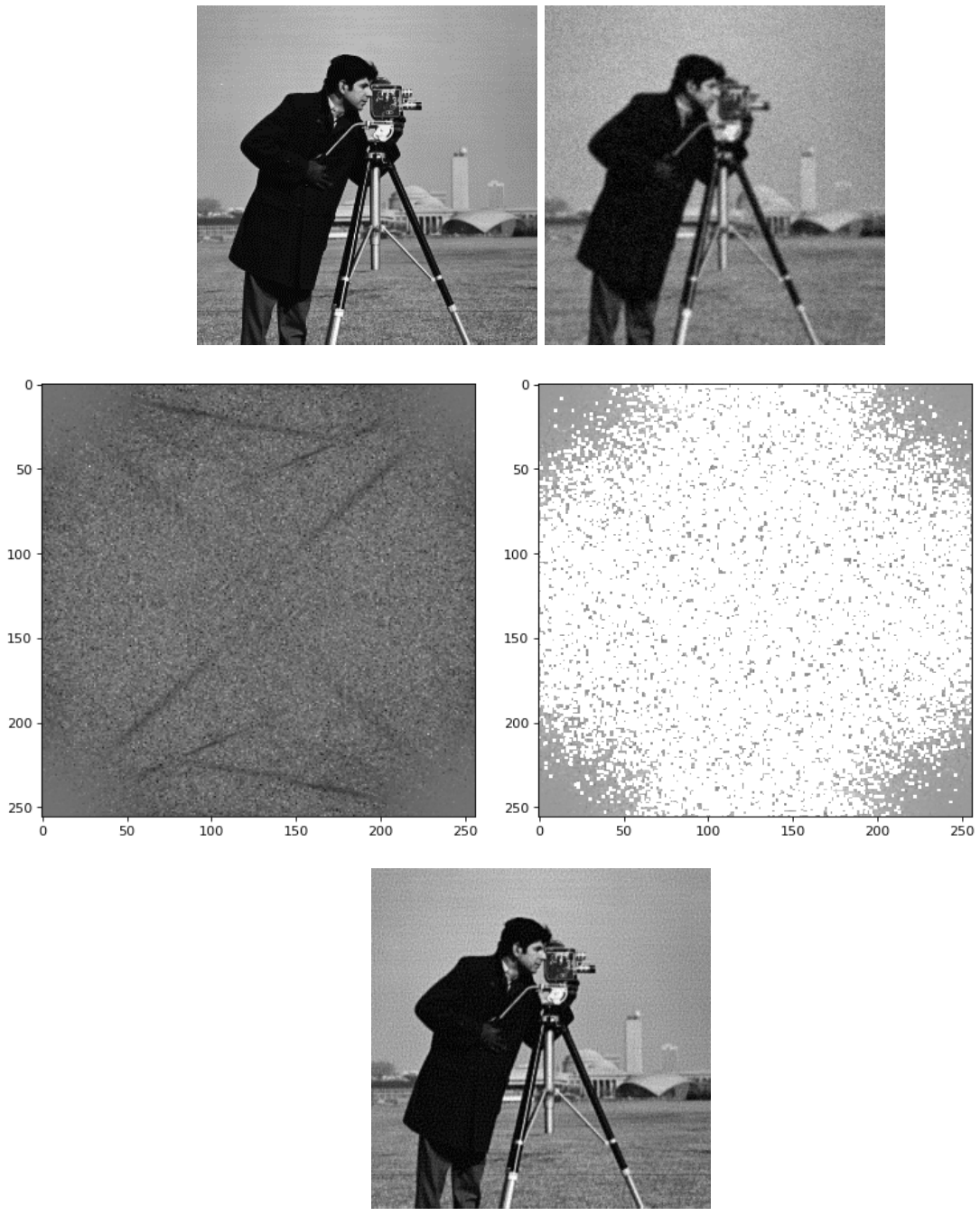

Figure 5: Cameraman (a) Original Image (b) Blurred Image with noise (c)SPF model of blurring (d) Pseudo Inverse SPF function (deblurring) (e) Restored Image

```
The Mean Squared error between original and blurred noisy image is 328.98387734939047
The Mean Squared error between original and recovered image is 81.20645249938954
```

Inference

Pseudo Inverse Filtering helps reduce the contribution of noise in places where 1/H factor is very high, which under normal inverse filtering would increase the affect of noise in the deblurrd image.

Question 3 : Restoration of Image using Wiener Filter

Aim

Restoring a noisy image using wiener filter.

Discussion

Wiener Filtering is a liner estimation filter, which could be used to minimize the MSE between the restoration and original image. When there is no noise in the system, this would be equivalent to inverse filter.

Algorithm

```
Step 1: Start
Step 2: Read noisy image.
Step 3: Pass noisy image through wiener filter.
Step 4: Obtain Reconstructed Image.
Step 5: Display reconstructed image and observe mena square error with original image.
```

Program Code

```
from PIL import Image, ImageOps
from scipy.signal.signaltools import wiener
import matplotlib.pyplot as plt
from Inverse_Filtering import Image_Original
import numpy as np

Cameraman_Image = Image.open("Cameraman.png")
Cameraman_gray =  ImageOps.grayscale(Cameraman_Image)

img = Cameraman_gray
filtered_img = wiener(img, (5, 5))  #Filter the image
f, (plot1, plot2) = plt.subplots(1, 2)

mean_square_error_orig_recovered = np.square(np.subtract(Image_Original,filtered_img)).mean()

print(f'The Mean Squared error between original and recovered image
is {mean_square_error_orig_recovered}')

plot1.imshow(img)
plot2.imshow(filtered_img)
plt.show()
```

Result

Figure 6: Cameraman (a) Original Image (b) Blurred Image with noise (e) Restored Image