

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 function *subset* in order to extract the subset of the image. The function is shown in the following code.

```
1 def subset(I, x, y, nu, nv):  
2     p_ul = (x,y) # tuple with img coord. upper left corner  
3     p_lr = (x+nu,y+nv) # tuple with img coord. lower right corner  
4  
5     S = I[p_ul[1] : p_lr[1], p_ul[0] : p_lr[0] , :] # subset S  
6         with all channels (:)  
7     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*.

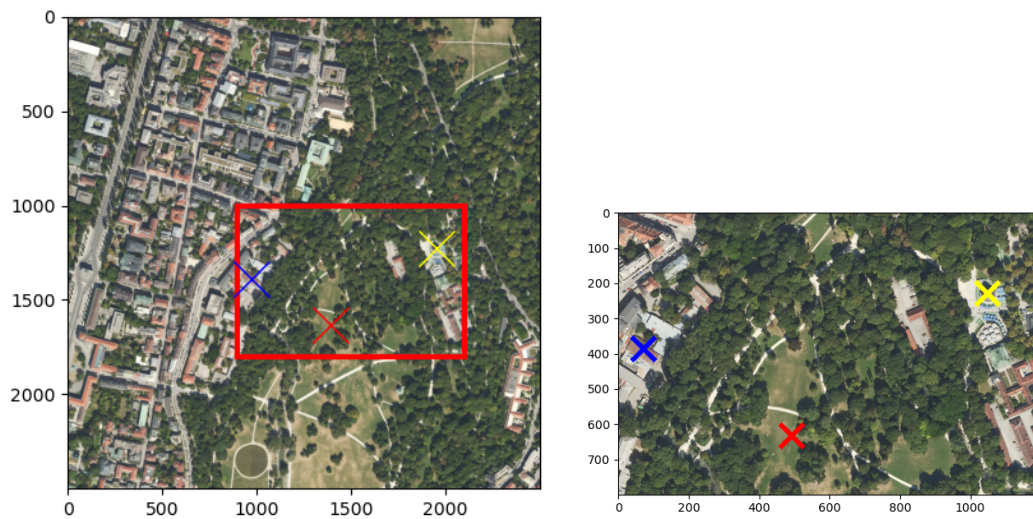


Figure 1: Subset of the image

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.

```
1 def median(listValues):  
2     median_value = 0.0  
3     if len(listValues) == 2:  
4         raise ValueError('Number of values must be odd')  
5     else:  
6         listValues.sort()  
7         median_value = listValues[int(len(listValues)/2)]  
8     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.

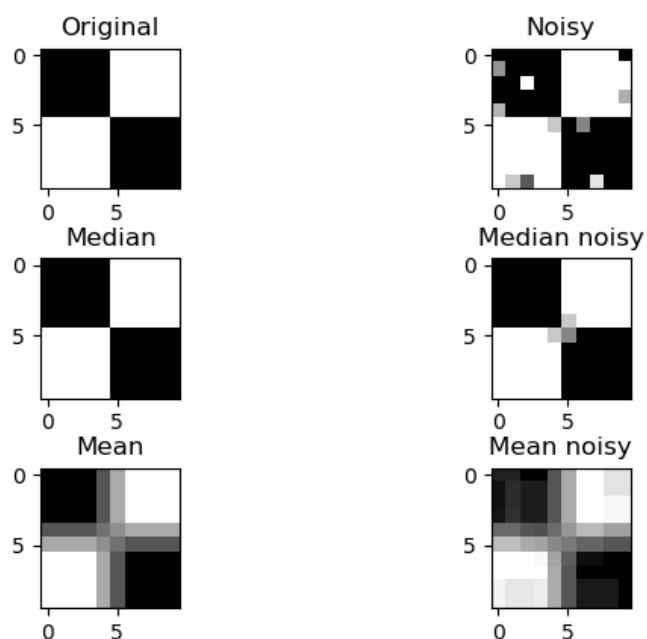


Figure 2: Mean and median filter

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 techniques 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 possibility 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.

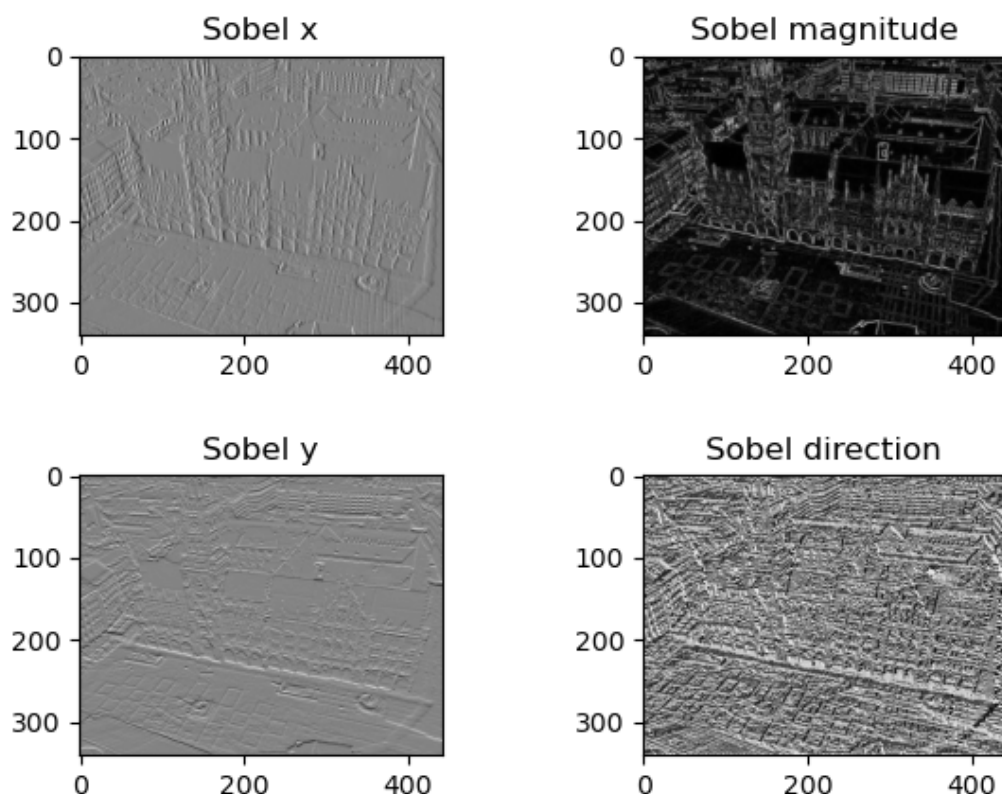


Figure 3: Sobel filter



Figure 4: Laplace filter

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

$$\begin{aligned}\text{Sobel filter x: } & \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \\ \text{Sobel filter y: } & \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \\ \text{Laplace filter: } & \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}\end{aligned}$$

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

```
1  Smag = np.sqrt(Sx**2 + Sy**2)
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

```
1 import matplotlib.pyplot as plt
2 import matplotlib.patches as patches
3 import numpy as np
4
5 # 1. Sampling and coord. systems
6
7 def subset(I, x, y, nu, nv):
8     p_ul = (x,y) # tuple with img coord. upper left corner
9     p_lr = (x+nu,y+nv) # tuple with img coord. lower right corner
10
11     S = I[p_ul[1] : p_lr[1], p_ul[0] : p_lr[0] , :] # subset S
12         with all channels (:)
13     return S
14
15 def uv2world(easting, northing, scale, u, v):
16     d_est = u*scale # offset of the given coord. u and the upper
17         left corner in meter
18     d_north = v*scale # offset of the given coord. v and the upper
19         left corner in meter
20     u_est = easting+d_est # easting of the given coord. u
21     v_north = northing+d_north # northing of the given coord. u
22     return u_est, v_north
23
24 def loadWordFile(path_in):
25     # read data as string and convert them to np array with float
26     values
27     f = open(path_in, 'r') # 'r' = read
28     data_string = f.read()
29     data_array = np.fromstring(data_string, dtype=float , sep = '\n')
30     f.close()
31     scale = data_array[0]
32     scale_negative = data_array[3]
33     easting = data_array[4]
34     northing = data_array[5]
35     return easting, northing, scale, scale_negative
36
37 # path to files
38 path_tfw = 'data/32692_5336.tfw'
39 path_I = 'data/32692_5336.tif'
40
41 # image coord. of the first pixel of the subset S according to the
42 image I
43 ps_img = np.array([900, 1000])
44 # width and height of S
45 nu, nv = [1200, 800]
46
47 # image coord. of the given point according to S
```

```
45 p1_img_subset = np.array([1050, 229])
46 p2_img_subset = np.array([72, 387])
47 p3_img_subset = np.array([493, 633])
48
49
50 img_dop = plt.imread(path_I) # read image
51 img_dop_subset = subset(img_dop, ps_img[0], ps_img[1], nu, nv) #
    subset of the original orthophoto image
52 p1_img = ps_img + p1_img_subset
53 p2_img = ps_img + p2_img_subset
54 p3_img = ps_img + p3_img_subset
55
56 # plot image
57 fig, ax = plt.subplots()
58 ax.plot(p1_img[0], p1_img[1], marker='x', color="yellow",
    markersize=20)
59 ax.plot(p2_img[0], p2_img[1], marker='x', color="blue",
    markersize=20)
60 ax.plot(p3_img[0], p3_img[1], marker='x', color="red",
    markersize=20)
61 ax.imshow(img_dop)
62 rect = patches.Rectangle((ps_img[0], ps_img[1]), nu, nv,
    linewidth=3, edgecolor='r', facecolor='none')
63 ax.add_patch(rect)
64 #plt.savefig('ex02_task1_1.png')
65
66
67 # img_dop_subset = ... # TODO #subset of the original orthophoto
    image
68 # world coordinates
69
70 easting, northing, scale, scale_negative = loadWordFile(path_tfw)
71 origin = np.array([easting+img_dop.shape[1]*scale_negative,
    northing+img_dop.shape[0]*scale])
72 ps_world = origin + np.array([ps_img[0]*scale,
    ps_img[1]*scale_negative])
73 p1_img_world = ps_world + np.array([p1_img_subset[0]*scale,
    p1_img_subset[1]*scale_negative])
74 p2_img_world = ps_world + np.array([p2_img_subset[0]*scale,
    p2_img_subset[1]*scale_negative])
75 p3_img_world = ps_world + np.array([p3_img_subset[0]*scale,
    p3_img_subset[1]*scale_negative])
76 print('p1_world: ', p1_img_world)
77 print('p2_world: ', p2_img_world)
78 print('p3_world: ', p3_img_world)
79
80 # plot subset
81 fig, ax = plt.subplots()
82 ax.plot(p1_img_subset[0], p1_img_subset[1], marker='x',
    color="yellow", markeredgewidth=4, markersize=20)
```

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

ex02_task1.py

```
1 import matplotlib.pyplot as plt  
2 import numpy as np  
3  
4  
5 # Task 2 - Mean and Median  
6  
7 def median(listValues):  
8     median_value = 0.0  
9     if len(listValues) == 2:  
10         raise ValueError('Number of values must be odd')  
11     else:  
12         sorted_listValues = np.sort(listValues)  
13         median_value = sorted_listValues[int(len(listValues)/2)]  
14  
15     return median_value  
16  
17 # 1D median & mean  
18 values = [19.7, 556.3, 23.2, 27.5, 16.3, 21.0, 27.2, 495.0, 25.3]  
19 print('Mean: ', np.mean(values))  
20 print('Median: ', median(values))  
21  
22 # 2D median & mean  
23  
24 def addBorder_mirror(img, br, bc):  
25     row,col = img.shape  
26     imgOut = np.zeros((row +2*br,col+2*bc),np.uint8)  
27     r = 0  
28     c = 0  
29     for px in np.nditer(imgOut[:, :], op_flags=["readwrite"]):  
30         rI = br - r  
31         cI = bc - c  
32         if rI < 0:  
33             rI = r - br  
34         if rI >= row:  
35             rI = row - (rI - row) - 2  
36         if cI < 0:  
37             cI = c - bc  
38         if cI >= col:  
39             cI = col - (cI - col) - 2  
40         px[...] = img[rI,cI]  
41         c+=1  
42         if c >= imgOut.shape[1]:
```

```
43         c=0
44         r+=1
45     return imgOut
46
47
48 def convolution(img,kernel):
49     rows,cols = kernel.shape
50     n = float(rows*cols)
51     rI,cI = img.shape
52     imgOut = np.zeros((rI,cI),np.float32)
53     startC = int(cols/2 )
54     startR = int(rows/2 )
55     imgBorder = addBorder_mirror(img, startR, startC)
56     imgBorder.astype(np.float32)
57     r = 0
58     c = 0
59     for pxOut in np.nditer(imgOut[:,:], op_flags=["writeonly"]):
60         it = np.nditer([imgBorder[r : r+2*startR+1 , c :
61             c+2*startC+1],kernel[:,:]],flags=["buffered","external_loop"],op_flags=
62             ["readonly"], op_dtypes=["float64","float64"])
63         val = 0.0
64         for i,k in it:
65             val += np.sum(i*k)
66         pxOut[...] = val
67         c+= 1
68         if c >= imgOut.shape[1]:
69             c=0
70             r+=1
71     return imgOut
72
73 def medianImg(img,size):
74     imgB = addBorder_mirror(img,int(size/2),int(size/2))
75     imgOut = np.zeros(img.shape,img.dtype)
76     c=0
77     r=0
78     for pxOut in np.nditer(imgOut[:,:], op_flags=["writeonly"]):
79         it = np.nditer(imgB[r : r+size , c :
80             c+size],flags=["buffered","external_loop"],op_flags=
81             ["readonly"])
82         for x in it:
83             # print(x)
84             pxOut[...] = median(x)
85         c+=1
86         if c >= imgOut.shape[1]:
87             c=0
88             r+=1
89     return imgOut
90
91 chessboard =
```



```

90         np.array([[0,0,0,0,0,255,255,255,255,255],[0,0,0,0,0,255,255,255,255,255],
91                  [0,0,0,0,0,255,255,255,255,255],[0,0,0,0,0,255,255,255,255,255],
92                  [0,0,0,0,0,255,255,255,255,255],[255,255,255,255,255,0,0,0,0,0],
93                  [255,255,255,255,255,0,0,0,0,0],[255,255,255,255,255,0,0,0,0,0],
94                  [255,255,255,255,255,0,0,0,0,0],[255,255,255,255,255,0,0,0,0,0]])
95 chessboard_noisy = np.loadtxt("data/chessboard_noisy.txt")
96
97 chessboard_median = medianImg(chessboard,3)
98 chessboard_mean =
99     convolution(chessboard,np.ones((3,3),np.float32)/9)
100 chessboard_noisy_median = medianImg(chessboard_noisy,3)
101 chessboard_noisy_mean =
102     convolution(chessboard_noisy,np.ones((3,3),np.float32)/9)
103 figure, axes = plt.subplots(3, 2)
104 plt.subplots_adjust(wspace=0.5, hspace=0.5)
105 axes[0,0].imshow(chessboard, cmap='gray')
106 axes[0,0].set_title("Original")
107 axes[1,0].imshow(chessboard_median, cmap='gray')
108 axes[1,0].set_title("Median")
109 axes[2,0].imshow(chessboard_mean, cmap='gray')
110 axes[2,0].set_title("Mean")
111 axes[0,1].imshow(chessboard_noisy, cmap='gray')
112 axes[0,1].set_title("Noisy")
113 axes[1,1].imshow(chessboard_noisy_median, cmap='gray')
114 axes[1,1].set_title("Median noisy")
115 axes[2,1].imshow(chessboard_noisy_mean, cmap='gray')
116 axes[2,1].set_title("Mean noisy")
117 #plt.show()
118 #plt.savefig("chessboard.png")
119 # Task 3 - Sobel and Laplace
120
121 def sobel(img):
122     ksx = np.zeros((3,3),np.float32)
123     ksx[:] = [[-1,0,1],[-2,0,2],[-1,0,1]]
124     ksy = np.zeros((3,3),np.float32)
125     ksy[:] = [[-1,-2,-1],[0,0,0],[1,2,1]]
126     Sx = convolution(img, ksx)
127     Sy = convolution(img, ksy)
128     return Sx,Sy
129
130 def laplace(img):
131     kl = np.zeros((3,3),np.float32)
132     kl[:] = [[0,-1,0],[-1,4,-1],[0,-1,0]]
133     return convolution(img,kl)
134
135 img=plt.imread("data/old_town.jpg")
136 Sx,Sy = sobel(img)
137 Smag = np.sqrt(Sx**2 + Sy**2)
138 Sdir = np.arctan2(Sy,Sx)
139 L = laplace(img)

```

```
138 figure, axes = plt.subplots(2, 2)
139 plt.subplots_adjust(wspace=0.5, hspace=0.5)
140 axes[0,0].imshow(Sx, cmap='gray')
141 axes[0,0].set_title("Sobel x")
142 axes[1,0].imshow(Sy, cmap='gray')
143 axes[1,0].set_title("Sobel y")
144 axes[0,1].imshow(Smag, cmap='gray')
145 axes[0,1].set_title("Sobel magnitude")
146 axes[1,1].imshow(Sdir, cmap='gray')
147 axes[1,1].set_title("Sobel direction")
148 #plt.savefig("sobel.png")
149
150 plt.figure()
151 laplace_img = laplace(img)
152 plt.imshow(laplace_img, cmap='gray')
153 plt.title("Laplace")
154 ##plt.savefig("laplace.png")
155 #plt.show()
```

ex02_task2_3.py