Labsheet 10
Image Restoration and Thresholding

Ragja Palakkadavath
(SC18M003)

Sub: Image and Video Processing Lab **Department**: Mathematics(MLC)
Date of Lab sheet: October 11, 2018 **Date of Submission**: October 16, 2018

1. Image Restoration using Constrained Matrix Inversion

Perform restoration of a degraded image using Constrained matrix inversion. Steps:

1. Select an image.
2. Degrade the image.
3. Select smoothing operator as Laplacian.
4. Choose the parameter gamma.(Experiment with different values of parameter lambda.)
5. Compute the mean grey value of the degraded image.
6. Compute the DFT of the degraded image.
7. Multiply the DFT of the degraded image with function Mcap(u, v) of equation (5.237) of Maria Petrou,point by point.
8. Take the inverse DFT of the result.
9. Add the mean grey value of the degraded image to all elements of the result, to obtain the restored image.

What are the other smoothing filters that can be used to perform the above method.Comment on the performance and see which gives the better result.
Refer Maria Petrou - Image Processing : Fundamentals (Pages starting from 464 for the procedure.)

Aim

To restore a linearly degraded image using the technique of constrained matrix inversion

Theory/Discussion

Constrained Matrix Inversion is a linear degradation restoration function. Since it is constrained method, Lagrangian Parameter are used to estimate the Least Square Error from the original Image.

$$\hat{M}(u,v) = \frac{1}{\hat{H}(u,v)} \times \frac{|\hat{H}(u,v)|^2}{|\hat{H}(u,v)|^2 + \gamma|\hat{L}(u,v)|^2}$$

Figure 1: Constrained Matrix Inversion Filter

Algorithm

1. Read an Image
2. Form the fourier transform of the Laplacian ($|\hat{L}(u,v)|^2$) using the equation to compute Laplacian
3. Choose the appropriate Lambda
4. Find the mean grey value of degraded image and also compute the degraded image's fourier transform
5. Form the $\hat{\text{M}}$(u,v) function using the Laplacian obtained in step2 and the degradation function applied on the image($\hat{\text{H}}$(u,v))
6. Multiply the degraded image and the inversion matrix filter
7. Find inverse fourier transform of the resulting matrix and save as image. Restored image is thus obtained

Code

```
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 16 18:12:11 2018

@author: IIST
"""
from PIL import Image
import numpy as np
import math
import matplotlib.pyplot as plt
from scipy import fftpack as fftp
from degradation import return_degraded_image

def get_mean_avg_val_imge():
    img = plt.imread('degraded.png')
    img=img.flatten()
    #plt.figure()
    cdf, bins, patch=plt.hist(img, bins=256, range=(0,256), normed=True, cumulative=Fals
    grey_levels=np.arange(0,256,1)
    #print(grey_levels)
    uL=np.dot(grey_levels ,cdf)
    theta=np.sum(cdf)
    u=uL/theta
    return u
def get_image():
    img = Image.open('Book.tif').convert('L')
    img.load()
    I = np.asarray( img, dtype="uint8" )
    data=I
    l_2 = np.zeros((688,688))
    N=688
    pi=math.pi
    for n in range(0,data.shape[1]):
        t1=1/N**4
        t2=(N-5)**2
        t3=2*(N-5)*math.cos(2*pi*n/N)
        t4=2*(N-5)*math.cos(2*pi*n*(N-1)/N)
        t5=2*math.cos((2*pi*(N-2)*n)/N)
```

```
            l_2 [0 ,n]=(t1*(t2+t3+t4+t5))
        for m in range(1,data.shape[0]):
            for n in range(0,data.shape[1]):
                l_2 [m,n]=(1/N**4)*(25+2-(10*math.cos(2*math.pi*n/N)-10*math.cos((2*math.pi*(
        gamma=10*30
        temp2=gamma*l_2
        g,H=return_degraded_image()
        u=get_mean_avg_val_imge()
        G=fftp.fft2(g)
        temp=abs(H)**2
        M_temp=temp/(temp+temp2)
        M=M_temp/H
        F=G*M
        f=fftp.ifft2(F)
        f=f.real
        f=f+u*(10**-31)
        plt.imsave('restored_book.png',f,cmap='gray')
get_image()
```

Result

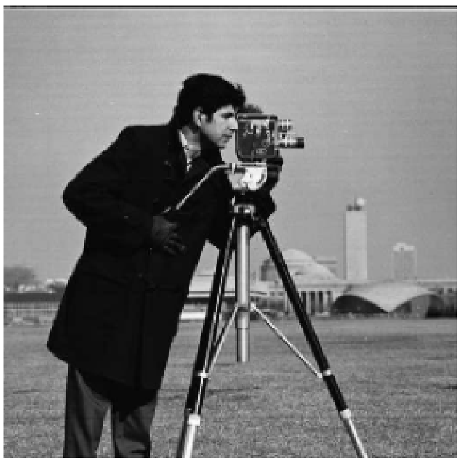

Figure 2: Original Cameraman

Figure 3: Degraded Cameraman

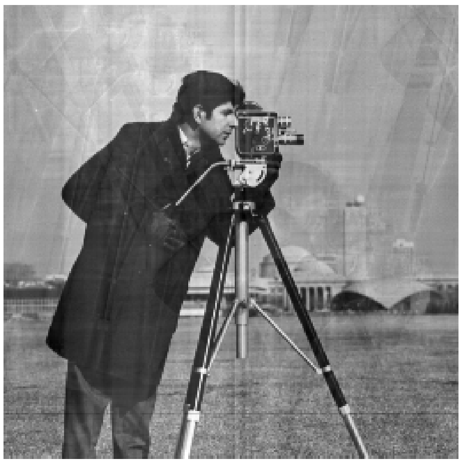
Figure 4: Restored Cameraman

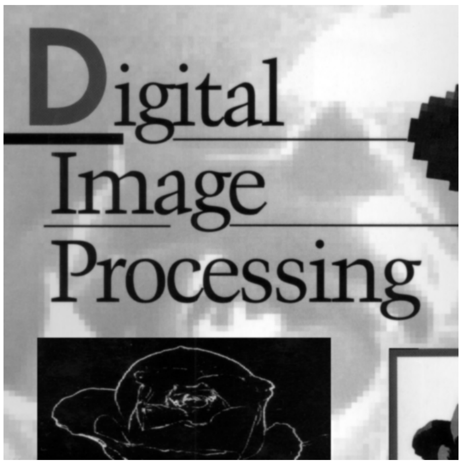

Figure 5: Original Book

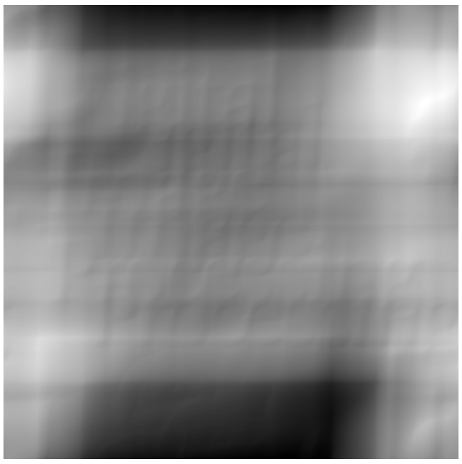
Figure 6: Degraded Book

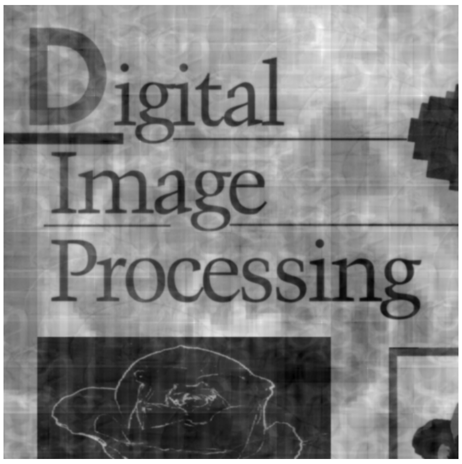

Figure 7: Restored Book

Inference

Restored image was obtained after multiple tries of providing various gamma values. Motion blur(linear degradation) on the image was removed and restored image was obtained using Constrained Matrix Inversion which is a linear, homogenenous filter.

2. Binary Thresholding

Perform binary thresholding of an image and obtain background and foreground.
Steps:

1. Select an image
2. Obtain the histogram of the image.
3. Find the threshold.
4. Assign 0 or 1 to each pixel according to the threshold

Aim

To perform binary thresholding of an image to seperate the background and foreground objects

Theory/Discussion

This image analysis technique is a type of image segmentation that isolates objects by converting grayscale images into binary images.

Algorithm

1. Read an Image
2. Obtain the histogram of the image
3. From the histogram, try to analyse the place where the different distributions meet so that the intensities can be seperated out
4. Set the distribution value as the threshold intensity
5. Any intensity below threshold is given 0 and any intensity above threshold is given 1 value
6. Save the image

Code

```
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 16 17:09:25 2018

@author: IIST
"""

from PIL import Image
import numpy as np
import math
import matplotlib.pyplot as plt
def get_image():
    img = Image.open('Book.tif').convert('L')
    data = np.asarray( img, dtype="float64" )
    new_image=np.zeros((data.shape[0],data.shape[1]))
    img.load()
    img = plt.imread('Book.tif')
    img=img.flatten()
    plt.figure()
    cdf, bins, patch=plt.hist(img, bins=255, range=(0,256), normed=True, cumulative=Fals
    for i in range(0,data.shape[0]):
        for j in range(0,data.shape[1]):
            if data[i][j]>=150 or data[i][j]<=50:
                new_image[i][j]=255
            else:
                new_image[i][j]=0
    #print(new_image)
    plt.imsave('book_bin_thresholding.png',new_image,cmap='gray')
    #plt.show()

get_image();
```

Result

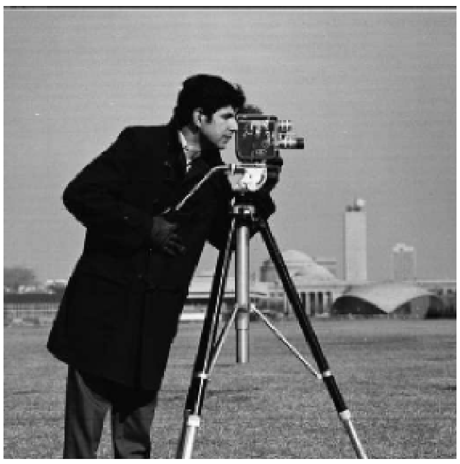
Figure 8: Original Cameraman

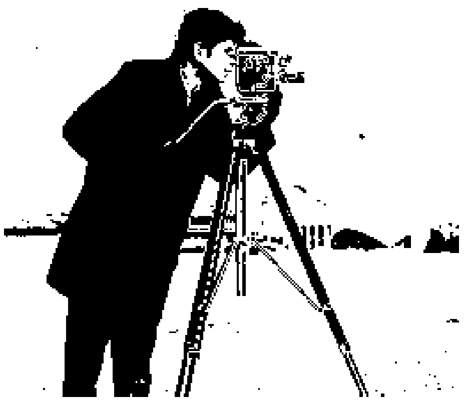
Figure 9: Cameraman - Binary Thresholding

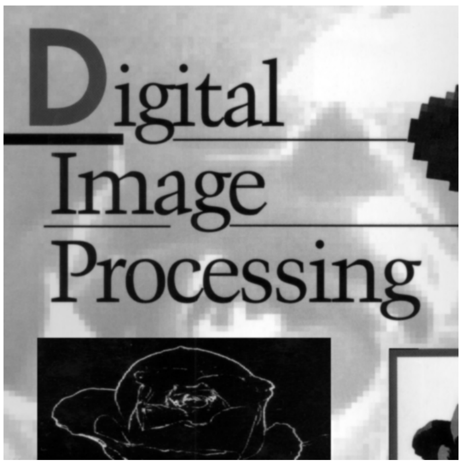

Figure 10: Original Book

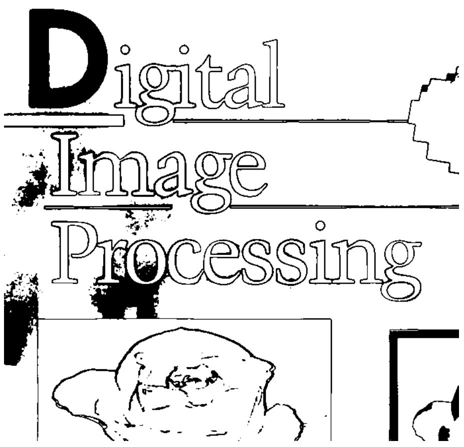
Figure 11: Book - Binary Thresholding

Inference

Binary Thresholding was performed on cameraman and book image. Final images were obtained where the background was seperated from the cameraman and similarly the writings in the book were seperated out from the background.

3. Otsu Thresholding

Write a program to convert a gray image to binary image using global thresholding (Otsus thresholding). *Refer Maria Petrou - Image Processing : Fundamentals (Page No: 541-545 for the procedure.)*

Aim

To convert a gray image to a binary image by finding the threshold value using Otsu's global thresholding algorithm

Theory/Discussion

Otsu's thresholding method involves iterating through all the possible threshold values and calculating a measure of spread for the pixel levels each side of the threshold, i.e. the pixels that either fall in foreground or background. The aim is to find the threshold value where the sum of foreground and background spreads is at its minimum.

Algorithm

```
1. Read an Image
2. Compute mean grey value of the image $(\mu) and (\theta) For each grey value compute, \sigma_B^2$(t) = $\frac{[\mu(t)-\mu\theta(t)]^2}{\theta(t)[1\theta(t)]}$
   using j(t) =$\sum x * p(x)$ and m(t)=$\sum p(x)$ formula
4. Whenever the $\sigma_B^2$(t) becomes lesser than the previous value, stop the iteration and use the
   current t value as threshold
5. Continue thresholding the image as per question 2 now
6. Save the image
```

Code

```
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 16 17:22:23 2018

@author: IIST
"""

from PIL import Image
import numpy as np
import math
import matplotlib.pyplot as plt
def get_image():
    img = Image.open('cameraman.tif').convert('L')
    data = np.asarray( img, dtype="float64" )
    new_image=np.zeros((data.shape[0],data.shape[1]))
    img.load()
    img = plt.imread('cameraman.tif')
    img=img.flatten()
    #plt.figure()
    cdf, bins, patch=plt.hist(img, bins=256, range=(0,256), normed=True, cumulative=Fals
    grey_levels=np.arange(0,256,1)
    #print(grey_levels)
    uL=np.dot(grey_levels,cdf)
    theta=np.sum(cdf)
    u=uL/theta
    sigma_sqr_b=np.NINF
    i=0
    diff=1
    while(diff>0 and i<256):
        print(i)
        ut=np.dot(grey_levels[:i],cdf[:i])
        thetat=np.sum(cdf[:i])
        num=(ut-u*thetat)**2
        den=thetat*(1-thetat)
        if(den==0):
            i=i+1
            continue
        sigma_sqr_b_new=num/den
        diff=sigma_sqr_b_new-sigma_sqr_b
        print(diff)
        sigma_sqr_b=sigma_sqr_b_new
        i=i+1
    for k in range(0,data.shape[0]):
        for j in range(0,data.shape[1]):
            if data[k][j]>=i-1:
                new_image[k][j]=255
            else:
                new_image[k][j]=0
    plt.imsave('cameraman_otsu_thresholding.png',new_image,cmap='gray')
get_image();
```

Result

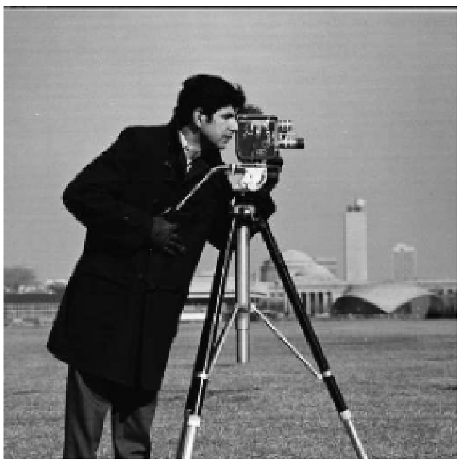

Figure 12: Original Cameraman

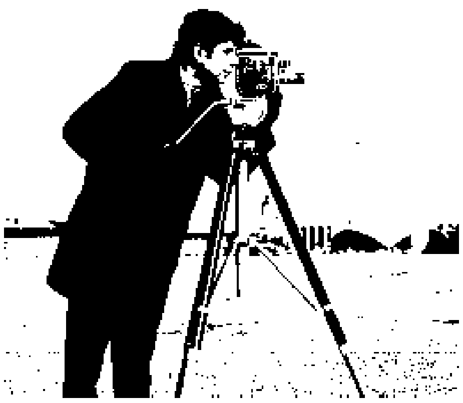

Figure 13: Cameraman - Otsu's Thresholding

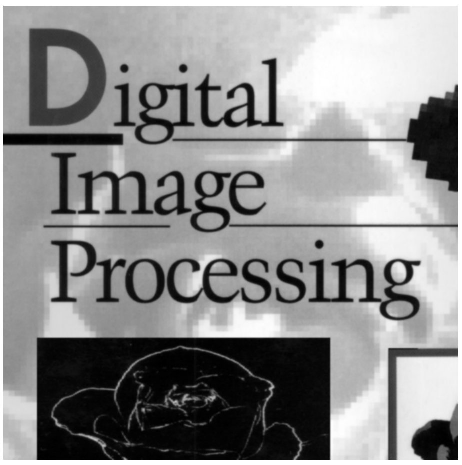

Figure 14: Original Book

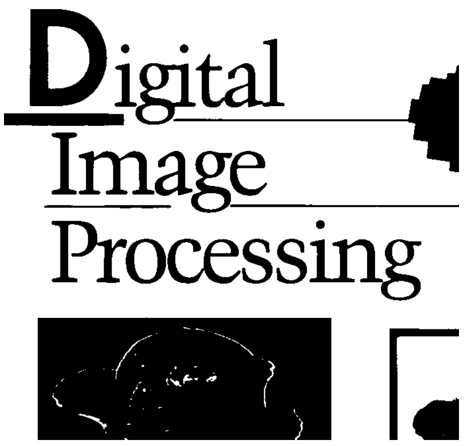

Figure 15: Book - Otsu's Thresholding

Inference

Thresholding was performed by finding the threshold value using Otsu's method. This made the task of finding the threshold automatic. And looking at the results, it gives a better binary segmentation of image than the usual binary thresholding.