Technische Universität München
Photogrammetrie und Fernerkundung
Univ.-Prof. Dr.-Ing. U. Stilla

Exercises in Photogrammetry, Remote Sensing, and Image Processing

Name, Given name:

Ho, Hsin-Feng

Exercise number.: 2

Topic: Sampling, Median, Sobel and Laplace filter

Study Program: ESPACE

(filled in by supervisor)

Date: ______________

Points: ______________

Supervisor: ______________

2023

1 Sampling - Coordinate systems

An orthophoto of English Garden in Munich with a pixel resolution of 0.4 m is given. Firstly we have to complete the funcion *subset* in ordert to extract the subset of the image. The function is shown in the following code.

```
    def subset(I, x, y, nu, nv):
    p_ul = (x,y) # tuple with img coord. upper left corner
    p_lr = (x+nu,y+nv) # tuple with img coord. lower right corner

    S = I[p_ul[1] : p_lr[1], p_ul[0] : p_lr[0] , :] # subset S with all channels (:)
    return S
```

The function *subset* takes the image I and the coordinates of the upper left corner (x, y) and the lower right corner $(x + nu, y + nv)$ of the subset as input. The function returns the subset S of the image I.

A subset of the image is shown in Figure 1. The subset is the area of the English Garden in Munich. The subset is extracted from the image with the function *subset*.

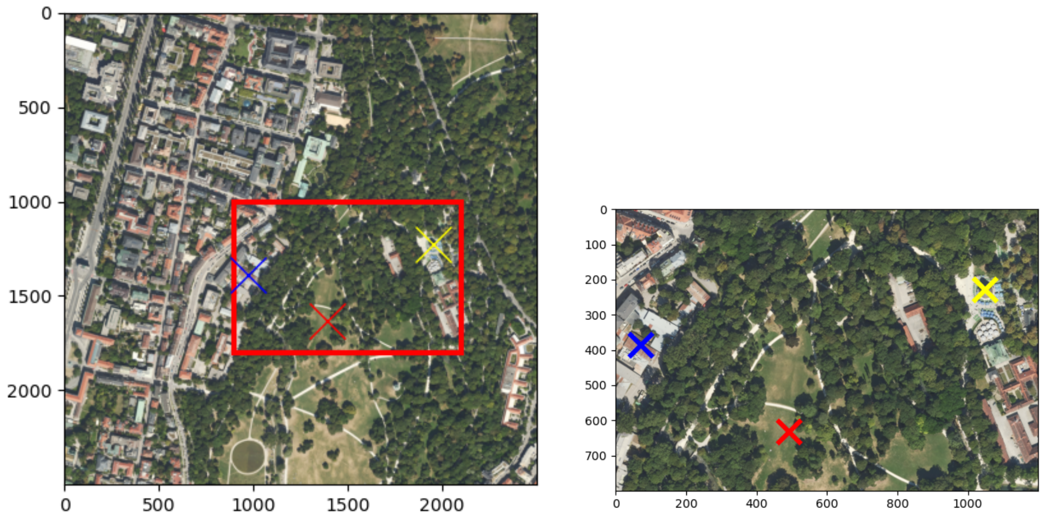

Figure 1: Subset of the image

We also have to calculate the UTM coordinates of the three points in the subset. The coordinates of the low right corner of the image is given. The coordinates of the three points are calculated with the scale and translation. The results are shown in the following.

```
p1_world:  [ 692780.   5336508.4]
p2_world:  [ 692388.8 5336445.2]
p3_world:  [ 692557.2 5336346.8]
```

2 Median and Mean

In the first part of this exercise we have to implement the functions *median* in order to calculate the median of a chess board like image. The function is shown in the following code.

```
def median(listValues):
median_value = 0.0
if len(listValues) == 2:
    raise ValueError('Number of values must be odd')
else:
    listValues.sort()
    median_value = listValues[int(len(listValues)/2)]
return median_value
```

And we have to calculate the mean and median for the following list of values: [19.7, 556.3, 23.2, 27.5, 16.3, 21.0, 27.2, 495.0, 25.3]. The results are shown in the following code.

```
Mean:  134.61111111111111
Median:  25.3
```

3 Filters

In the second part of this exercise we have to use the mean filter and median filter to filter the chess board like image. The results are shown in Figure 2.

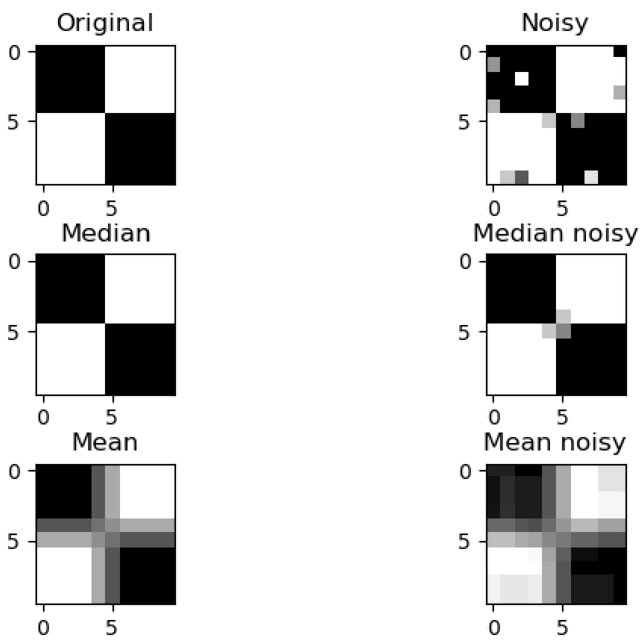

Figure 2: Mean and median filter

From the results we can see that the mean filter is not able to remove the noise in the image. However, the median filter is able to remove the noise in the image. The reason is that the median filter is more robust to outliers than the mean filter. And the mean filter will blur the image. If we want to remove the noise in the image, we should use the median filter. If we want to blur the image, we should use the mean filter.

There are other thechniques to perform the expanding of the image before applying the convolution. For example, we can use the zero padding to expand the image. The advantage of the zero padding is that it is easy to implement. The disadvantage of the zero padding is that it will introduce the artifacts in the image. Another example is the mirror padding. The advantage of the mirror padding is that it will not introduce the artifacts in the image. The disadvantage of the mirror padding is that it is difficult to implement. The other posibility is the replicate padding. It's similar to the mirror padding, which will not introduce the artifacts in the image. However, it is also difficult to implement.

4 Sobel and Laplace filter

In the third part of this exercise we have to implement the Sobel filter and Laplace filter. The results are shown in Figure 3 and Figure 4.

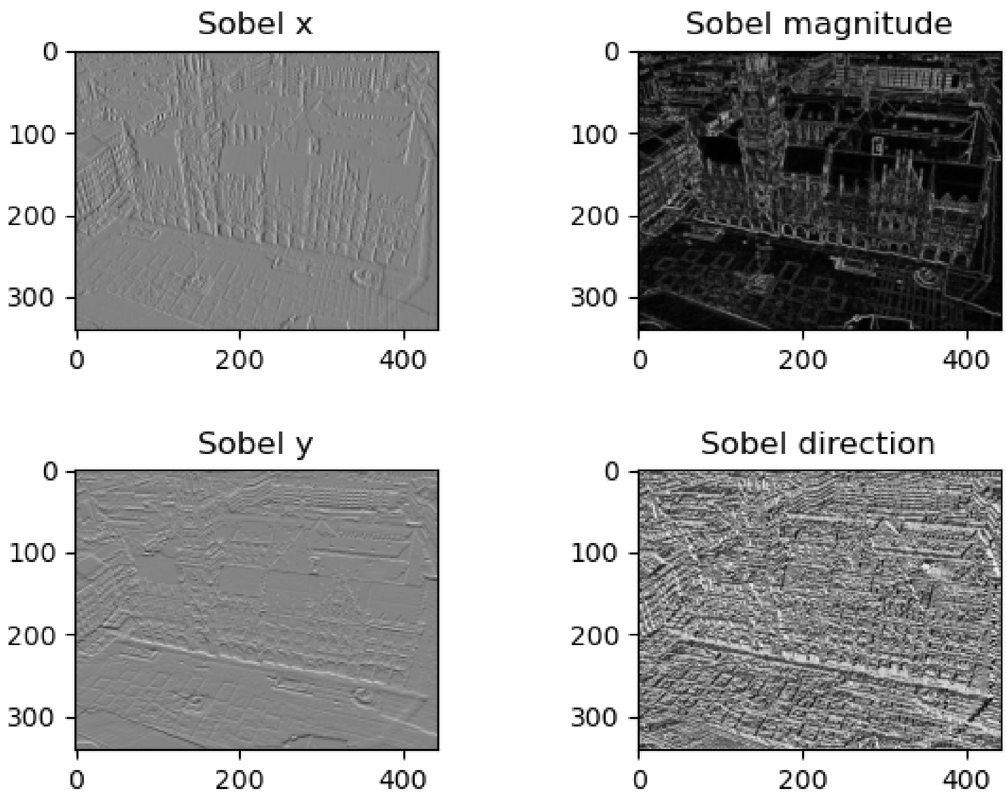

Figure 3: Sobel filter

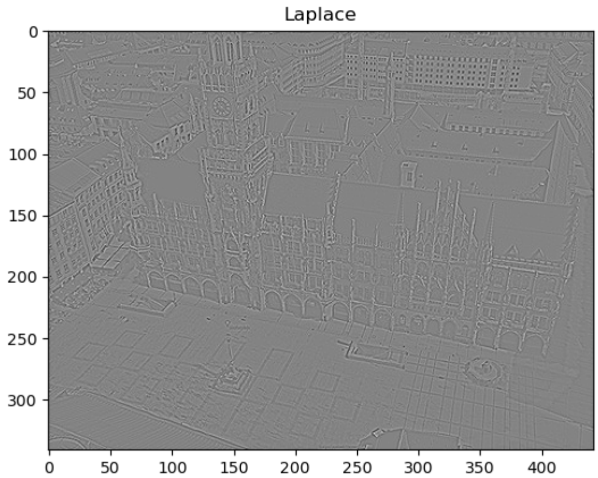

Figure 4: Laplace filter

We have first to define the kernel of the Sobel filter and Laplace filter.

$$\text{Sobel filter x: } \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\text{Sobel filter y: } \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$\text{Laplace filter: } \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

The calculation of magnitude and direction of the gradient is shown in the following code.

```
    Smag = np.sqrt(Sx**2 + Sy**2)
    Sdir = np.arctan2(Sy,Sx)
```

The Sobel filter and Laplace filter are both used for edge detection in an image. The Sobel filter is a discrete differentiation operator. It is used to compute an approximation of the gradient of the image intensity function. The Laplace filter is a second order derivative edge detector. It is used to find the zero crossings in the second derivative of the image intensity function.

Code

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import numpy as np

# 1. Sampling and coord. systems

def subset(I, x, y, nu, nv):
    p_ul = (x,y) # tuple with img coord. upper left corner
    p_lr = (x+nu,y+nv) # tuple with img coord. lower right corner

    S = I[p_ul[1] : p_lr[1], p_ul[0] : p_lr[0] , :] # subset S with all channels (:)
    return S

def uv2world(easting, northing, scale, u, v):
    d_est = u*scale # offset of the given coord. u and the upper left corner in meter
    d_north = v*scale # offset of the given coord. v and the upper left corner in meter
    u_est = easting+d_est # easting of the given coord. u
    v_north = northing+d_north # northing of the given coord. u
    return u_est, v_north


def loadWordFile(path_in):
  # read data as string and convert them to np array with float values
  f = open(path_in, 'r') # 'r' = read
  data_string = f.read()
  data_array = np.fromstring(data_string, dtype=float , sep = '\n')
  f.close()
  scale = data_array[0]
  scale_negative = data_array[3]
  easting = data_array[4]
  northing = data_array[5]
  return easting, northing, scale, scale_negative

# path to files
path_tfw = 'data/32692_5336.tfw'
path_I = 'data/32692_5336.tif'


# image coord. of the first pixel of the subset S according to the image I
ps_img = np.array([900, 1000])
# width and height of S
nu, nv = [1200, 800]

# image coord. of the given point according to S
```

```
p1_img_subset = np.array([1050, 229])
p2_img_subset = np.array([72, 387])
p3_img_subset = np.array([493, 633])


img_dop  = plt.imread(path_I) # read image
img_dop_subset = subset(img_dop, ps_img[0], ps_img[1], nu, nv) #
    subset of the original orthophoto image
p1_img = ps_img + p1_img_subset
p2_img = ps_img + p2_img_subset
p3_img = ps_img + p3_img_subset

# plot image
fig, ax = plt.subplots()
ax.plot(p1_img[0], p1_img[1], marker='x', color="yellow",
    markersize=20)
ax.plot(p2_img[0], p2_img[1], marker='x', color="blue",
    markersize=20)
ax.plot(p3_img[0], p3_img[1], marker='x', color="red",
    markersize=20)
ax.imshow(img_dop)
rect = patches.Rectangle((ps_img[0], ps_img[1]), nu, nv,
    linewidth=3, edgecolor='r', facecolor='none')
ax.add_patch(rect)
#plt.savefig('ex02_task1_1.png')


# img_dop_subset = ... # TODO #subset of the original orthophoto
    image
# world coordinates

easting, northing, scale, scale_negative = loadWordFile(path_tfw)
origin = np.array([easting+img_dop.shape[1]*scale_negative,
    northing+img_dop.shape[0]*scale])
ps_world = origin + np.array([ps_img[0]*scale,
    ps_img[1]*scale_negative])
p1_img_world = ps_world + np.array([p1_img_subset[0]*scale,
    p1_img_subset[1]*scale_negative])
p2_img_world = ps_world + np.array([p2_img_subset[0]*scale,
    p2_img_subset[1]*scale_negative])
p3_img_world = ps_world + np.array([p3_img_subset[0]*scale,
    p3_img_subset[1]*scale_negative])
print('p1_world: ', p1_img_world)
print('p2_world: ', p2_img_world)
print('p3_world: ', p3_img_world)

# plot subset
fig, ax = plt.subplots()
ax.plot(p1_img_subset[0], p1_img_subset[1], marker='x',
    color="yellow", markeredgewidth=4, markersize=20)
```

```
ax.plot(p2_img_subset[0], p2_img_subset[1], marker='x',
    color="blue", markeredgewidth=4, markersize=20)
ax.plot(p3_img_subset[0], p3_img_subset[1], marker='x',
    color="red", markeredgewidth=4, markersize=20)
ax.imshow(img_dop_subset)
#plt.savefig('ex02_task1_2.png')
#plt.show()
```

ex02_task1.py

```
import matplotlib.pyplot as plt
import numpy as np


# Task 2 - Mean and Median

def median(listValues):
    median_value = 0.0
    if len(listValues) == 2:
        raise ValueError('Number of values must be odd')
    else:
        sorted_listValues = np.sort(listValues)
        median_value = sorted_listValues[int(len(listValues)/2)]

    return median_value

# 1D median & mean
values = [19.7, 556.3, 23.2, 27.5, 16.3, 21.0, 27.2, 495.0, 25.3]
print('Mean: ', np.mean(values))
print('Median: ', median(values))

# 2D median & mean

def addBorder_mirror(img, br, bc):
    row,col = img.shape
    imgOut = np.zeros((row +2*br,col+2*bc),np.uint8)
    r = 0
    c = 0
    for px in np.nditer(imgOut[:,:], op_flags=["readwrite"]):
        rI = br - r
        cI = bc - c
        if rI < 0:
            rI = r - br
        if rI >= row:
            rI = row - (rI - row) - 2
        if cI < 0:
            cI = c - bc
        if cI >= col:
            cI = col - (cI - col) - 2
        px[...] = img[rI,cI]
        c+=1
        if c >= imgOut.shape[1]:
```

```
                c=0
                r+=1
    return imgOut


def convolution(img,kernel):
    rows,cols = kernel.shape
    n = float(rows*cols)
    rI,cI = img.shape
    imgOut = np.zeros((rI,cI),np.float32)
    startC = int(cols/2 )
    startR = int(rows/2 )
    imgBorder = addBorder_mirror(img, startR, startC)
    imgBorder.astype(np.float32)
    r = 0
    c = 0
    for pxOut in np.nditer(imgOut[:,:], op_flags =["writeonly"]):
        it = np.nditer([imgBorder[r : r+2*startR+1 , c :
           c+2*startC+1],kernel[:,:]],flags=["buffered","external_loop"],op_fl
           =["readonly"], op_dtypes=["float64","float64"])
        val = 0.0
        for i,k in it:
            val += np.sum(i*k)
        pxOut[...] = val
        c+= 1
        if c >= imgOut.shape[1]:
            c=0
            r+=1
    return imgOut


def medianImg(img,size):
    imgB = addBorder_mirror(img,int(size/2),int(size/2))
    imgOut = np.zeros(img.shape,img.dtype)
    c=0
    r=0
    for pxOut in np.nditer(imgOut[:,:], op_flags =["writeonly"]):
        it = np.nditer(imgB[r : r+size , c :
           c+size],flags=["buffered","external_loop"],op_flags
           =["readonly"])
        for x in it:
           # print(x)
            pxOut[...] = median(x)
        c+=1
        if c >= imgOut.shape[1]:
            c=0
            r+=1
    return imgOut


chessboard =
```

```
   np.array([[0,0,0,0,0,255,255,255,255,255],[0,0,0,0,0,255,255,255,255,255],
                        [0,0,0,0,0,255,255,255,255,255],[0,0,0,0,0,255,255,255,
                        [0,0,0,0,0,255,255,255,255,255],[255,255,255,255,255,0,
                        [255,255,255,255,255,0,0,0,0,0],[255,255,255,255,255,0,
                        [255,255,255,255,255,0,0,0,0,0],[255,255,255,255,255,0,

chessboard_noisy = np.loadtxt("data/chessboard_noisy.txt")

chessboard_median = medianImg(chessboard,3)
chessboard_mean =
   convolution(chessboard,np.ones((3,3),np.float32)/9)
chessboard_noisy_median = medianImg(chessboard_noisy,3)
chessboard_noisy_mean =
   convolution(chessboard_noisy,np.ones((3,3),np.float32)/9)
figure, axes = plt.subplots(3, 2)
plt.subplots_adjust(wspace=0.5, hspace=0.5)
axes[0,0].imshow(chessboard, cmap='gray')
axes[0,0].set_title("Original")
axes[1,0].imshow(chessboard_median, cmap='gray')
axes[1,0].set_title("Median")
axes[2,0].imshow(chessboard_mean, cmap='gray')
axes[2,0].set_title("Mean")
axes[0,1].imshow(chessboard_noisy, cmap='gray')
axes[0,1].set_title("Noisy")
axes[1,1].imshow(chessboard_noisy_median, cmap='gray')
axes[1,1].set_title("Median noisy")
axes[2,1].imshow(chessboard_noisy_mean, cmap='gray')
axes[2,1].set_title("Mean noisy")
#plt.show()
#plt.savefig("chessboard.png")
# Task 3 - Sobel and Laplace

def sobel(img):
    ksx = np.zeros((3,3),np.float32)
    ksx[:]= [[-1,0,1],[-2,0,2],[-1,0,1]]
    ksy = np.zeros((3,3),np.float32)
    ksy[:]= [[-1,-2,-1],[0,0,0],[1,2,1]]
    Sx = convolution(img, ksx)
    Sy = convolution(img, ksy)
    return Sx,Sy

def laplace(img):
    kl = np.zeros((3,3),np.float32)
    kl[:]= [[0,-1,0],[-1,4,-1],[0,-1,0]]
    return convolution(img,kl)

img=plt.imread("data/old_town.jpg")
Sx,Sy = sobel(img)
Smag = np.sqrt(Sx**2 + Sy**2)
Sdir = np.arctan2(Sy,Sx)
L = laplace(img)
```

```
figure, axes = plt.subplots(2, 2)
plt.subplots_adjust(wspace=0.5, hspace=0.5)
axes[0,0].imshow(Sx, cmap='gray')
axes[0,0].set_title("Sobel x")
axes[1,0].imshow(Sy, cmap='gray')
axes[1,0].set_title("Sobel y")
axes[0,1].imshow(Smag, cmap='gray')
axes[0,1].set_title("Sobel magnitude")
axes[1,1].imshow(Sdir, cmap='gray')
axes[1,1].set_title("Sobel direction")
#plt.savefig("sobel.png")

plt.figure()
laplace_img = laplace(img)
plt.imshow(laplace_img, cmap='gray')
plt.title("Laplace")
##plt.savefig("laplace.png")
#plt.show()
```

ex02_task2_3.py