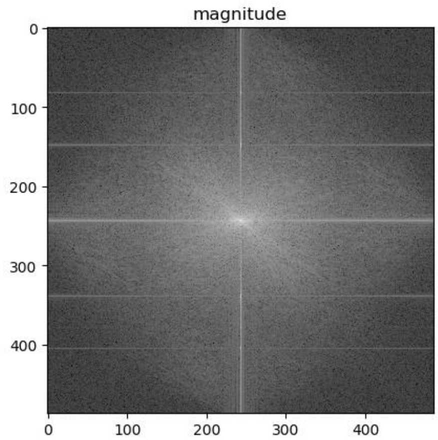
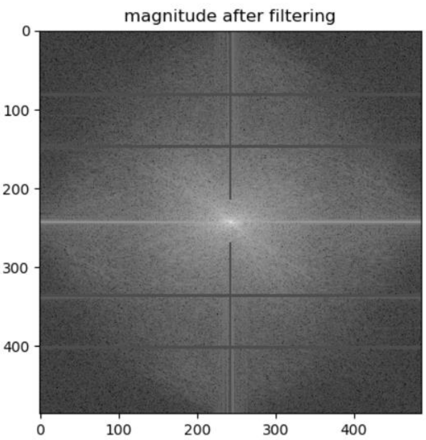
谭树杰 11849060

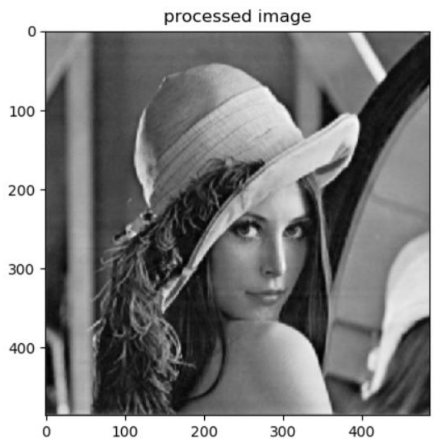
Lab7 Report

1. **There are two noisy images lenaWithNoise.pgm and cameraWithNoise.pgm. The task is to get the noise patterns and to design filters to remove or reduce the noise to get uncorrupted or better quality lena and camera images.**

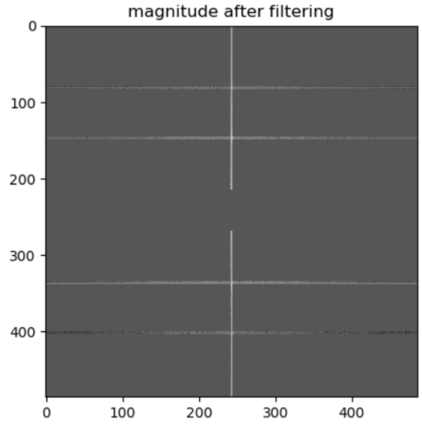
(1) lenaWithNoise.pgm and its magnitude image, magnitude image after bandreject filtering and processed image using bandreject filter are shown below

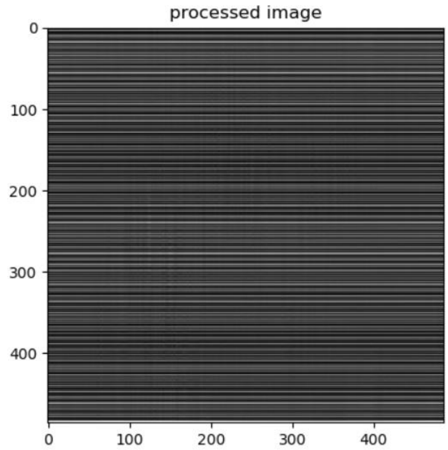






Magnitude after bandpass filtering and processed image are shown below respectively.





Analysis:

We can see four additional horizontal lines and an abnormal vertical line in the magnitude image, which are the magnitude of the noise image. We first calculate the location and then design a bandreject filter to filter the noise frequencies. As we can see, the image improved significantly.

We can get noise pattern by using a bandpass filter to the noise image. As we can see, the noises are mainly periodic noise.

The implementation code is shown below:

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


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)
    mag = 20 * np.log(mag)
    plt.figure(1)
    plt.imshow(mag, 'gray'), plt.title('magnitude before filtering')
    plt.show()
    # calculate location of noise in magnitude image
    # maxi = maxj = max = 0
    # for i in range(0, 20):
    #     for j in range(0, 485):
    #         if mag[i, j] > max:
    #             maxi, maxj = (i, j)
    #             max = mag[i, j]
    # print("maxi " + str(maxi) + " maxj " + str(maxj) + " is " + str(max))

    h = np.ones_like(img, dtype=float)
    h[82-2:82+2, :] = 0
    h[148-2:148+2, :] = 0
    h[403-2:403+2, :] = 0
    h[337-2:337+2, :] = 0
    h[0:216, 243 - 1:243 + 1] = 0
    h[269:, 243 - 1:243 + 1] = 0

    # h = 1 - h   to get bandpass filter
    mag = mag * h
    plt.figure(2)
    plt.imshow(mag, 'gray'), plt.title('magnitude after filtering')
    plt.show()

    img_fft_filtered = img_fft_fftshift * h
    img_fft_filtered_unshift = np.fft.fftshift(img_fft_filtered)
    img_filt = np.fft.ifft2(img_fft_filtered_unshift)
```

```python
    dst = np.abs(img_filt)
    return dst


if __name__ == "__main__":
    path_in = './in/'
    path_out = './out/'
    img_path = 'lenaWithNoise.pgm'    # lenaWithNoise.pgm

    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)
    cv2.normalize(img, img, 0, 1, cv2.NORM_MINMAX)
    cv2.imshow("with noise", img)
    dst = apply_filter(img)
    plt.figure(3)
    plt.imshow(dst, cmap='gray'), plt.title('processed image')
    plt.show()
    cv2.imwrite(img_path_out, dst)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

(2) cameraWithNoise.pgm and filtered image using mean filter 2, 3 and 4 times are shown below. The filter size is 3 * 3.

Analysis:

As we can see, the noise pattern is salt and pepper noise. Therefore, we design a mean filter with size 3 * 3 to reduce the noise. After filtering, the image improved a lot, although with the price of blurring. We repeatedly using the same filter and the image becomes more and more smoothing, hence we should choose the appropriate times of repeated filtering.

The implementation code is shown below:

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

def median_filter(img, size=3):
    pad_width = int(size / 2)
    img_padded = np.pad(img, ((pad_width, pad_width), (pad_width, pad_width)), 'constant')
    out = np.zeros_like(img, dtype=float)
    for row in range(img.shape[0]):
        for col in range(img.shape[1]):
            tmp = []
            for r in range(-pad_width, pad_width+1):
                for c in range(-pad_width, pad_width+1):
                    tmp.append(img_padded[(row+pad_width)+r, (col+pad_width)+c])
            tmp.sort()
            out[row, col] = tmp[ int(size * size / 2) + 1]
    return out

if __name__ == "__main__":
    path_in = './in/'
    path_out = './out/'
    img_path = 'cameraWithNoise.pgm'

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

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

    #####
    dst = median_filter(img, size=3)
    cv2.imshow("dst 1", dst)
    dst = median_filter(img, size=3)
    cv2.imshow("dst 2", dst)
    dst = median_filter(img, size=3)
    cv2.imshow("dst 3", dst)
    cv2.imwrite(img_path_out, dst)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

2. Use arithmetic mean filter, geometric mean filter, median filter, Alpha-trimmed mean filter, and adaptive median filter respectively to reduce the lenaD1.pgm, lenaD2.pgm, lenaD3.pgm. Analyze the results to see which filter is effective to which image.

(1) lenaD1 and result images by processing it with arithmetic mean filter, geometric mean filter, median filter, Alpha-trimmed mean filter, and adaptive median filter respectively are shown below (3 * 3 indicates the filter size, d is pixels trimmed and 3,11 in adaptive median filter represents the minimum, maximum filter size. The parameter representation in the following images are similar)1



LenaD1.pgm (3*3) arithmetic mean (3*3) geometric mean(3*3)



median filter (3*3) Alpha-trimmed mean(3*3,d=2) adaptive median filter(3*3,3,11)



LenaD1.pgm (5*5) arithmetic mean (5*5) geometric mean(5*5)

median filter (5*5) Alpha-trimmed mean(5*5,d=2) adaptive median filter(5*5,5,7)

Analysis:

Obviously, geometric median performs worst for that it introduces many black blocks in both filter size of 3 * 3 and 5 * 5.

As we can see, when the filter size is 3 * 3, there are still strong noise signal in the processed images. The adaptive median filter seems more sharper than others. As we increase filter size to 5, the noise is smoothed more, but the image is blurred more at the same time. Arithmetic mean filter, median filter, Alpha-trimmed mean filter are suitable for this kind of noise.

(2) lenaD2 and result images by processing it with arithmetic mean filter, geometric mean filter, median filter, Alpha-trimmed mean filter, and adaptive median filter respectively are shown below:

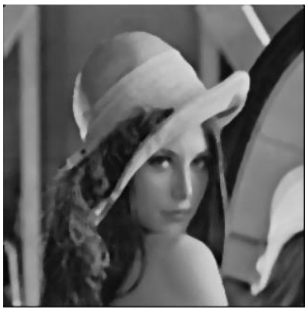
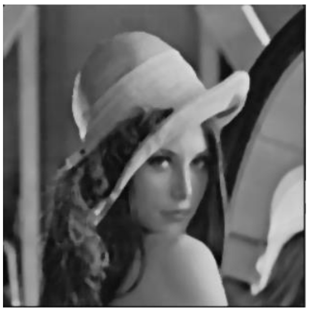
LenaD2.pgm (3*3) arithmetic mean (3*3) geometric mean(3*3)

median filter (3*3) Alpha-trimmed mean(3*3,d=2) adaptive median filter(3*3,3,11)



LenaD2.pgm (5*5) arithmetic mean (5*5) geometric mean(5*5)


median filter (5*5) Alpha-trimmed mean(5*5, d=4) adaptive median filter(5*5,3,7)

Analysis:

Again, geometric median performs worst for that it introduces many black blocks in both filter size of 3 * 3 and 5 * 5. It's hard to recognize original image when the filter size is 5 * 5.

Then the arithmetic mean filter blurred too details compared with others. The median filter and Alpha-trimmed mean filter shows good performance, but the adaptive median filter is the best choice to remove noise in lenaD2.pgm

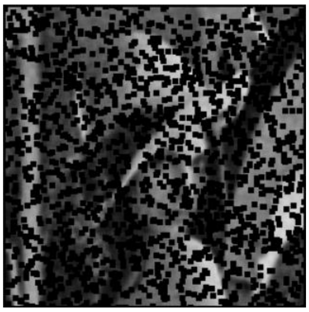
(3) lenaD3 and result images by processing it with arithmetic mean filter, geometric mean filter, median filter, Alpha-trimmed mean filter, and adaptive median filter respectively are shown below:



LenaD3.pgm (3*3) arithmetic mean (3*3) geometric mean(3*3)

median filter (3*3) Alpha-trimmed mean(3*3,d=2) adaptive median filter(3*3,3,11)

LenaD3.pgm (3*3) arithmetic mean (3*3) geometric mean(3*3)

median filter (3*3) Alpha-trimmed mean(3*3, d=2) adaptive median filter(3*3,3,11)

Analysis:

Again, geometric median performs worst for that it introduces many black blocks in both filter size of 3 * 3 and 5 * 5. It's hard to recognize original image when the filter size is 5 * 5.

Then the arithmetic mean filter can't remove enough noise compared with others. The median filter and Alpha-trimmed mean filter shows better performance, but they would blur more details than adaptive median filter. Therefore, adaptive median filter is the best choice to remove noise once again.

The implementation code is shown below:

```python
import numpy as np
import cv2


# arithmetic_mean_filter(img, size=3):
# original image src won't be changed
def arithmetic_mean_filter(src, size=3):
    margin = int(size / 2)
    dst = np.zeros_like(src, dtype=float)
    img_padded = np.pad(src, ((margin, margin), (margin, margin)), 'constant')
    for r in range(margin, img_padded.shape[0] - (margin+1)):
        for c in range(margin, img_padded.shape[1] - (margin + 1)):
            filter_window = np.copy(img_padded[r - margin:r + margin + 1, c - margin:c + margin + 1])
            dst[r - margin, c - margin] = np.sum(filter_window) / (size * size)
    return dst


# geometric_mean_filter(img, size=3):
def geometric_mean_filter(src, size=3):
    margin = int(size / 2)
    dst = np.zeros_like(src, dtype=float)
    img_padded = np.pad(src, ((margin, margin), (margin, margin)), 'constant')
    tmp = np.ones_like(src, dtype=float)
    for row in range(src.shape[0]):
        for col in range(src.shape[1]):
            for r in range(-margin, margin+1):
                for c in range(-margin, margin+1):
                    tmp[row, col] *= img_padded[(row+margin)+r, (col+margin)+c]
            dst[row, col] = np.power(tmp[row, col], 1. / (size * size))
    return dst


# median_filter(img, size=3):
def median_filter(src, size=3):
    margin = int(size / 2)
    dst = np.zeros_like(src, dtype=float)
    img_padded = np.pad(src, ((margin, margin), (margin, margin)), 'constant')
    for r in range(margin, img_padded.shape[0] - (margin + 1)):
        for c in range(margin, img_padded.shape[1] - (margin + 1)):
            filter_window = np.copy(img_padded[r - margin:r + margin + 1, c - margin:c + margin + 1])
            dst[r - margin, c - margin] = np.median(filter_window)
    return dst
```

```python
# alpha_trimmed_median_filter(img, size=3, d=2):
def alpha_trimmed_median_filter(src, size=3, d=2):
    margin = int(size / 2)
    dst = np.zeros_like(src, dtype=float)
    img_padded = np.pad(src, ((margin, margin), (margin, margin)), 'constant')
    for r in range(margin, img_padded.shape[0] - (margin + 1)):
        for c in range(margin, img_padded.shape[1] - (margin + 1)):
            filter_window = np.copy(img_padded[r - margin:r + margin + 1, c - margin:c + margin + 1])
            filter_vector = np.reshape(filter_window, (size*size,))
            filter_vector_sorted = np.sort(filter_vector)
            dst[r - margin, c - margin] = np.median(filter_vector_sorted[int(d / 2):-int(d / 2)])
    return dst

# adaptive_median_filter(img, minsize=3, maxsize=11):
# the kernel is of size * size
def adaptive_median_filter(src, minsize=3, maxsize=11):
    margin = int(maxsize / 2)
    dst = np.zeros_like(src, dtype=float)
    img_padded = np.pad(src, ((margin, margin), (margin, margin)), 'constant')

    def adaptive_one(r, c, size_):
        margin_ = int(size_ / 2)
        filter_window = np.copy(img_padded[r - margin_:r + margin_ + 1, c - margin_:c + margin_ + 1])
        med = np.median(filter_window)
        min_ = np.amin(filter_window)
        max_ = np.amax(filter_window)
        zxy = img_padded[r, c]
        if min_ < med < max_:
            # to B
            if min_ < zxy < max_:
                return zxy
            else:
                return med
        else:
            size_ += 2
            if size_ <= maxsize:
                return adaptive_one(r, c, size_)
            else:
                return med
    for row in range(margin, img_padded.shape[0] - (margin + 1)):
        for col in range(margin, img_padded.shape[1] - (margin + 1)):
            dst[row - margin, col - margin] = adaptive_one(row, col, minsize)
    return dst
```

```python
if __name__ == "__main__":
    path_in = './in/'
    path_out = './out/'
    img_path = 'lenaD1.pgm'

    img_path_in = path_in + img_path
    img_path_out = path_out + 'lenaD1_filtered.jpg'

    # Main code
    img = cv2.imread(img_path_in, cv2.IMREAD_GRAYSCALE)
    img = img.astype(float)
    cv2.normalize(img, img, 0, 1, cv2.NORM_MINMAX)
    cv2.imshow("lenaD1.pgm with noise", img)

    #####
    siz = 3
    dst = arithmetic_mean_filter(img, size=siz)
    cv2.imshow("arithmetic_mean_filter", dst)
    dst = geometric_mean_filter(img, size=siz)
    cv2.imshow("geometric_mean_filter", dst)
    dst = median_filter(img, size=siz)
    cv2.imshow("median_filter", dst)
    dst = alpha_trimmed_median_filter(img,size=siz, d=2)
    cv2.imshow("alpha_trimmed_median_filter", dst)
    dst = adaptive_median_filter(img, minsize=5, maxsize=7)
    cv2.imshow("adaptive_median_filter", dst)

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