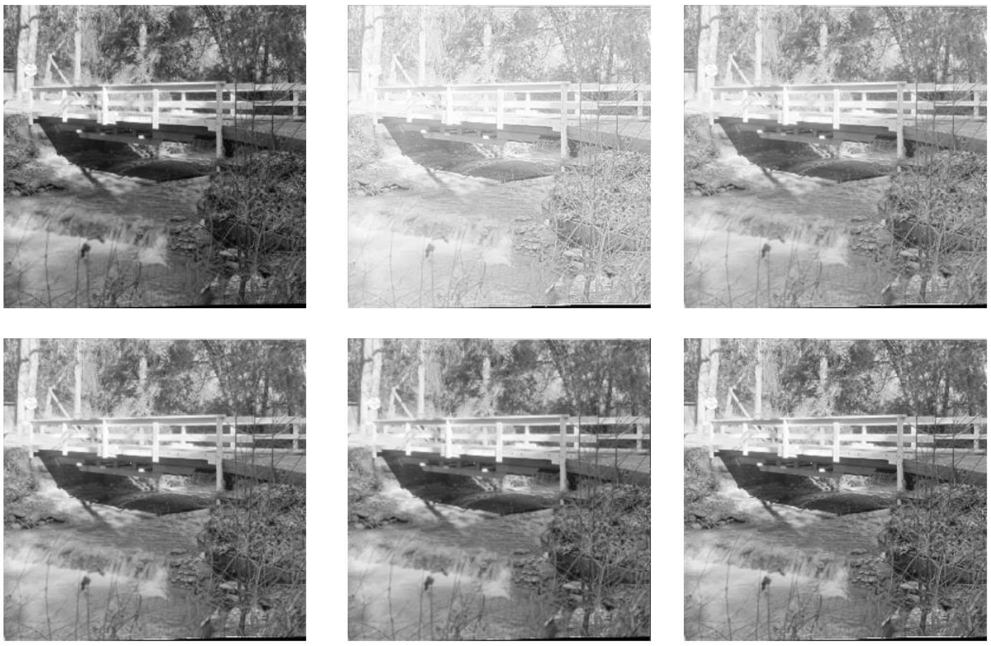
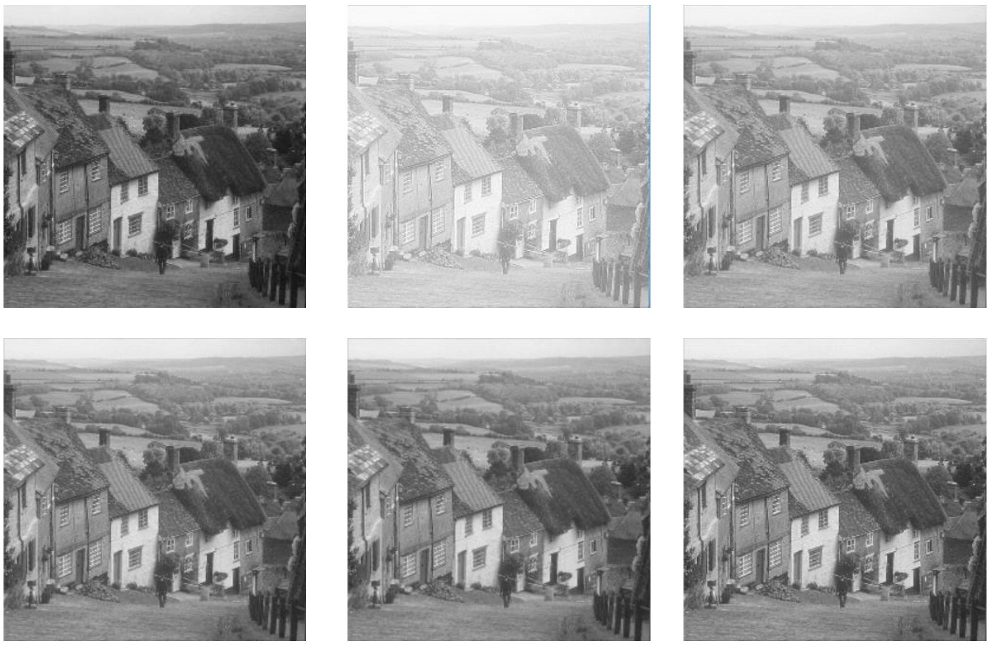
谭树杰 11849060

Lab6 Report

(1) Use a homomorphic filter to enhance bridge.pgm and goldhill.pgm filter

The picture shown below is the bridge.pgm and goldhill.pgm and their processed image with $(\gamma_{\mathbf{h}}, \gamma_l)$ = (2,0.25),(2,0.5),(2,0.75),(1.5， 0.75), (3,0.75) respectively. (c = 1 and D0 = 200)





Analysis:

If the parameters $\gamma_{\mathbf{h}} > 1 \; and \; \gamma_l < 1$, the filter function tends to attenuate the contribution made by the low frequencies(illumination) and amplify the contribution made by high frequencies(reflectance). As we can see in above images, we get simultaneous dynamic range compression and contrast enhancement of the image when we choose appropriate parametes.

The implementation by C++ and opencv is shown below:

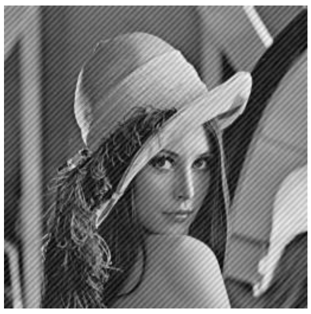
```
#include <opencv2/opencv.hpp>
#include <iostream>
#include<cmath>
using namespace cv;
using namespace std;
cv::Mat homomophic_filter(cv::Mat src, double gamma_h, double gamma_l, double c, double
D0);
int main(){
    cv::Mat srcImage = imread("..\\images\\goldhill.pgm", CV_LOAD_IMAGE_GRAYSCALE);
    if (!srcImage.data)
        return -1;
    cv::imshow("bridge", srcImage);
    cv::Mat dstImage = homomophic_filter(srcImage, 3, 0.75, 1, 200);
    cv::imshow("goldhill.jpg", dstImage);
    cv::waitKey(0);
    return 0;
}
Mat gaussian_filter(float P, float Q, float c, float D0){
    Mat GMat(P, Q, CV_64F); // Gaussian filter matrix
    double distSqure;
    for (int u = 0; u < P; u++){
        for (int v = 0; v < Q; v++){
            distSqure = (u - P / 2) * (u - P / 2) + (v - Q / 2) * (v - Q / 2);
            GMat.ptr<double>(u)[v] = 1 - exp(-c * distSqure / (D0 * D0));
        }
    }
    int cx = P / 2;
    int cy = Q / 2;
    Mat tmp;
    Mat q0(GMat, Rect(0, 0, cx, cy));
    Mat q1(GMat, Rect(cx, 0, cx, cy));
    Mat q2(GMat, Rect(0, cy, cx, cy));
    Mat q3(GMat, Rect(cx, cy, cx, cy));
    q0.copyTo(tmp);
    q3.copyTo(q0);
    tmp.copyTo(q3);
    q1.copyTo(tmp);
    q2.copyTo(q1);
    tmp.copyTo(q2);
    return GMat;
}
```

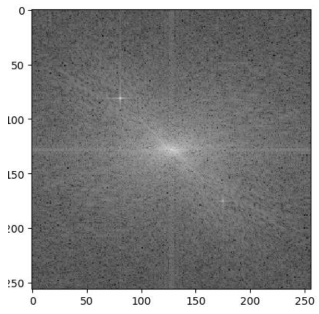
```
cv::Mat homomophic_filter(cv::Mat src, double gamma_h, double gamma_l, double c, double
D0){
    int P = src.rows;
    int Q = src.cols;
    src.convertTo(src, CV_64F, 1.0/ 255);
    src += 0.000001;  // Scalar::all(1)
    log(src, src);

    Mat GMat = gaussian_filter(P, Q, c, D0);
    // calculate dft
    cv::Mat planes[] = { cv::Mat_<double>(src),
        cv::Mat::zeros(src.size(), CV_64F) };
    Mat completeI;
    merge(planes, 2, completeI);
    dft(completeI, completeI);
    for (int row = 0; row < P; row++){
        for (int col = 0; col < Q; col++){
            completeI.at<Vec2d>(row, col)[0] = ((gamma_h - gamma_l) *
GMat.ptr<double>(row)[col] + gamma_l) * completeI.at<Vec2d>(row, col)[0];
            completeI.at<Vec2d>(row, col)[1] = ((gamma_h - gamma_l) *
GMat.ptr<double>(row)[col] + gamma_l) * completeI.at<Vec2d>(row, col)[1];
        }
    }
    // inverse discrete fourier transform
    dft(completeI, completeI, DFT_SCALE | DFT_INVERSE);
    split(completeI, planes);
    Mat dst = planes[0];
    exp(dst, dst);
    dst -= 0.000001;
    return dst;
}
```

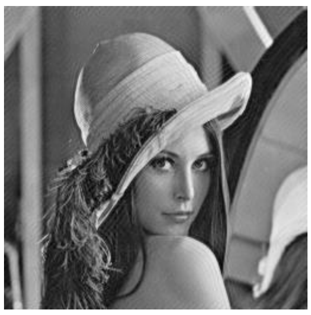
(2) Add sinusoidal noise to lena.pgm and use bandreject filter to recover lena.pgm

The lena.pgm, lena.jpg with sinusoidal noise and its magnitude spectrum,recovered image using bandreject filter are shown below respectively.

Analysis: We can observe that there a conjugate pair of strong signals in the spectrum, which represent the noise. We can calculate the frequency of the noise by $\boldsymbol{D0} = (\mathbf{256} - \mathbf{175}) * \sqrt{\mathbf{2}} \approx 115$, so I set highcut = 120, lowcut = 110 in bandreject filter. The image was improved significantly after filtering. There are a little distortion in the result image, for that we filtered some frequencies of the origin lena.pgm image.







The implementation by python is shown below:

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

def noisy(image):
    if len(img.shape) is not 2:
        raise Exception('Improper image')
    row, col = image.shape
    out = np.copy(image)
    for i in range(row):
        for j in range(col):
            out[i,j] = out[i,j]  + 20 * np.sin(20 * i + 20 * j)
    return out

def bandreject_filter(img, lowcut, highcut):
    u = int(img.shape[0] / 2)
    v = int(img.shape[1] / 2)
    out = np.empty_like(img, dtype=float)
    for row in range(img.shape[0]):
        for col in range(img.shape[1]):
            duv = (row - u)**2 + (col - v) ** 2
            if (duv >= lowcut ** 2) and (duv <= highcut ** 2):
                out[row, col] = 0
            else:
                out[row, col] = 1
    return out

def apply_filter(img):
    if len(img.shape) is not 2:
        raise Exception('Improper image')
    img_fft = np.fft.fft2(img)
    img_fft_fftshift = np.fft.fftshift(img_fft)

    # calculate magnitude
    mag = np.abs(img_fft_fftshift)
    maxi = maxj = max = 0
    for i in range(150, 190):
        for j in range(150, 190):
            if mag[i, j] > max:
                maxi, maxj = (i, j)
                max = mag[i, j]
    print("maxi "+ str(maxi) + " maxj " + str(maxj) + " is " + str(max))
    mag = 20 * np.log(mag)
    plt.imshow(mag, 'gray')
    h = bandreject_filter(img, 110, 120)
    img_fft_filtered = img_fft * h

    img_fft_filtered_unshift = np.fft.fftshift(img_fft_filtered)
    img_filt = np.fft.ifft2(img_fft_filtered_unshift)
    dst = np.abs(img_filt)
    cv2.normalize(dst, dst, 0, 1, cv2.NORM_MINMAX)
    return dst

if __name__ == "__main__":
    # import cv2
    path_in = './in/'
    path_out = './out/'
    img_path = 'lena.jpg'

    img_path_in = path_in + img_path
    img_path_out = path_out + 'lena_band.jpg'

    # Main code
    img = cv2.imread(img_path_in, cv2.IMREAD_GRAYSCALE)
    img = img.astype(float)
    img_sinusoidal =noisy(img)
    cv2.normalize(img_sinusoidal, img_sinusoidal, 0, 1, cv2.NORM_MINMAX)
    cv2.imshow("with noise", img_sinusoidal)
    dst = apply_filter(img_sinusoidal)

    cv2.imshow("dst", dst)
    cv2.imwrite(img_path_out, dst)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

(3) Use your cell phone to take three images : one image of your face as template, one large image with your face, and another large image without your face. Use correlation and the template to determine which

of the later two images contains your face (note, the face template may not be same as the one in the image)

The template image, image without my face, image with my face are shown below:

Analysis: We calculated that max correlation of face.jpg and template.jpg is 266244065.0 and max correlation of noface.jpg and template.jpg is266244065.0, so face.jpg has my face, as desired.

The implementation by python is shown below:

```
import numpy as np
import cv2

def correlation(m_img, t_img):
        return np.sum(m_img * t_img)

def matching(main_img, template_img):
        max_correlation = 0
        h, w = template_img.shape
        for x in range(main_img.shape[0]-h):
                for y in range(main_img.shape[1]-w):
                        if max_correlation < correlation(main_img[x:x+h, y:y+w], template_img):
                                max_correlation = correlation(main_img[x:x+h, y:y+w], template_img)
        return max_correlation

if __name__ == "__main__":
        # import cv2
        path_in = './in/'
        path_out = './out/'

        # Main code
        template_img = cv2.imread(path_in + "template.jpg", cv2.IMREAD_GRAYSCALE)
        template_img = template_img.astype(float)

        face = cv2.imread(path_in + "face.jpg", cv2.IMREAD_GRAYSCALE)
        face = face.astype(float)
        noface = cv2.imread(path_in + "noface.jpg", cv2.IMREAD_GRAYSCALE)
        noface = noface.astype(float)

        cor_face = matching(face, template_img)
        cor_noface = matching(noface, template_img)
        print("max correlation of face.jpg and template.jpg is " + str(cor_face))
        print("max correlation of noface.jpg and template.jpg is " + str(cor_noface))
        if (cor_face > cor_noface):
                print("face.jpg has my face")
        else:
                print("noface.jpg has my face")
```