SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

IMAGE AND VIDEO PROCESSING

COURSE PROJECT 1

Basic image processing

Author:
Jie YANG

Supervisor:
JIANHONG SHI

November 9, 2018

CONTENTS

Basic image processing

Introduction

These course project is compilation of former class projects. In this course project, we should do log transformation, gamma correction with different gamma value on three different PGM images. In second problem, we will generate a image which gray levels varying from 200 to 220 using gaussian distribution. After that insert a square in the range $[80, 100]$, finally remove the gray background. In third problem, we should write a function which can set filter size and weight coefficients to do spatial filtering. In forth problem, noised be added into image, and different size mean filter will be used to remove noise. In fifth problem, unsharp masking technique required to enhance three images. In last problem, IHPF, BHPF, GHPF will be implied on three images.

After finishing all these six problems, students could know how to do basic image processing like enhancement, filtering, transformation.

Method

Log transformation

Log transformation is a kind of method to increase image intensity. The general form of the log transformation is **Eq.1**. Where c is a constant, and it is assumed that $r \geq 0$. Before log transformation, image intensity should be normalized to $[0, 1]$, so that the formulation we used is **Eq.2**.

$$s = c * log(1 + r) \tag{1}$$

$$s = 255 * \frac{log(1 + r/255)}{log2} \tag{2}$$

Gamma correction

Gamma correction used to map image intensity nonlinearly by exponential function. The less γ **Eq.3** is the brighter image is, because it will increase the dark area intensity larger than brighter area. In this experiment, same to log transformation, we should normalize r to $[0, 1]$, so that the formulation used is **Eq.4**.

$$s = cr^{\gamma} \tag{3}$$

$$s = 255 * \left(\frac{r}{255}\right)^{\gamma} \tag{4}$$

Background removing

Base on the intensity difference of background and foreground, there is no crosstalk between them, so that we set a threshold (130) to distinguish which group the pixel should be classified. The algorithm is:

```
for pixel in image:
    if pixel >130:
        pixel=0
```

Custom mask

First use standard input function to read input value from keyboard. Then transfer input values to mask. After that select the same size of mask area (ROI) from image and do dot multiplication to calculate summation value. Finally use mean value to reset intensity and remove to next pixel. The algorithm is:

```
for pixel in image:
    get ROI
    S=sum(ROI*mask)
    pixel=S/length(mask)
```

Noise removing

At this session, first add random noise to image, we set the noise is in range [0, 100], then use the mask filter made in last section to do mean filtering by setting all wights coefficients to 1. The formulation is **Eq.5**

$$\hat{f}(x,y) = \frac{1}{|mn|} \sum_{(s,t)\in Np(m*n)} g(s,t) \tag{5}$$

Image unsharpen

A process that has been used for many years by the printing and publishing industry to sharpen images consists of subtracting an unsharp (smoothed) version of an image from the original image.This process, called unsharp masking, consists of the following steps:

1. Blur the original image.

2. Subtract the blurred image from the original (the resulting difference is called the *mask*.)
3. Add the mask to the original.

Letting $\bar{f}(x,y)$ denote the blurred image, unsharp masking is expressed in equation form as follows. First we obtain the mask in **Eq.6**.

$$g_{mask}(x,y) = f(x,y) - \bar{f}(x,y) \tag{6}$$

Then we add a weighted portion of the mask back to the original image in **Eq.7**.

$$g(x,y) = f(x,y) + g_{mask}(x,y) \tag{7}$$

High pass filters

Filter is the transfer function in frequency domain. Image can be transform to frequency domain, then multiply with the filter select what we want. For ideal high pass filter.**Eq.8** D_0 is the cut off frequency, it only pass frequency component which higher that D_0. In this report, we set $\frac{D_0}{D}$ as cut off frequency.

$$H(u,v) = \begin{cases} 0 & D(u,v) \le D_0 \\ 1 & D(u,v) > D_0 \end{cases} \tag{8}$$

Butterworth high pass filter (BHPF) is a wildly used low pass filter, when D_0 increase, it scut off frequency also increase, n is the order of it.**Eq.9** This polynomial filter can recede the ringing artifacts, for it has tails after cut off frequency. But as n increase, the ringing artifacts will increase.

$$H(u,v) = 1 - \frac{1}{1 + [\frac{D(u,v)}{D_0}]^{2n}} \tag{9}$$

Gaussian high pass filter (GHPF) is another common used low pass filter, it use Guassian function as transfer function, **Eq.10**, it can remove ringing artifacts perfectly in reconstruct image.

$$H(u,v) = 1 - e^{-D^2(u,v)/2D_0^2} \tag{10}$$

To do frequency filtering, we can multiply the filter to get what we want in frequency, **Eq.11** it is more easy to do in frequency domain. Because, in spatial domain, we should do convolution.

$$G(u,v) = H(u,v)F(u,v) \tag{11}$$

Results

Problem 1

The log transformation can increase image intensity. For lena and bridge image after log transformation, the background become brighter. It seem that the images become more clear. However, the log transformation increase background of fingerprint which decrease contrast let the fingerprint hard to distinguish. **Fig.1**

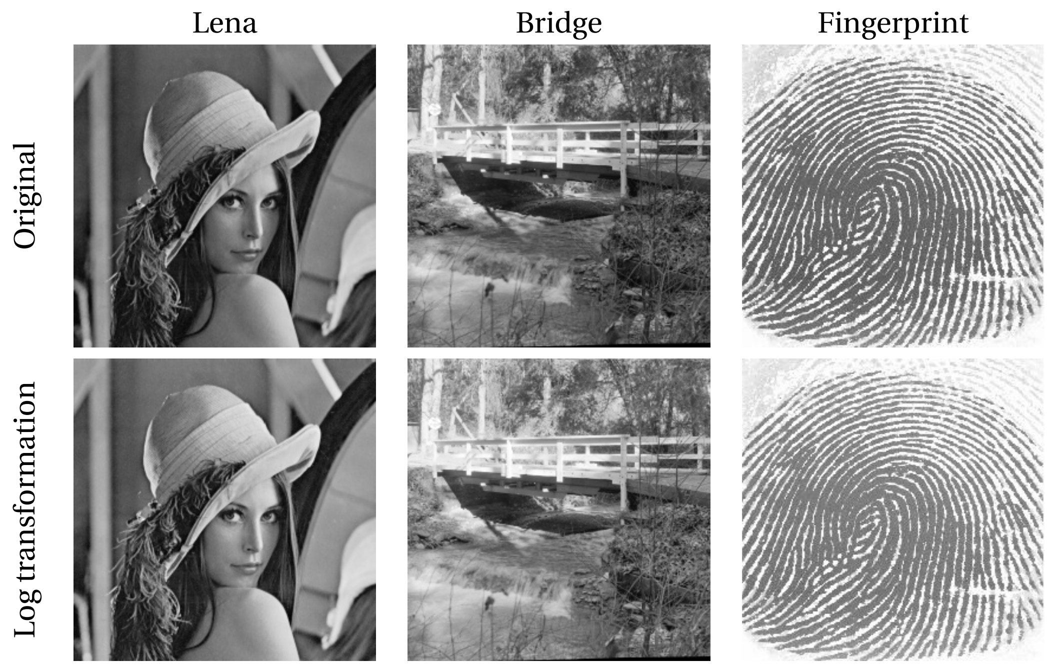
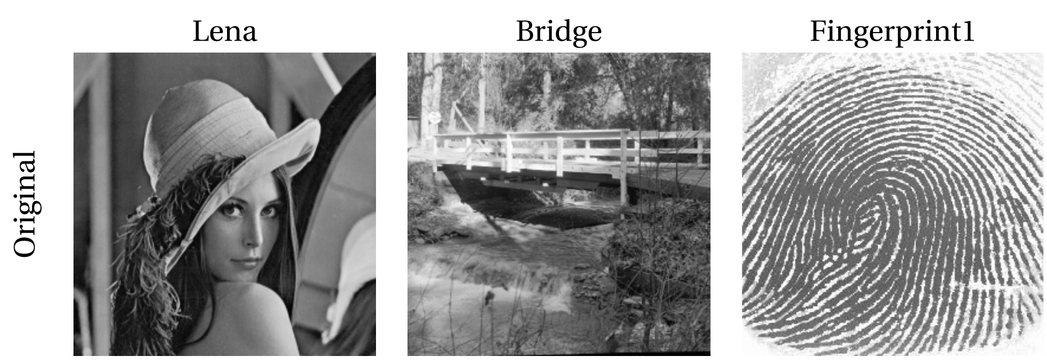


Figure 1: Log transformation

As the γ increase, the intensity of image gradually decrease, when $\gamma < 1$, gamma correct will increase lighter part while reverse to attenuate it when $\gamma > 1$. When $\gamma = 1$, the corrected image is same to original image. **Fig.2**

For Lena image, when $\gamma \leq 0.25$, the image be overexposure. The gamma correction also enhance background. For Bridge image, the best γ is 0.5, it increased dark part intensity so that plants near river are clear to see. For Fingerprint, when $\gamma = 2$, it can increase contrast of fingerprint, in this condition,the veins be darker compare to background. **Fig.2**



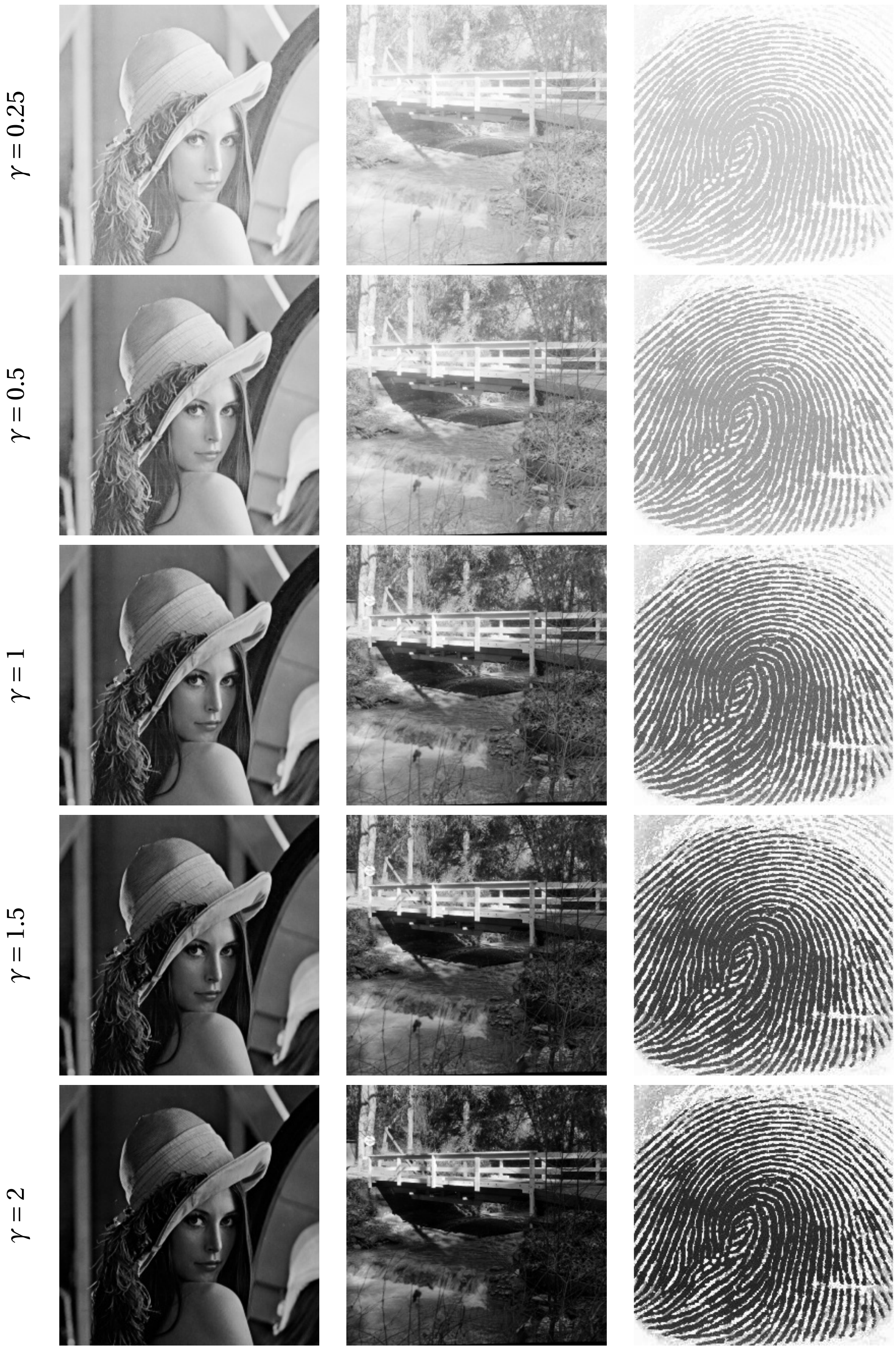
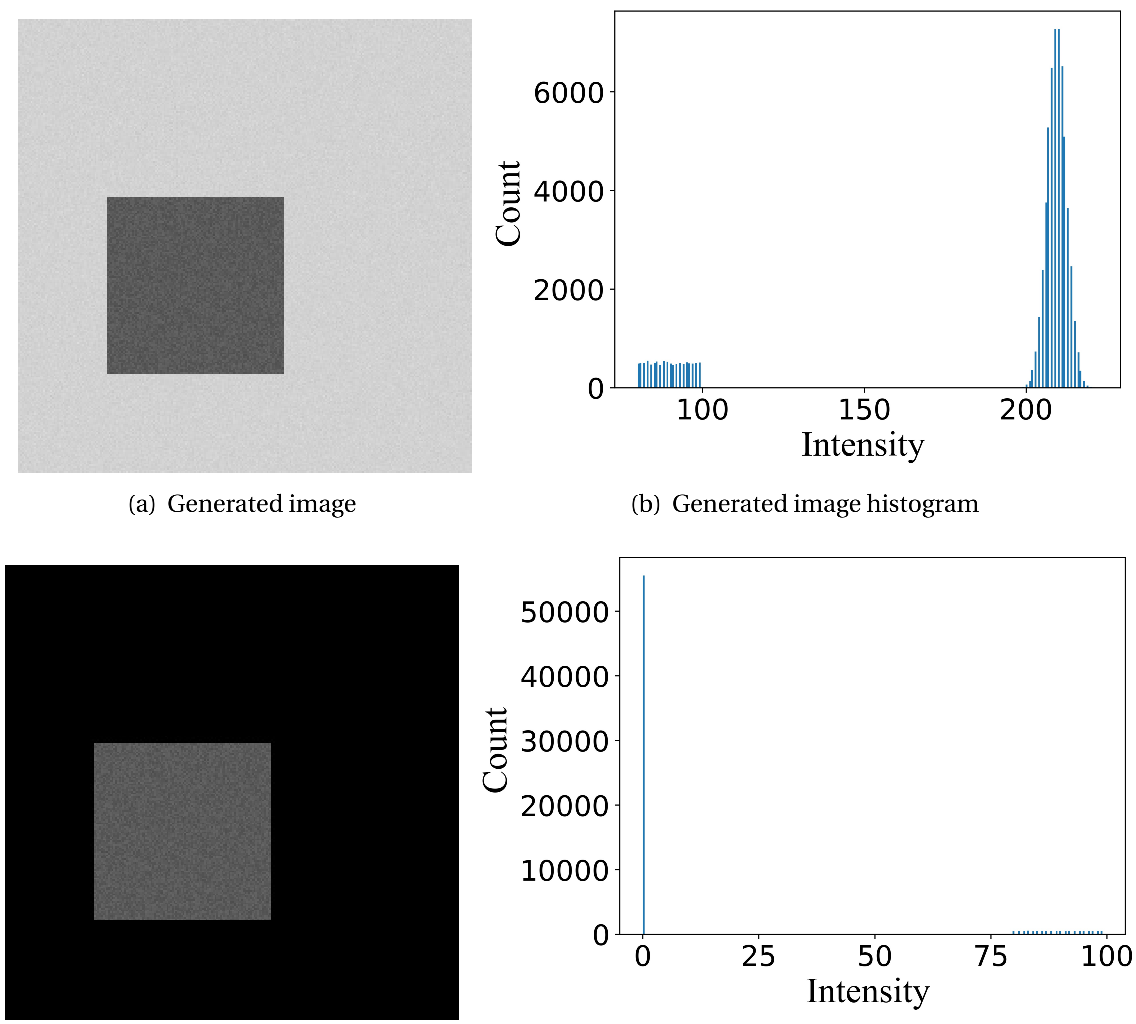


Figure 2: Gamma correction

Problem 2

From histogram **Fig.3-b**, it is clear to see that the distribution of square of background are separate very well which means that the threshold method in proper for this problem. The result image **Fig.3-c** proof that the background be removed clearly. Its histogram **Fig.3-d** all indicate all pixels of background are reset to 0 after processing.



(a) Generated image

(b) Generated image histogram

(c) Background removed

(d) Background removed histogram

Figure 3: Background removing

Problem 3

In this section, we set the mask size to 3×3 and let all weight coefficients to 1. This is mean filter. After this mean filter, the image become little blurry. **Fig.4-b**.Compare to opencvv build in mean filter **Fig.4-c**. They have same blur performance.

(a) Original (b) Custom mean filter (c) Opencv mean filter

Figure 4: Custom filter

Problem 4

After adding noise, there are many dots in image. The histogram also changed, seems to be a normal distribution. After filtering with 3 × 3 mean filter, the image be blurred, noise also decrease little. From the histogram, there is a little fluctuation. The intensity near to 50 be removed. After 5 × 5 mean filter. The image become more blurry with smoother noise. The histogram becomes more close to original histogram,. But still remove pixel near to 50. At the same time, pixel near to 0 increased compare with original image. **Fig.5**

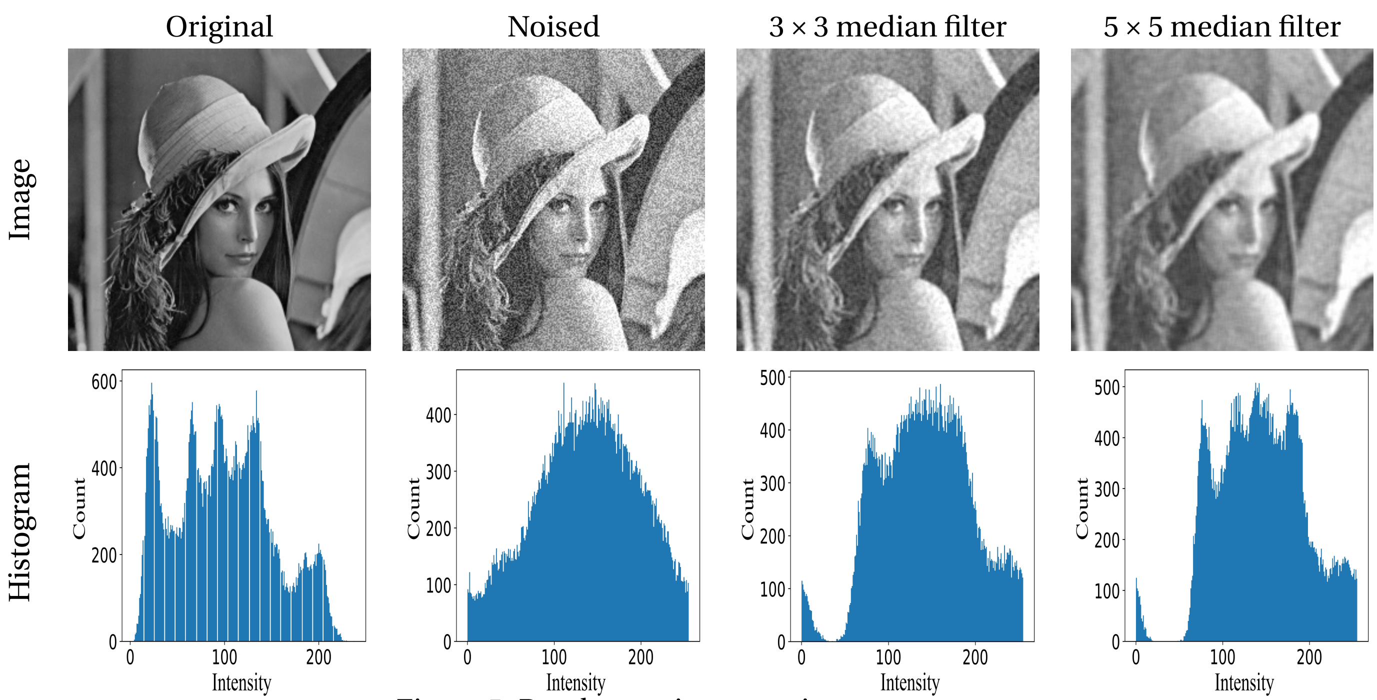


Figure 5: Random noise removing

Problem 5

In this section, we first use 3×3 Gaussian kernel to blur image, then subtract the mask image. It is clear to see that Mask image extract the edge or image. Finally add the mask image on the original image to get unsharp image. Compare to original image, unsharp image is more sharp. The edges becomes more clearly. **Fig.6**

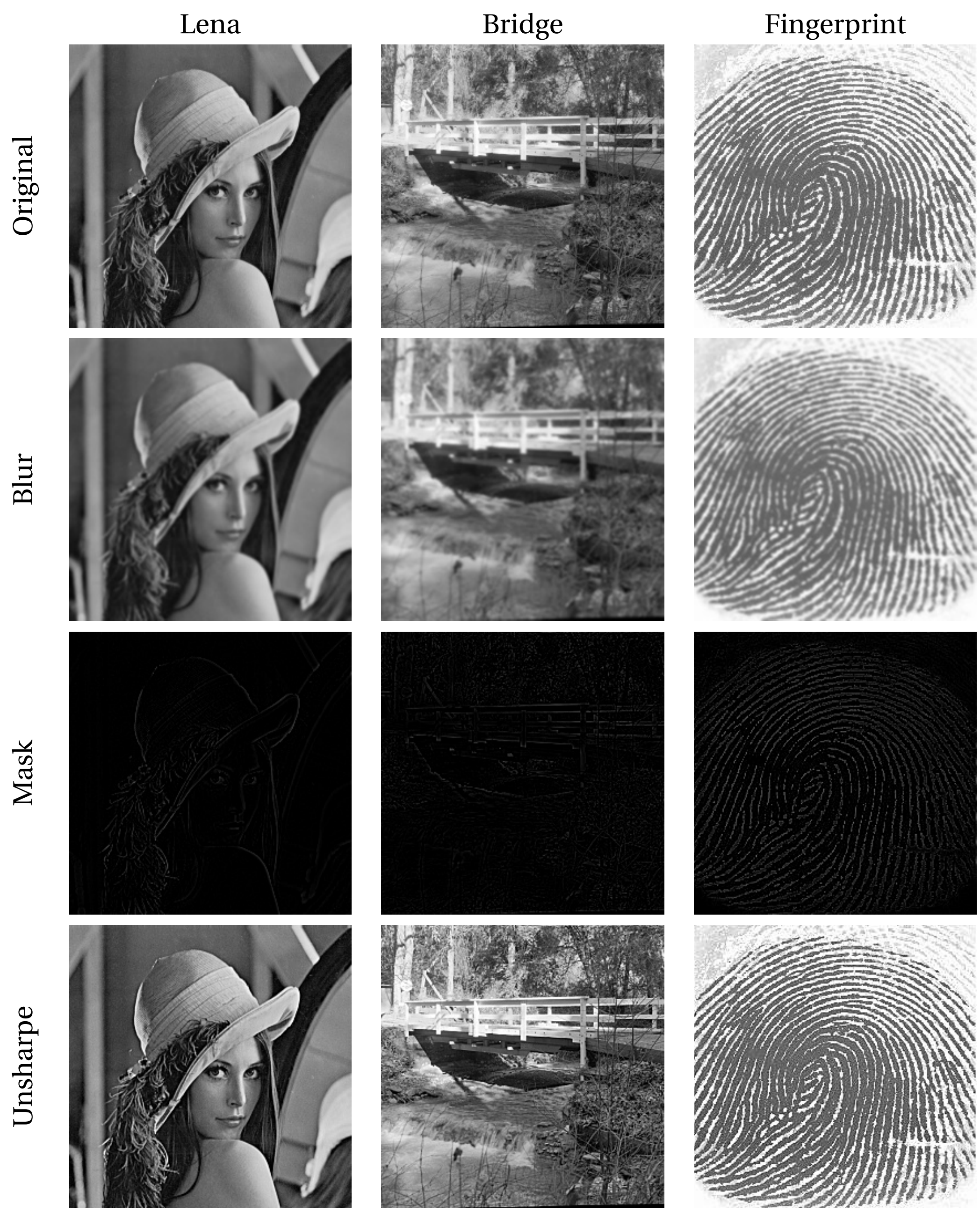


Figure 6: Image unsharpen

Problem 6

It is clear to see that high pass filter can extract edges in image. For IHPF, there are artifacts image. BHPFcan attenuate the artifacts and GHPF has no artifacts. The GHPF has best performance in edge extraction of all three images. **Fig.7**

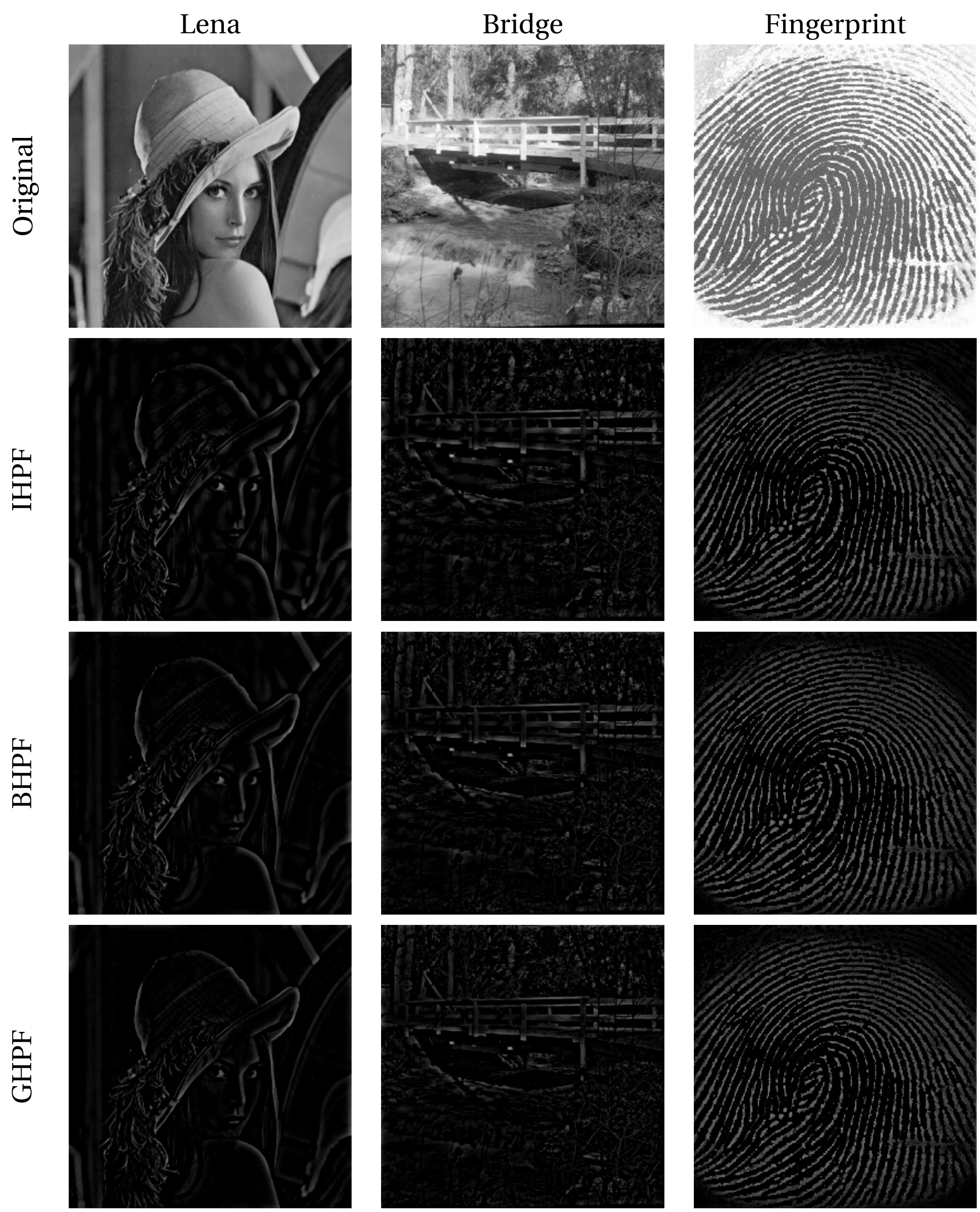


Figure 7: High pass filters

Discussion

In this project, we practiced several basic image processing methods. From whole project, it is useful to know that log transform and gamma correct can change image intensity nonlinear. So that can select increase local intensity of image. Filtering is useful in image processing, it can be used to remove noise but also cause image blur. If there are big difference of histogram between ROI and background. The simple method ia that set a threshold to separate them. High pass filter can extract image edge. GHPF will have best performance, for it has no artifacts.

There still is an interesting thing that after adding the noise. The histogram will change a lot. It is not the naive merge of image histogram and noise histogram. At this time the histogram is joint distribution of them.

Supplementary

This is the code used in this project.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
'''
 @File   : CourseProj1.py
 @Author: Yangjie
 @license : Copyright(C), SUSTech,Shenzhen,China
 @Contact : yangj3@mail.sustc.edu.cn
 @Date   : 2018/11/8
 @IDE : PyCharm
 @Desc   :
 Tis script developed for image and video processing course project 1

 Class:

 It define a IMG class to store several information of image, like ...
                                     image pathway,
 image name, image saved pathway, etc. IMG also has several functions...
                                      to load, save,
 plot histogram of image.


 Functions:
 sfft(img): fft(img), then shift to center ---> fshift

 isfft(fshift): shift fft img then do ifft ---> img_back

 cal_R(x,y,img): calculate distance from center fot each pixel  ---> ...
                                     R

    logTrans(img): do log transformation on img  --->  cimg

    GammaCorrect(img, gamma): do log transformation on img --->  cimg

```

```
 P1(imset): main function of Problem 1, imset is image name set

    GenIma(): generate image --->   img

 P2(imname): main function of Problem 1,imname is image name

    MaskFiltering(img, mask): filter img with mask --->  rimg

    inputmask(): generate mask base on input a  --->  np.array(mask),...
                                        a

 P3(imname): main function of Problem 3,imname is image name

    add_rand_noise(img): add random noise to img --->  Nimg

    meanf(img, m, n) do mean filtering on img use m* n filter ---> ...
                                        rimg

    P4(imname): main function of Problem 3,imname is image name
P5(imnameset): main function of Problem 1, imnameset is image name ...
                                        set
    it first blur image then minus original image to get mask and
    add mask to original image to unsharpen image

     IHPF(d, img): generate IHPF ---> H, R

     GHPF(d, img): generate GHPF ---> H, R

     BHPF(d, img): generate BHPF ---> H, R

     HP(img, d, n, filter_type): do high pass filter on image base on
     filter type and cut off frequency d, which it ratio of image ...
                                        size,
     and calculate cut off frequency D --->  rimg.real, D

 P6(imset): main function of Problem 6, imset is image name set

 Main functions are excude when  __name__=='__main__'

'''


import cv2
import numpy as np
from matplotlib import pyplot as plt


class IMG:
    def __init__(self, name, mark=None):
        self.path = 'D:\graduated\Image_process\lab\PGM_images\\'
        self.savepath = 'D:\graduated\Image_process\lab\lab_report\...
```

```
                                      course_project_1\imagesave\\'
        self.name = name
        self.prop = '.pgm'
        self.mark = mark
        # self.img=None

    def load(self):
        self.imapath = self.path + self.name + self.prop
        self.img = np.float64(cv2.imread(self.imapath, 0))
        self.save(self.img, 'original')
        return self.img

    def save(self, img, mark=None, flag=0):
        if flag:
            img = cv2.equalizeHist(np.uint8(img))
        self.mark = mark
        savepath = self.savepath + self.name + '_' + self.mark + '...
                                      jpg'
        cv2.imwrite(savepath, img)
        return img

    def disp(self, winName, img, sizeflag=cv2.WINDOW_NORMAL):

        img = cv2.equalizeHist(np.uint8(img))
        if sizeflag == 1:
            sizeflag = cv2.WINDOW_AUTOSIZE
        cv2.namedWindow(winName, sizeflag)
        cv2.imshow(winName, img)
        cv2.waitKey(0)
        cv2.destroyWindow(winName)
        return img

    def psave(self, img, mark=None, cb=0):  # shown image in windows ...
                                      and save
        fig = plt.gcf()
        plt.imshow(img, cmap='gray')
        if cb:
            plt.colorbar()
        plt.xticks([]), plt.yticks([])
        savepath = self.savepath + self.name + '_' + mark + '.jpg'
        fig.savefig(savepath, dpi=500, bbox_inches='tight')
        plt.close()

    def fsave(self, fig, mark=None):  # save plot fihiure
        plt.tick_params(labelsize=20)
        # plt.xticks([]), plt.yticks([])
        savepath = self.savepath + self.name + '_' + mark + '.jpg'
        fig.savefig(savepath, dpi=500, bbox_inches='tight')
        plt.close()
    def plthist(self, img,mark):
        font2 = {'family' :'Times New Roman', 'weight' : 'normal','...
```

```
                                             size'   :25}
        img=np.uint8(img)
        fig = plt . gcf ()
        plt . hist (  img . ravel () ,256 );
        plt . xlabel ( 'Intensity ',font2)
        plt . ylabel ( 'Count ',font2)
        self.fsave (fig , mark)
        plt . close ()


def sfft(img):
    f = np.fft.fft2(img)
    fshift = np.fft.fftshift(f)
    return fshift


def isfft(fshift):
    f_ishift = np.fft.ifftshift(fshift)
    img_back = np.fft.ifft2(f_ishift)
    return img_back




def cal_R(x,y,img):
    N=img.shape[0]
    M=img.shape[1]
    u=x-M/2
    v=N/2-y
    R=np.sqrt(u**2+v**2)
    return R


##---------p1------------------------

def logTrans(img):
    cimg = 255 * np.log(img / 255 + 1) / np.log(2)
    return cimg


def GammaCorrect(img, gamma):
    cimg = 255 * np.power(img / 255, gamma)
    return cimg


def P1(imset):
    for imname in imset:
        I = IMG(imname)  # 'cameraWithNoise' 'LenaWithNoise'
        img = I.load()
        cimg = logTrans(img)
        I.save(cimg, 'logtrans');
```

```
        for gamma in [0.25, 0.5, 1, 1.5, 2]:
            cimg = GammaCorrect(img, gamma)
            I.save(cimg, 'gamma_' + str(int(gamma * 100)))


##---------p1-----------------------


##-------p2-------------------------
def GenIma():
    img = np.uint8(3 * np.random.randn(256, 256) + 210)
    sqaure = np.random.randint(80, 100, (100, 100))
    img[100:200, 50:150] = sqaure
    return img


def P2(imname):
    I = IMG(imname)
    img = GenIma()
    I.save(img, 'original')
    I.plthist(img, mark='original_hist')

    for x in range(img.shape[0]):
        for y in range(img.shape[1]):
            if img[x, y] > 110:
                img[x, y] = 0

    I.save(img, 'select')
    I.plthist(img, mark='select_hist')


##-------p2-------------------------


## ----------------------------p3--------------------
def MaskFiltering(img, mask):
    m, n = mask.shape[:2]
    rimg = np.zeros((int(img.shape[0] - m + 1), int(img.shape[1] - n ...
                                                 + 1)))
    for x in range(rimg.shape[0]):
        for y in range(rimg.shape[1]):
            ROI = img[x:x + m, y:y + n]
            val = ROI * mask
            rimg[x, y] = np.mean(val)
    return rimg


def inputmask():
    a = input('please input mask, \n split with , for items \t\t ; ...
                                                 for rows \n')
    row = a.split(';')
```

```
    col = list(map(lambda x: x.split(','), row))
    mask = []
    for row in col:
        mask.append(list(map(float, row)))
    print('mask is: \n', mask)
    return np.array(mask), a


def P3(imname):
    I = IMG(imname)  # 'cameraWithNoise' 'LenaWithNoise'
    img = I.load()
    mask, strmask = inputmask()
    rimg = MaskFiltering(img, mask)
    I.save(rimg, mark='mymean')
    cvimg = cv2.blur(np.uint8(img), mask.shape)
    I.save(cvimg, mark='opencvmean');


##-----------------------p3----------------------


##-----------------------------  P4--------------------------------

def add_rand_noise(img):
    noise = np.random.randint(0, 100, (img.shape))
    Nimg = img + noise
    return Nimg


def meanf(img, m, n):
    rimg = np.zeros((int(img.shape[0] - m + 1), int(img.shape[1] - n ...
                                               + 1)))
    for x in range(rimg.shape[0]):
        for y in range(rimg.shape[1]):
            sum = 0;
            for M in range(m):
                for N in range(n):
                    sum += img[x + M, y + N]
            val = sum / (m * n)
            rimg[x, y] = val
    return rimg


def P4(imname):
    I = IMG(imname)  # 'cameraWithNoise' 'LenaWithNoise'
    img = I.load()
    I.plthist(img, 'original_hist')
    Nimg = add_rand_noise(img)
    I.save(Nimg, mark='noised')
    I.plthist(Nimg, mark='noised_hist')
    for mfsize in [3, 5]:
```

```
        fimg = meanf(Nimg, mfsize, mfsize)
        I.save(fimg, mark='denoised_' + str(mfsize) + 'x' + str(...
                                        mfsize))
        I.plthist(fimg, mark='denoised_'+ str(mfsize) + 'x' + str(...
                                        mfsize)+'_hist')


##-------------------- p4-----------------


##-------------------- p5---------------
def P5(imnameset):
    for imname in imnameset:
        I = IMG(imname)  # 'cameraWithNoise' 'LenaWithNoise'
        img = I.load()
        blur = cv2.GaussianBlur(img, (5, 5), 0)
        I.save(blur, 'blur');
        mask = img - blur
        I.save(mask, 'mask');
        unsharp = mask + img
        I.save(unsharp, 'unsharpe');


##-------------------- p5-------------


##------------------------------  p6--------------------------

def IHPF(d, img):
    R = np.around(d * img.shape[1] / 2)
    H = np.ones(img.shape)
    for y in range(img.shape[0]):
        for x in range(img.shape[1]):
            r = cal_R(x, y, img)
            if r < R:
                H[y, x] = 0
    return H, R


def GHPF(d, img):
    R = np.around(d * img.shape[1] / 2)
    H = np.zeros(img.shape)
    for y in range(img.shape[0]):
        for x in range(img.shape[1]):
            r = cal_R(x, y, img)
            a = 0.5 * (r / R) ** 2
            H[y, x] = 1 - 1 / np.exp(a)
    return H, R


def BHPF(d, n, img):
```

```
    R = np.around(d * img.shape[1] / 2)
    H = np.zeros(img.shape)
    for y in range(img.shape[0]):
        for x in range(img.shape[1]):
            r = cal_R(x, y, img)
            H[y, x] = 1 - 1 / (1 + (r / R) ** (2 * n))

    return H, R


def HP(img, d, n, filter_type):
    fimg = sfft(img)

    if filter_type == 'IHPF':
        H, D = IHPF(d, img)
    if filter_type == 'GHPF':
        H, D = GHPF(d, img)
    if filter_type == 'BHPF':
        H, D = BHPF(d, n, img)
    Fimg = fimg * H
    rimg = isfft(Fimg)
    return rimg.real, D


def P6(imset):
    filter_type_set = ['IHPF', 'GHPF', 'BHPF']
    d = 0.1
    n = 2
    for imname in imset:
        for typef in filter_type_set:
            I = IMG(imname)  # 'cameraWithNoise' 'LenaWithNoise'
            img = I.load()
            HFimg, D = HP(img, d, n, typef)
            I.save(HFimg, mark=typef + '_D0_' + str(int(D)))

# ---------------  p6-------------------------------------


if __name__=='__main__':
    print('-------------problem 1--------------')
    P1( ['lena', 'bridge', 'circles','fingerprint1'])


    print('-------------problem 2--------------')
    P2('Genimg')


    print('-------------problem 3--------------')
    P3('lena')


```

```
    print('-------------problem 4--------------')
    P4('lena')


    print('-------------problem 5--------------')
    P5( ['lena', 'bridge', 'circles','fingerprint1'])


    print('-------------problem 6--------------')
    P6( ['lena', 'bridge', 'circles','fingerprint1'])


    print('-------------All done--------------')
```

Code Listing 1: Python code for image processing