HW2 Report

Computer Vision

Tuğba Özkal - 150120053 - 29 Ekim 2017

Spatial Filter

For the first question these libraries are used.

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

All images are read and they are inverted to array.

```
imgN1 = cv2.imread('cameramanN1.jpg')
imgN1_gray = cv2.cvtColor(imgN1, cv2.COLOR_BGR2GRAY)

imgN2 = cv2.imread('cameramanN2.jpg')
imgN2_gray = cv2.cvtColor(imgN2, cv2.COLOR_BGR2GRAY)

imgN3 = cv2.imread('cameramanN3.jpg')
imgN3_gray = cv2.cvtColor(imgN3, cv2.COLOR_BGR2GRAY)


rowN1, colN1 = imgN1_gray.shape
imgN1_arr = np.array(imgN1_gray)

rowN2, colN2 = imgN2_gray.shape
imgN2_arr = np.array(imgN2_gray)

rowN3, colN3 = imgN3_gray.shape
imgN3_arr = np.array(imgN3_gray)
```

If we assume an image row number is x and column number is y, we need a new array has x+2 row and y+2 column. A function is created to rise row and column number.

```
def riseRowCol(img_arr, row, col):
    array = [[0 for y in range(col+2)] for z in range(row+2)]
    for i in range(row):
        for j in range(col):
            array[i+1][j+1] = img_arr[i][j]

    return array
```

Another function is also created for mean filtering.

```
def mean(A, B, row, col):
   arr = [[0 for y in range(col)] for z in range(row)]
   total = 0;
   for i in range(1,row+1):
      for j in range(1,col+1):
         total = A[i-1][j-1] * B[0][0]
         total += A[i-1][j] * B[0][1]
         total += A[i-1][j+1] * B[0][2]
         total += A[i][j-1] * B[1][0]
         total += A[i][j] * B[1][1]
         total += A[i][j+1] * B[1][2]
         total += A[i+1][j-1] * B[2][0]
         total += A[i+1][j] * B[2][1]
         total += A[i+1][j+1] * B[2][2]
         arr[i-1][j-1] = total
   return arr
```

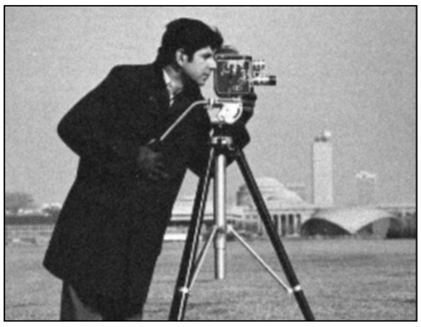
All images are filtered by this function and the results are written as image by using PIL library. I used the array which is given below to calculate mean value.

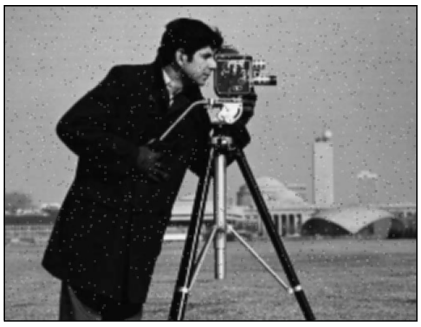
```
kernel = [[1/9,1/9,1/9],[1/9,1/9,1/9],[1/9,1/9,1/9]]
```

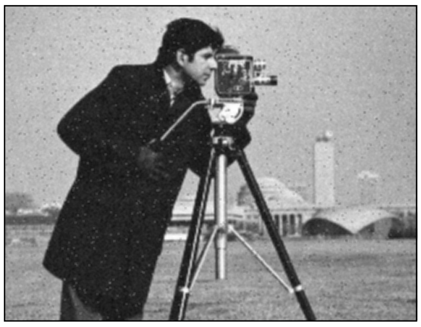
$$\hat{f}(x,y) = \frac{1}{mn} \sum_{(s,t)\in S_{xy}} g(s,t)$$

S_{xy} : set of pixels (window) around pixel (x,y)

New images are given below.


N1

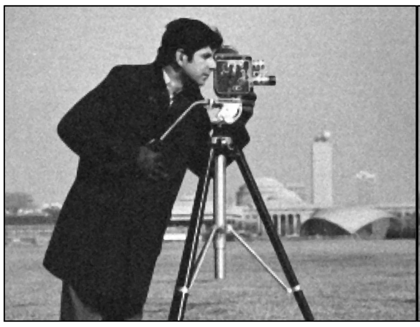

N2

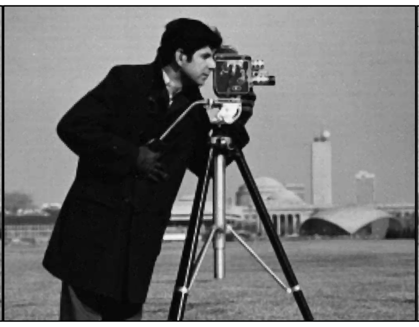

N3

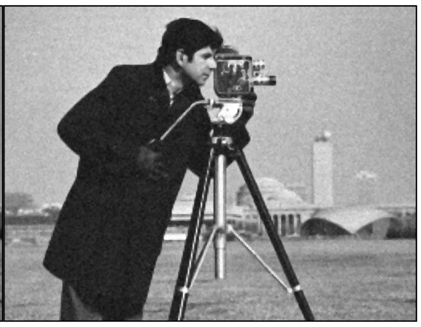
```
def median(A, row, col):
    arr = [[0 for y in range(col)] for z in range(row)]
    kernel = [[0 for y in range(3)] for z in range(3)]
    for i in range(1,row+1):
        for j in range(1,col+1):
            kernel[0] [0] = A[i-1] [j-1]
            kernel[0] [1] = A[i-1] [j]
            kernel[0] [2] = A[i-1] [j+1]
            kernel[1] [0] = A[i] [j-1]
            kernel[1] [1] = A[i] [j]
            kernel[1] [2] = A[i] [j+1]
            kernel[2] [0] = A[i+1] [j-1]
            kernel[2] [1] = A[i+1] [j]
            kernel[2] [2] = A[i+1] [j+1]
            arr[i-1] [j-1] = np.median(kernel)
    return arr
```

$$\hat{f}(x,y) = \underset{(s,t)\in S_{xy}}{\text{median}}\{g(s,t)\}$$

Median filter is applied for 3 images by using the function which is given above. The results are here.


N1


N2

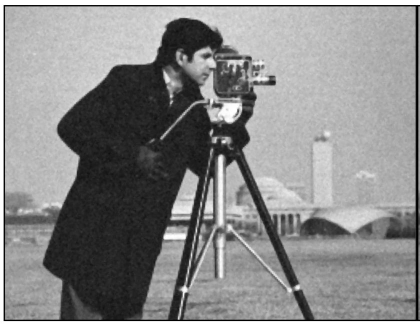
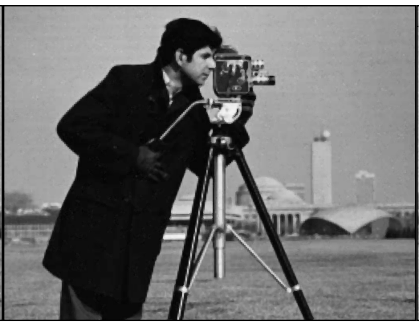
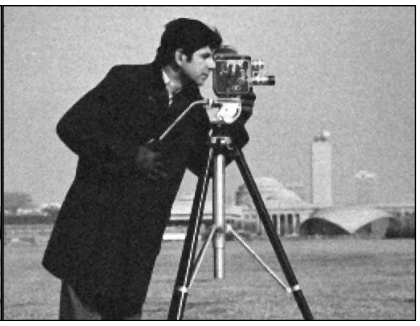

N3

Mean and median filter is applied together with 2 different alfa values.

$$\hat{I} = \alpha \ I_{mean} + (1-\alpha) \ I_{median}$$

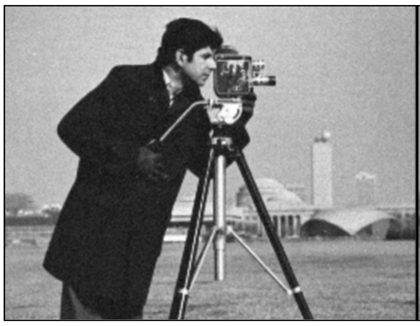
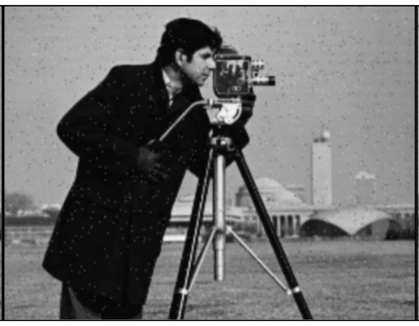
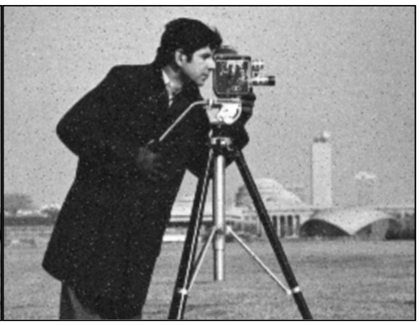
- α : the blending parameter
- I_{mean} : the mean intensity in your filter window
- $I_{median:}$ the median intensity in your filter window

<u>alfa = 0.1</u>

N1 **N2** **N3**

<u>alfa = 0.8</u>

N1 **N2** **N3**

Its function is given below.

```
def MeanMedian(A, B, row, col, alfa):
   arr = [[0 for y in range(col)] for z in range(row)]
   for i in range(row):
      for j in range(col):
         arr[i][j] = (alfa * A[i][j]) + ((1 - alfa) * B[i][j])
   return arr
```

For the first image which has gaussian noise, mean filtering is more appropriate. Mean filtering makes the image less noisy.

For the second image which has impulsive noise, median filter is more appropriate. Median filtering makes the image has less noise.

Kirsch Compass Operator

For the second question these libraries are used.

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

Image is read named I and it is converted to array. A new array created which has row+2 rows and column+2 columns. The values of new rows and columns are zero.

Kirsch operator matrices defined as these.

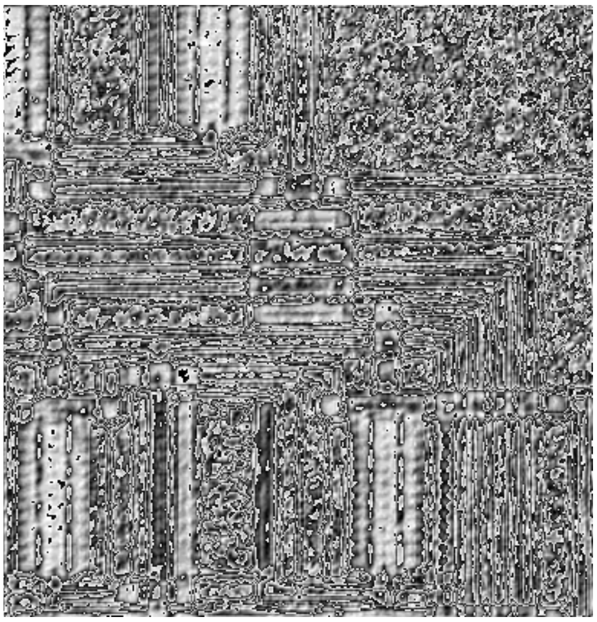
```
N = [[-3,-3,-3],[-3,0,-3],[5,5,5]]
S = [[5,5,5],[-3,0,-3],[-3,-3,-3]]
E = [[5,-3,-3],[5,0,-3],[-5,-3,-3]]
W = [[-3,-3,5],[-3,0,5],[-3,-3,5]]
NW = [[-3,-3,-3],[-3,0,5],[-3,5,5]]
NE = [[-3,-3,-3],[5,0,-3],[5,5,-3]]
SW = [[-3,5,5],[-3,0,5],[-3,-3,-3]]
SE = [[5,5,-3],[5,0,-3],[-3,-3,-3]]
```

Convolution function:

```
def convolution(A, kernel, row, col):
   arr = [[0 for y in range(col)] for z in range(row)]
   total = 0;
   for i in range(1,row+1):
      for j in range(1,col+1):
         total = A[i-1][j-1] * kernel[2][2]
         total += A[i-1][j] * kernel[2][1]
         total += A[i-1][j+1] * kernel[2][0]
         total += A[i][j-1] * kernel[1][2]
         total += A[i][j] * kernel[1][1]
         total += A[i][j+1] * kernel[1][0]
         total += A[i+1][j-1] * kernel[0][2]
         total += A[i+1][j] * kernel[0][1]
         total += A[i+1][j+1] * kernel[0][0]
         arr[i-1][j-1] = total

   return arr
```

I tried to get better results but I could not.
My results seem like this.



East Direction