Lab Instructions - session 10

Image Pyramid, Multiscale corner detection

Image Downsampling

We intend to downsize the following image by a factor of '*s*' (default *s*=4). The following code reduces the size of the image by selecting every '*s*' pixel both in the x and y directions (that is `I[::s, ::s, :]`). The output image, however, looks a bit weird. So, we need to do a little better than just picking every s pixels. Run the following code. Notice that initially **J**, **Jb** and **Jg** (three images displayed in the second row of the figure) are all the same.

File: **downsize.py**

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

I = cv2.imread('toosi.jpg')
s = 4; # downsize with a factor of s

# Downsize by sampling every s pixels:
J = I[::s, ::s, :]

Jb = Jg = J

# blur with a box filter, then downsample
# ksize = s + 1;
# Ib = cv2.boxFilter(I, -1, (ksize,ksize))
# Jb = Ib[::s, ::s, :]


# blur with a Gaussian filter, then resample
# sigma = (s+1)/np.sqrt(12) # equivalent sigma for Guassian kernel
# Ig = cv2.GaussianBlur(I, (0,0),sigma)
# Jg = Ig[::s, ::s, :]

f, ax = plt.subplots(2,3, gridspec_kw={'height_ratios': [s,1]})

# do not change this (turns off the axes)
for a in ax.ravel():
    a.axis('off')

ax[0,1].set_title('Original')
ax[0,1].imshow(I[:,:,::-1])

ax[1,0].set_title('Downsized')
ax[1,0].imshow(J[:,:,::-1], interpolation='none')
```

```
ax[1,1].set_title('Box Blur + Downsized')
ax[1,1].imshow(Jb[:,:,::-1], interpolation='none');

ax[1,2].set_title('Gaussian Blur + Downsized')
ax[1,2].imshow(Jg[:,:,::-1], interpolation='none');

plt.show()
```

- What do you think is wrong with the output image? Why is this happening?
- Redefine **Jb** by uncommenting the three lines after the line `# blur with a box filter, then downsample`. To create Jb, we first blur the image with a box filter and then downsample as before. How does this change the output image?
- Uncomment the three lines after the line `# blur with a Gaussian filter, then resample` to redefine **Jg**. This is the same as **Jb**, but this time a Gaussian filter is used for blurring instead of a box filter. See the result.

The initial assignment of Jb = Jg = J causes all three methods (simple sampling, box blur, and Gaussian blur) to initially share the same array.

Uncommenting the respective lines for Jb and Jg ensures that these operations are performed correctly and independently, resulting in the expected smoother downscaled images.

Image pyramid

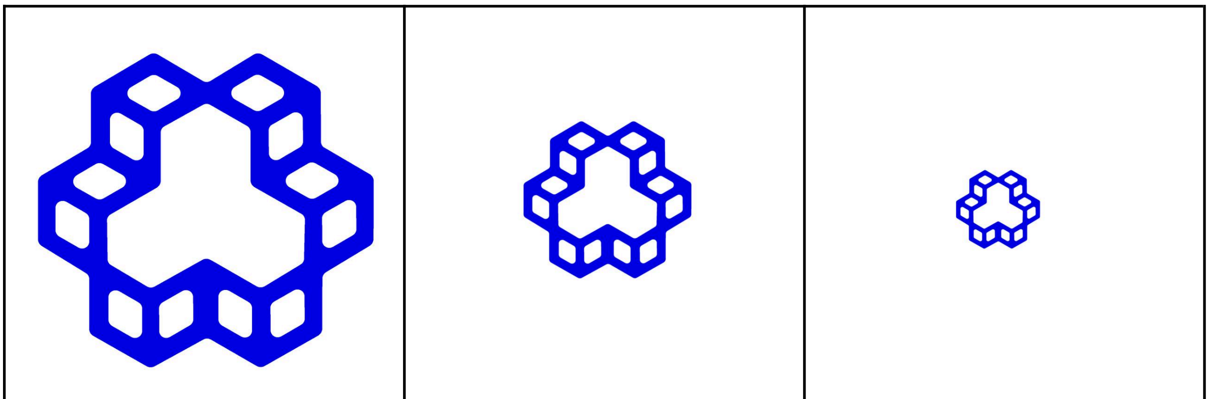
An image pyramid is created by repeatedly blurring and downsampling an image. The OpenCV function `pyrDown` creates the next level of the pyramid from the previous level. A pyramid can be created by repeatedly calling this function. The following code creates an image pyramid and displays it. Here, the pyramid gets stored in a Python list (not a numpy array).

File: **pyramid.py**

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

I = cv2.imread('toosi.jpg')

psize = 6 # size of the pyramid (no. of levels)

# building the pyramid
J = I.copy()
Pyr = [J] # the first element is simply the original image
for i in range(psize-1):
    J = cv2.pyrDown(J) # blurs, then downsampled by a factor of 2
    Pyr.append(J)

# display the pyramid
# do not bother about the next two lines
size_list = [2**(psize-i-1) for i in range(psize)]
f, ax = plt.subplots(1,psize, gridspec_kw={'width_ratios': size_list})

# do not change this (turns off the axes)
for a in ax.ravel():
    a.axis('off')

for l in range(psize):
    ax[l].set_title('l=%d'%l)
    J = Pyr[l]
    ax[l].imshow(J[:,:,::-1], interpolation='none');

plt.show()
```

Multiscale corner detection:

We have 3 images saved to the files **kntu1.jpg**, **kntu2.jpg** and **kntu4.jpg**. All images are of the same size (800 by 800 pixels). However, the logo in images kntu2.jpg and kntu4.jpg are, respectively, 2 and 4 times smaller than the logo in kntu1.jpg.



kntu1.jpg kntu2.jpg kntu4.jpg

We want to find the correct window size for detecting Harris corners in each image. For this, we run the Harris corner detection algorithm for window sizes 2, 4, 8, 16, 32, and 64 for each image. Run the following code and find a proper window size for detecting corners in **kntu1.jpg**. The logo has **78** corners.

File: **multiscale_corner.py**

```
import cv2
import numpy as np

I = cv2.imread('kntu1.jpg')
G = cv2.cvtColor(I,cv2.COLOR_BGR2GRAY)
G = np.float32(G)

for k in range(1,7):
    win_size = 2**k # 2^k
    soble_kernel_size  = 3 # kernel size for gradients
    alpha = 0.04
    H = cv2.cornerHarris(G,win_size,soble_kernel_size,alpha)
    H = H / H.max()

    C = np.uint8(H > 0.01) * 255
    nc,CC = cv2.connectedComponents(C);

    J = I.copy()
    J[C != 0] = [0,0,255]
    cv2.putText(J,'winsize=%d, corners=%d'%(win_size, nc-1),(20,40), \
                cv2.FONT_HERSHEY_SIMPLEX, 1,(0,0,255),2)

    cv2.imshow('corners',J)
```

```
    if cv2.waitKey(0) & 0xFF == ord('q'):
        break
```

- What is the first win_size for which the algorithm detects the right number of corners?
- Test the above code on kntu2.jpg and kntu4.jpg. In each case take note of the **smallest win_size** for which the algorithm correctly detects the corners. What do these numbers say? when the image scales down by factor of 2 the window size also must shrink by factor of two

Task 1:

We want to use the above concept to decide the logo is how many times larger or smaller in one image compared to the other. To do this, we write a function named `first_correct_winsize` which finds the smallest window size (as a power of 2, i.e. 2,4,8,16,32,64) that correctly detects all the **78** corners. By comparing the window size for two images you can compare the sizes of the logos. Just fill the function body and leave the rest of the code unchanged.

File: **task1.py**

```
import cv2
import numpy as np

NO_CORNERS = 78

def first_correct_winsize(I):
    "find the smallest win_size for which all corners are detected"
    # write your code here


    return 4 # incorrect


I1 = cv2.imread('kntu1.jpg')
I2 = cv2.imread('kntu4.jpg')

s1 = first_correct_winsize(I1)
s2 = first_correct_winsize(I2)

J = np.concatenate((I1,I2), 1)

if s1 < s2:
    txt = 'Logo 1 is %d times smaller than logo 2'%(s2/s1)
elif s1 > s2:
    txt = 'Logo 1 is %d times larger than logo 2'%(s1/s2)
else:
    txt = 'Logo 1 is about the same size as logo 2'

cv2.putText(J,txt,(20,40), \
                cv2.FONT_HERSHEY_SIMPLEX, 1,(0,0,255),2)
```

```
cv2.imshow('scale',J)
cv2.waitKey(0)
```

Multiscale corner detection with image pyramid

An alternative approach is to keep the Harris window size fixed and change the image size instead. In the following, we use a fixed window size (win_size = 4) and run the corner detection algorithm for different image sizes in an image pyramid. Run the code and see the result. Then move on quickly to **task 2**.

File: **multiscale_corner_pyramid.py**

```
import cv2
import numpy as np

I = cv2.imread('kntu1.jpg')

psize = 6 # size of the pyramid (no. of levels)

# building the pyramid
J = I.copy()
Pyr = [J] # the first element is simply the original image
for i in range(psize-1):
    J = cv2.pyrDown(J) # blurs, then downsampled by a factor of 2
    Pyr.append(J)

for k in range(psize): # k = 0,1,..., psize-1
    J = Pyr[k]
    G = cv2.cvtColor(J,cv2.COLOR_BGR2GRAY)
    G = np.float32(G)

    win_size = 4 # do not change this
    soble_kernel_size  = 3 # kernel size for gradients
    alpha = 0.04
    H = cv2.cornerHarris(G,win_size,soble_kernel_size,alpha)
    H = H / H.max()

    C = np.uint8(H > 0.01) * 255
    nc,CC = cv2.connectedComponents(C);

    J[C != 0] = [0,0,255]

    JJ = np.zeros(I.shape,dtype=I.dtype)
    JJ[:J.shape[0],:J.shape[1],:] = J;
    cv2.putText(JJ,'scale=1/%d, corners=%d'%(2**k, nc-1),(360,30), \
                cv2.FONT_HERSHEY_SIMPLEX, 1,(0,0,255),2)

    cv2.imshow('corners',JJ)
```

```
    if cv2.waitKey(0) & 0xFF == ord('q'):
        break
```

Task 2

The job is the same as **task 1**. But this time we do it using an image pyramid. This time you need to write a function called **`first_correct_scale`** which finds the first image scale in a pyramid that correctly detects all the **78** corners. By comparing the image scales for the two images you will find their relative size. Notice that you just need to add a little piece of code and fill in the function body after the line saying **`#! write your code here! *********************`**

File: **task2.py**

```
import cv2
import numpy as np

NO_CORNERS = 78

def first_correct_scale(I):
    "find the smallest scale for which all corners are detected"

    psize = 6 # size of the pyramid

    # building the pyramid
    J = I.copy()
    Pyr = [J] # the first element is simply the original image
    for i in range(psize-1):
        J = cv2.pyrDown(J) # blurs, then downsampled by a factor of 2
        Pyr.append(J)

    for k in range(psize): # k = 0,1,..., psize-1
        J = Pyr[k]
        G = cv2.cvtColor(J,cv2.COLOR_BGR2GRAY)
        G = np.float32(G)

        win_size = 4 # do not change this!!
        soble_kernel_size  = 3 # kernel size for gradients
        alpha = 0.04

        #! write your code here! *****************************************
        nc = 79 # !!! delete this line!

        if nc-1 == NO_CORNERS: # if the connected components
            return 2**k

I1 = cv2.imread('kntu1.jpg')
I2 = cv2.imread('kntu4.jpg')

sc1 = first_correct_scale(I1)
sc2 = first_correct_scale(I2)
```

```
J = np.concatenate((I1,I2), 1)

if sc1 < sc2:
    txt = 'Logo 1 is %d times smaller than logo 2'%(sc2/sc1)
elif sc1 > sc2:
    txt = 'Logo 1 is %d times larger than logo 2'%(sc1/sc2)
else:
    txt = 'Logo 1 is about the same size as logo 2'

cv2.putText(J,txt,(20,40), \
                cv2.FONT_HERSHEY_SIMPLEX, 1,(0,0,255),2)
cv2.imshow('scale',J)
cv2.waitKey(0)
```

Have some fun!

[This is not part of your lab] Run the next code and see the result. You will learn how to do this soon. For the time being, just run it. You can use your own photo.

File: **create_pyramid.py**

```
import numpy as np
import cv2

I = cv2.imread('toosi.jpg')
m,n,_ = I.shape

P1 = np.array([[0,0], [0, m-1], [n-1,0], [n-1,m-1]])

psize = 7 # size of the pyramid (no. of levels)

J = np.ones((600,500,3), dtype=np.uint8)*255
m2,n2,_ = J.shape

v  = np.array([(n2//2,0)])
P2 = np.array([(0,4*m2//5),
               (5*n2//6,m2),
               (3*n2//12,7*m2//12),
               (n2,8*m2//12)])

cv2.line(I, (0,0), (0,m-1), (1,1,1),4)
cv2.line(I, (0,0), (n-1,0), (1,1,1),4)
cv2.line(I, (n-1,m-1), (0,m-1), (1,1,1),4)
cv2.line(I, (n-1,m-1), (n-1,0), (1,1,1),4)

for i in range(4):
    cv2.line(J, (v[0,0],v[0,1]), (P2[i,0],P2[i,1]), (0,0,0),2)

p21 = P2[1].copy()

for i in range(psize):
    H, status = cv2.findHomography(P1, P2)
    K = cv2.warpPerspective(I, H, (J.shape[1],J.shape[0]))
    msk = K.max(axis=2) != 0
```

```
    J[msk,:] = K[msk,:]

    cv2.line(J, (v[0,0],v[0,1]), (p21[0],p21[1]), (0,0,0),2)
    cv2.imshow('',J)
    cv2.waitKey()
    P2 = (P2 + v)/2
```

References

- [OpenCV-Python Tutorials - Image Pyramids]()