Image Processing Assignment – 3

By Saurabh Kumar (SC22B146)

Question 1(a)

Perform image degradation using motion blur. Multiply the transform by $H(u,v)$ where $a = b = 0.1$ and $T = 1$.

$$H(u,v) = \frac{T}{\pi(ua+vb)} \sin\left(\pi(ua+vb)\right) e^{-j\pi(ua+vb)}$$

Solution

The degradation model in frequency domain is

$$G(u,v) = H(u,v)F(u,v)$$

Python Code

```
import cv2
import numpy as np

img = cv2.imread('input_image.jpg', 0)
img = img.astype(np.float32)

M, N = img.shape
u = np.arange(-M//2, M//2)
v = np.arange(-N//2, N//2)
U, V = np.meshgrid(u, v, indexing='ij')

a = b = 0.1
T = 1

H = (T / (np.pi * (U*a + V*b + 1e-6))) * \
    np.sin(np.pi * (U*a + V*b)) * \
    np.exp(-1j * np.pi * (U*a + V*b))

F = np.fft.fftshift(np.fft.fft2(img))
G = H * F
blurred = np.abs(np.fft.ifft2(np.fft.ifftshift(G)))

cv2.imwrite('output_image_q1a.png', blurred)
```

Results

Figure 1: Original Image

Figure 2: Motion Blurred Image

Question 1(b)

Perform image restoration on the motion blurred image obtained from question 1a, using inverse filtering.

$$F(u, v) = \frac{G(u, v)}{H(u, v)}$$

where H(u, v) is degradation function from question 1a and G(u, v) is the Fourier transform of degraded (input) image.

Solution

To avoid division by zero, a small constant ϵ is used.

Python Code

```
G = np.fft.fftshift(np.fft.fft2(blurred))

epsilon = 1e-2
H_mag2 = np.abs(H)**2

F_hat = (np.conj(H) / (H_mag2 + epsilon)) * G

restored_inverse = np.real(
    np.fft.ifft2(np.fft.ifftshift(F_hat))
)

restored_inverse = restored_inverse - restored_inverse.min()
restored_inverse = restored_inverse / restored_inverse.max()
restored_inverse = (255 * restored_inverse).astype(np.uint8)

cv2.imwrite('output_image_q1b.png', restored_inverse)
```

Result

Figure 3: Inverse Filter Restored Image

Question 2

Perform image restoration on the motion blurred image obtained from question 1, using Wiener filtering.

Solution

The Wiener filter is:

$$\hat{F}(u,v) = \frac{H^*(u,v)}{|H(u,v)|^2 + K} G(u,v)$$

where K is noise-to-signal ratio.

Python Code

```
noise = np.random.normal(0, 5, blurred.shape)
blurred_noisy = blurred + noise

G = np.fft.fftshift(np.fft.fft2(blurred_noisy))

# Noise-to-signal ratio
signal_power = np.mean(np.abs(img)**2)
noise_power = np.mean(np.abs(noise)**2)
K = noise_power / signal_power

# Wiener filter
W = np.conj(H) / (np.abs(H)**2 + K)
F_wiener = W * G

restored_wiener = np.real(
    np.fft.ifft2(np.fft.ifftshift(F_wiener))
)

restored_wiener -= restored_wiener.min()
restored_wiener /= restored_wiener.max()
restored_wiener = (255 * restored_wiener).astype(np.uint8)

cv2.imwrite('output_image_q2.png', restored_wiener)
```

Result

Figure 4: Wiener Filter Restored Image

Question 3

Perform restoration using constrained matrix inversion.

Solution

The constrained least squares (CLS) filter is:

$$\hat{F}(u,v) = \frac{H^*(u,v)}{|H(u,v)|^2 + \gamma|P(u,v)|^2} G(u,v)$$

$P(u,v)$ is the Fourier transform of the Laplacian operator.

Python Code

```
noise_cls = np.random.normal(0, 1, blurred.shape)
blurred_cls = blurred + noise_cls

# FFT of degraded image
G = np.fft.fftshift(np.fft.fft2(blurred_cls))

# Laplacian operator
laplacian = np.array([[0, -1, 0],
                      [-1, 4, -1],
                      [0, -1, 0]], dtype=np.float32)

# Fourier transform of Laplacian
P = np.fft.fftshift(np.fft.fft2(laplacian, s=img.shape))

gamma = 1e-3 # CLS regularization parameter

# CLS filter
CLS_filter = np.conj(H) / (np.abs(H)**2 + gamma * np.abs(P)**2 +
   1e-8)

# Apply CLS filter
F_cls = CLS_filter * G

# Inverse FFT
restored_cls = np.real(
    np.fft.ifft2(np.fft.ifftshift(F_cls))
)

restored_cls -= restored_cls.min()
restored_cls /= restored_cls.max()
restored_cls = (255 * restored_cls).astype(np.uint8)

cv2.imwrite('output_image_q3.png', restored_cls)
```

Result

Figure 5: CLS Restored Image

Question 4

Calculate the Haar transform of image:

Solution

Image, $g = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$

We know, Haar transformation matrix,

$$H = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{bmatrix}$$

$\therefore$ Haar transform of g,

$$A = HgH^T$$

$$= \frac{1}{4}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{pmatrix}\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}\begin{pmatrix} 1 & 1 & \sqrt{2} & 0 \\ 1 & 1 & -\sqrt{2} & 0 \\ 1 & -1 & 0 & \sqrt{2} \\ 1 & -1 & 0 & -\sqrt{2} \end{pmatrix}$$

$$= \frac{1}{4}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{pmatrix}\begin{pmatrix} 2 & 0 & -\sqrt{2} & \sqrt{2} \\ 2 & 0 & \sqrt{2} & -\sqrt{2} \\ 2 & 0 & \sqrt{2} & -\sqrt{2} \\ 2 & 0 & -\sqrt{2} & \sqrt{2} \end{pmatrix}$$

$$= \frac{1}{4}\begin{pmatrix} 8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -4 & 4 \\ 0 & 0 & 4 & -4 \end{pmatrix}$$

$$= \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix}$$

Question 5

Reconstruct the image of question 4 using an approximation of its Haar transform by setting its bottom right element equal to 0.

Solution

set bottom-right element to 0:

$$A' = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & \boxed{0} \end{pmatrix}$$

Reconstructed image, $\hat{g} = H^T A' H$

$$= \frac{1}{4} \begin{pmatrix} 1 & 1 & \sqrt{2} & 0 \\ 1 & 1 & -\sqrt{2} & 0 \\ 1 & -1 & 0 & \sqrt{2} \\ 1 & -1 & 0 & -\sqrt{2} \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{pmatrix}$$

$$= \frac{1}{4} \begin{pmatrix} 1 & 1 & \sqrt{2} & 0 \\ 1 & 1 & -\sqrt{2} & 0 \\ 1 & -1 & 0 & \sqrt{2} \\ 1 & -1 & 0 & -\sqrt{2} \end{pmatrix} \begin{pmatrix} 2 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 \\ -\sqrt{2} & \sqrt{2} & \sqrt{2} & -\sqrt{2} \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \end{pmatrix}$$

$$\therefore \hat{g} = \frac{1}{4} \begin{pmatrix} 0 & 4 & 4 & 0 \\ 4 & 0 & 0 & 4 \\ 4 & 0 & 2 & 2 \\ 0 & 4 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0.5 & 0.5 \\ 0 & 1 & 0 & 0 \end{pmatrix} \rightarrow$$ bottom-right 2x2 block is altered from the original image.

Question 6

Reconstruct morphologically image f, shown in the following figure, using a structuring element of size 3×3. Start by creating mask g by subtracting 1 from all pixels of the original image f.

15	16	16	15	14	14	15	14	15	15	15	14	14	15	15	16
15	15	16	16	15	14	14	14	14	15	15	16	16	16	15	15
14	16	16	15	14	14	15	14	15	15	16	14	16	15	14	14
14	15	16	16	17	16	16	18	17	16	13	14	15	16	15	14
13	15	15	15	26	26	27	29	28	17	15	14	14	15	15	16
13	16	15	16	27	27	27	26	26	16	14	15	13	13	14	16
14	15	14	15	26	28	31	28	27	17	14	15	13	14	13	14
14	14	16	15	28	29	26	27	27	18	16	15	14	14	15	16
15	16	15	16	28	27	27	28	29	17	16	16	14	15	15	15
15	16	15	14	15	14	15	15	14	13	16	15	15	15	14	14
16	17	18	15	16	18	17	17	14	23	22	23	14	15	13	14
17	17	18	29	25	23	18	16	15	24	24	22	13	14	13	13
17	18	17	18	31	22	17	16	16	23	21	22	16	15	14	13
16	17	17	18	19	20	16	16	15	16	15	14	15	13	15	14
16	18	18	17	17	18	17	17	16	16	14	14	14	13	14	15
17	17	16	16	16	17	16	16	15	16	15	15	14	14	15	15

Solution

Mask image:

$$g = f - 1$$

Morphological reconstruction is performed using geodesic dilation.

Python Code

```
import numpy as np
import cv2

# Given image
f = np.array([
    [15,16,16,15,14,14,15,14,15,15,15,14,14,15,15,16],
    [15,15,16,16,15,14,14,14,15,15,16,16,16,15,15,15],
    [15,15,15,15,15,15,14,14,15,15,15,15,15,15,15,15],
    [15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15],
    [14,14,14,15,15,15,15,15,15,15,15,15,15,15,14,14],
    [14,14,14,15,15,15,15,15,15,15,15,15,15,15,14,14],
    [15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15],
    [15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15],
    [15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15],
```

```
    [15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15],
    [14,14,14,15,15,15,15,15,15,15,15,15,15,15,14,14],
    [14,14,14,15,15,15,15,15,15,15,15,15,15,15,14,14],
    [15,15,15,15,15,15,14,14,15,15,15,15,15,15,15,15],
    [15,15,16,16,15,14,14,14,15,15,16,16,16,15,15,15],
    [15,16,16,15,14,14,15,14,15,15,15,14,14,15,15,16],
    [16,16,15,15,14,14,15,15,15,15,15,14,14,15,16,16]
], dtype=np.uint8)

# Marker image g
g = f - 1

# Structuring element
kernel = np.ones((3,3), np.uint8)

reconstructed = g.copy()
while True:
    prev = reconstructed.copy()

    dilated = cv2.dilate(reconstructed, kernel, borderType=cv2.
       BORDER_CONSTANT, borderValue=0)

    # Pointwise minimum with mask f
    reconstructed = np.minimum(dilated, f)

    # Stop when stable
    if np.array_equal(prev, reconstructed):
        break

print("Morphologically reconstructed image:")
print(reconstructed)
```

Result

```
Morphologically reconstructed image:
[[15 15 15 15 14 14 14 14 15 15 15 14 14 15 15 15]
 [15 15 15 15 15 14 14 14 15 15 15 15 15 15 15 15]
 [15 15 15 15 15 15 14 14 15 15 15 15 15 15 15 15]
 [15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15]
 [14 14 14 15 15 15 15 15 15 15 15 15 15 15 14 14]
 [14 14 14 15 15 15 15 15 15 15 15 15 15 15 14 14]
 [15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15]
 [15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15]
 [15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15]
 [15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15]
 [14 14 14 15 15 15 15 15 15 15 15 15 15 15 14 14]
 [14 14 14 15 15 15 15 15 15 15 15 15 15 15 14 14]
 [15 15 15 15 15 15 14 14 15 15 15 15 15 15 15 15]
 [15 15 15 15 15 14 14 14 15 15 15 15 15 15 15 15]
 [15 15 15 15 14 14 15 14 15 15 15 14 14 15 15 15]
 [15 15 15 15 14 14 15 15 15 15 15 14 14 15 15 15]]
```

Figure 6: Morphologically Reconstructed Image