Labsheet 7
Image Enhancement,Degradation and Restoration

Ragja Palakkadavath
(SC18M003)

Sub: Image and Video Processing Lab
Department: Mathematics(MLC)
Date of Lab sheet: September 20, 2018
Date of Submission: September 26, 2018

1. Image Enhancement

- Implement global unsharp masking algorithm (Refer Maria Petrou Page No. 357)
- Implement single scale retinex algorithm (Refer Maria Petrou Page No. 360)

Aim

To implement global unsharp masking algorithm and single scale retinex algorithm techniques for better enhancement to an image

Theory/Discussion

Unsharp Masking: This algorithm subtracts from the original image a blurred version of it, so that only high frequency details are left, which are subsequently used to form the enhanced image. The blurred version is usually created by convolving the original image with a Gaussian mask

Single Scale Retinex: This algorithm consists of two basic ingredients:

1. (i) local grey value normalisation by division with the local mean value
2. (ii) conversion into a logarithmic scale that spreads more the dark grey values and less the bright values

The transformation of the original grey value f(x, y) to a new grey value g(x, y) is expressed as:
$g(x, y) = ln(f(x, y) + 1) - ln\overline{f(x, y)}$

Algorithm

1. Read an Image
2. Convolve the image with a gaussian/average filter
3. Find the difference of the convolved image and original image
4. Assign values to the new image based on this difference - if the difference is too low, set that pixel value as 0 or too high, set that pixel value as 255, else keep the pixel value as it is
5. Save the image

1. Read an Image
2. Convolve the image with a gaussian/average filter
3. Find the difference of the natural log transform of the convolved and original image
4. Save the image

Code

```python
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 26 20:47:44 2018

@author: IIST
"""

from PIL import Image
import numpy as np
from matplotlib import pyplot as plt
img = Image.open('cameraman.tif').convert('L')
image = np.array(img)
h= np.ones((3,3))
h=np.float_(h)
h=h/(9)
f_image=np.fft.fft2(image)
final_filter = np.zeros((256,256))
final_filter[:h.shape[0],:h.shape[1]] = h

f_filter=np.fft.fft2(final_filter)

final_image=f_image*f_filter
image2=np.fft.ifft2(final_image)

unsharp_image = np.subtract(image,image2)
imgenh=np.zeros((256,256))
for i in range(0,256):
        for j in range(0,256):
            if(unsharp_image[i,j]<=-50):
                imgenh[i,j]=0
            elif(unsharp_image[i,j]>=50):
                imgenh[i,j]=255
            else:
                imgenh[i,j]=image[i,j]
plt.imsave('1_a_unmask_sharpening.png',imgenh, cmap='gray')
```

```
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 26 20:47:44 2018

@author: IIST
"""

import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import math

img = Image.open('cameraman.tif').convert('L')
image = np.array(img)
h= np.ones((256,256))
h=np.float_(h)
h=h/(256*256)
f_image=np.fft.fft2(image)
#final_filter = np.zeros((256,256))
#final_filter[:h.shape[0],:h.shape[1]] = h

f_filter=np.fft.fft2(h)

final_image=f_image*f_filter
image2=np.fft.ifft2(final_image)

imgenh=np.zeros((256,256))
for i in range(0,256):
        for j in range(0,256):
            imgenh[i][j]=(math.log((image[i][j]+1)/image2[i][j]))*255
plt.imsave('1_retinex.png',imgenh, cmap='gray')
```

Result

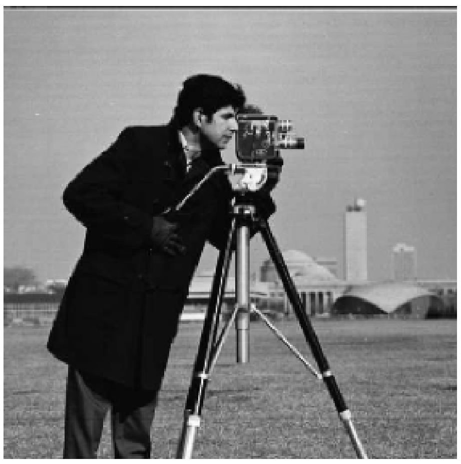

Figure 1: Original Cameraman Image

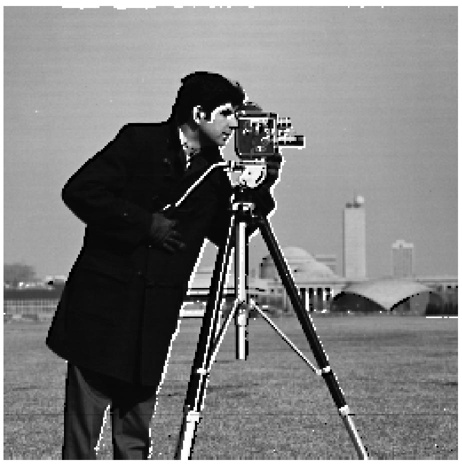

Figure 2: Global Unmask Sharpened Image

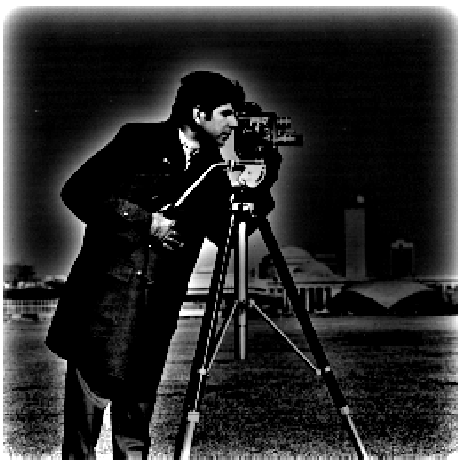

Figure 3: Retinex Transformed Image

Inference

Sharpened version of the image was obtained after applying the global unsharp masking algorithm Single scale retinex algorithm was applied on the image

2. Image Degradation

Perform image degradation using motion blur
(use Book.tif)
Steps:

- Compute Fourier transform of image
- Multiply the transform by H(u, v) given by:$H(u,v) = \frac{T}{\pi(ua+vb)} sin(\pi(ua+vb))e^{-j\pi(ua+vb)}$ where a = b = 0.1 and T = 1
- Take the inverse transform of resultant image to get the motion blurred image (Refer Gonzalez Page 350, Example 5.10)

Aim

To learn to degrade the given image by a motion blur degradation function

Theory/Discussion

Motion blur is the apparent streaking of moving objects in a photograph or a sequence of frames, such as a film or animation. It results when the image being recorded changes during the recording of a single exposure, due to rapid movement or long exposure. Deconvolution or Inverse Filtering can be a method to remove this degradation if the point spread function/degradation function is known

Algorithm

```
1. Read an Image
2. Compute the Fourier Transform of the image
3. Form the H matrix using the given formula for Motion Blur Degradation
4. Convolve Fourier Transform of Image and H
5. Take inverse Fourier Transform of the modified image
6. Save the image
```

Code

```python
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 26 14:49:39 2018

@author: IIST
"""
import numpy as np
import math
from PIL import Image
from scipy import fftpack as fftp
import imageio
import cmath
from matplotlib import pyplot as plt
def return_degraded_image():
    img = Image.open('C:/Users/IIST/Downloads/Ragja/7_SC18M003_Ragja_IP2018/Book.tif')
    img.load()
    data = np.asarray( img, dtype="float64" )

    image_fft=fftp.fft2(data)
    a=1
    b=1
    T=1
    H=np.zeros((688,688),dtype="complex")

    for u in range(1,689):
        for v in range(1,689):
            den=math.pi*(u*a+v*b)
            #print(den)
            temp=math.sin(den)
            complexno=1j
            exponent=cmath.exp((-complexno)*den)
            num=T*temp*exponent
            #num=complex(numerator)
            H[u-1,v-1]=num/den
    final_image2=image_fft*H
    image2=np.fft.ifft2(final_image2)
    plt.imsave('2_degraded_image.png',image2.real, cmap='gray')
    return(image2)
```

Result

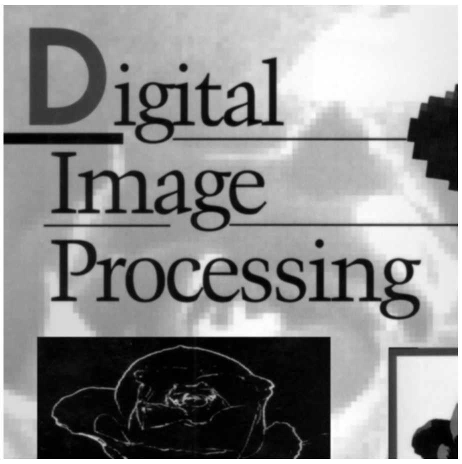

Figure 4: Original Image

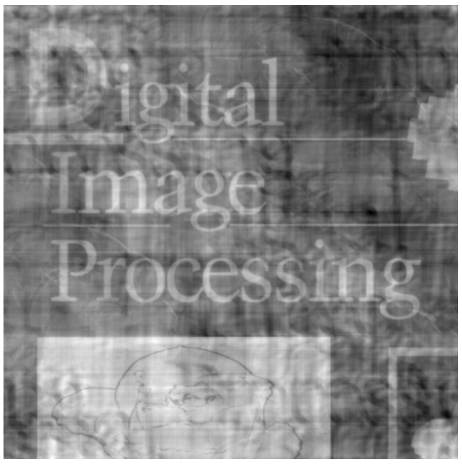

Figure 5: Degradation - Motion Blur

Inference

The image was degraded using the given formula. The degradation was a motion blur. Various levels of degradation was obtained by changing the parameters a,b and T.

3. Image Degradation

Perform image degradation using atmospheric turbulence (use Book.tif)
Steps:

- Compute Fourier transform of image
- Multiply the transform by H(u, v) given by:$H(u,v) = e^{-k(u2+v2)^{\frac{5}{6}}}$ where k = 0.0025(severe turbulence),k = 0.001(mild turbulence) and k = 0.00025(low turbulence)
- Take the inverse transform of resultant image to get the degraded image

Aim

To learn to apply degradadation techniques to the given image by an atmospheric turbulence degradation function

Theory/Discussion

Atmospheric turbulence affects the imaging system at a long distance, which causes time-varying blur. The degradation extent depends on the turbulence strength, which is characterized by the value of the Fried parameter.

Algorithm

1. Read an Image
2. Compute the Fourier Transform of the image
3. Form the H matrix using the given formula for the Atmospheric Turbulence
4. Convolve Fourier Transform of Image and H
5. Take inverse Fourier Transform of the modified image
6. Save the image

Code

```
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 26 14:49:39 2018

@author: IIST
"""
import numpy as np
import math
from PIL import Image
from scipy import fftpack as fftp
import imageio
import cmath
from matplotlib import pyplot as plt

img = Image.open('C:/Users/IIST/Downloads/Ragja/7_SC18M003_Ragja_IP2018/Book.tif')
img.load()
data = np.asarray( img, dtype="float64" )

image_fft=fftp.fft2(data)
a=0.1
k=0.00025
T=1
H=np.zeros((688,688),dtype="complex")

for u in range(1,689):
    for v in range(1,689):
        den=u**2+v**2
        exponent=k*(den**(5/6))
        H[u-1,v-1]=math.exp(-exponent)

final_image2=image_fft*H
```

```
image2=np.fft.ifft2(final_image2)
imgfinal = Image.fromarray((image2).astype('uint'))
#imgfinal.save('equalized_image.png')
plt.imsave('3_low_turbulence.png',image2.real, cmap='gray')
#img=imageio.imwrite('inverse_filtered.png',image2)
```

Result

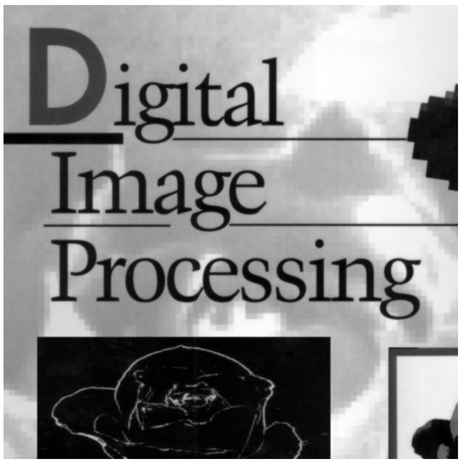

Figure 6: Original Image

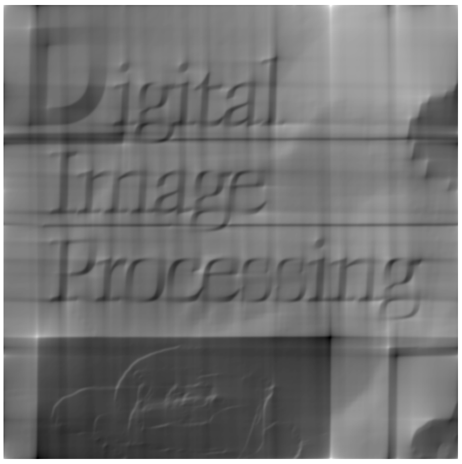

Figure 7: Degradation - Low Turbulence

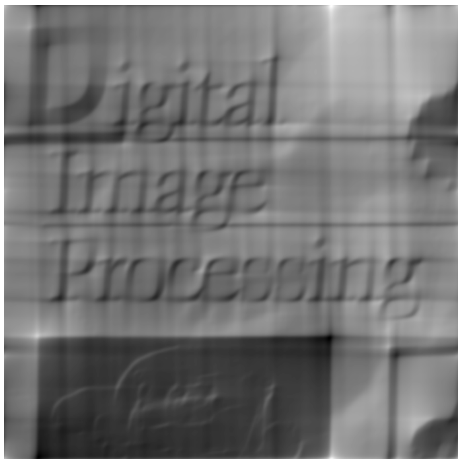

Figure 8: Degradation - Mild Turbulence

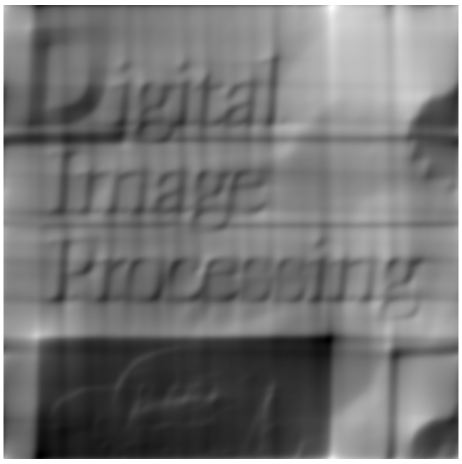

Figure 9: Degradation - Severe Turbulence

Inference

Turbulence Degradation was applied on the given image. Values of the turbulence factor was changed from low to mild to severe and the changes in the image were observed. The degradation increased with increase in severity of the turbulence factor k

4. Image Restoration

Perform image restoration on the motion blurred image obtained from Qn.2, using inverse filtering.

Steps:

Fourier Transform of restored image = $\frac{G(u,v)}{H(u,v)}$

where H(u, v) is degradation function from Qn.2 and G(u, v) is the fourier transform of degraded(input) image (Refer Gonzalez Page 351).

Aim

To restore a motion blurred degraded image using inverse filtering techniques

Theory/Discussion

The inverse filtering is a restoration technique for deconvolution, i.e., when the image is blurred by a known lowpass filter, it is possible to recover the image by inverse filtering or generalized inverse filtering. If we know of or can create a good model of the blurring function that corrupted an image, the quickest and easiest way to restore that is by inverse filtering. Unfortunately, since the inverse filter is a form of high pass filer, inverse filtering responds very badly to any noise that is present in the image because noise tends to be high frequency.

Algorithm

1. Read an Image
2. Call the image degradation function and obtain G and H
3. Obtain the original image by dividing G with H
4. Save the image

Code

```
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 26 18:43:20 2018

@author: IIST
"""

import numpy as np
import math
from PIL import Image
from scipy import fftpack as fftp
import imageio
import cmath
from matplotlib import pyplot as plt
from degradation import return_degraded_image
img = Image.open('C:/Users/IIST/Downloads/Ragja/7_SC18M003_Ragja_IP2018/2_degraded_image
img.load()
G = np.asarray( img, dtype="float64" )
image_array,H=return_degraded_image()
G=fftp.fft2(image_array)

F=np.zeros((688,688),dtype="complex")

#avg_filter_value=(H[0,0]+H[0,1]+H[0,1]+H[0,2]+H[2,0]+H[1,1]+H[1,2]+H[2,1]+H[2,2])/9
#H_=H_+avg_filter_value
F=np.divide(G,H)

image2=np.fft.ifft2(F)
plt.imsave('restored_image.png',image2.real, cmap='gray')

# -*- coding: utf-8 -*-
"""
Created on Wed Sep 26 14:49:39 2018

@author: IIST
"""
import numpy as np
import math
from PIL import Image
from scipy import fftpack as fftp
import imageio
import cmath
from matplotlib import pyplot as plt
def return_degraded_image():
    img = Image.open('C:/Users/IIST/Downloads/Ragja/7_SC18M003_Ragja_IP2018/Book.tif')
    img.load()
    data = np.asarray( img, dtype="float64" )

    image_fft=fftp.fft2(data)
    a=1
    b=1
    T=1
    H=np.zeros((688,688),dtype="complex")

    for u in range(1,689):
        for v in range(1,689):
            den=math.pi*(u*a+v*b)
            #print(den)
            temp=math.sin(den)
            complexno=1j
            exponent=cmath.exp((-complexno)*den)
            num=T*temp*exponent
            #num=complex(numerator)
            H[u-1,v-1]=num/den
    final_image2=image_fft*H
    image2=np.fft.ifft2(final_image2)
    plt.imsave('2_degraded_image.png',image2.real, cmap='gray')
    return(image2,H)
```

Result

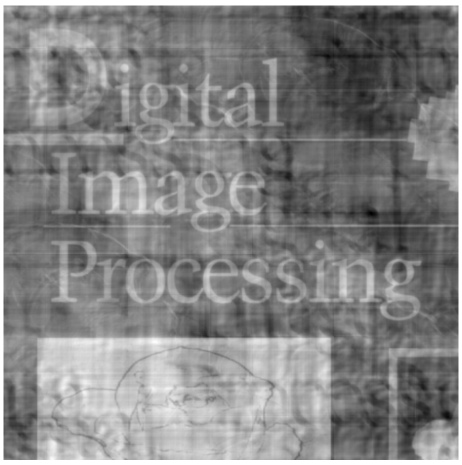
Figure 10: Degraded Image

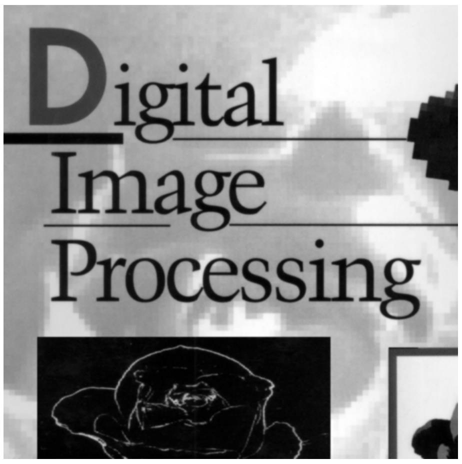

Figure 11: Restored Image

Inference

Inverse filtering was performed on the motion blurred image. For some values of parameters a and b, the restoration was full of stripes - a case of high values of $\frac{N(u,v)}{H(u,v)}$ dominating the $F(u,v)$ values. However, this issue was solved using only a few values of H close to/around the origin for performing the division operation to obtain F(u,v).