# Lab Instructions - session 11

## SIFT detection, description, and matching

The following code detects SIFT key points for a sequence of images and displays their location. The function (method) `sift.detect` returns a list of keypoints and the function `drawKeypoints` displays them. Run the code to see the results. Press any key to see the next image. Press 'q' to quit.

File: **sift_detect.py**

```
import numpy as np
import cv2
import glob

for fname in glob.glob('*.jpg'):
    I = cv2.imread(fname)
    G = cv2.cvtColor(I,cv2.COLOR_BGR2GRAY)

    #sift = cv2.FeatureDetector_create("SIFT") # opencv 2.x.x
    sift = cv2.xfeatures2d.SIFT_create() # opencv 3.x.x
    # if the above fails use "sift = cv2.SIFT()"

    keypoints = sift.detect(G,None)

    cv2.drawKeypoints(G,keypoints,I)
    # cv2.drawKeypoints(G,keypoints,I, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

    # display keypoint properties
    # for kp in keypoints:
    #     print('-'*40)
    #     print('location=(%.2f,%.2f)'%(kp.pt[0], kp.pt[1]))
    #     print('orientation angle=%1.1f'%kp.angle)
    #     print('scale=%f'%kp.size)

    cv2.putText(I,"Press 'q' to quit, any key for next image",(20,20), \
                cv2.FONT_HERSHEY_SIMPLEX, .5,(255,0,0),1)

    cv2.imshow('sift_keypoints',I)

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

### Noise and artifacts, thresholding sensivity

What do you think about the location of the sift features? Some of them are located in the flat regions (e.g. sky, etc.) of the image. Why do you think this happens?

To get more insight about the detected SIFT keypoints, add the flag `cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS` to the `cv2.drawKeypoints` function:

```
cv2.drawKeypoints(G,keypoints,I,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
```

When assigning orientation, we use a threshold on HoG, which means multiple orientations could be selected.

Now, for each feature, you can see the natural scale and the assigned orientation.

- As you can see some features have two or more orientations. These are in fact different features with the same location and scale (but different orientation). Can you tell why this happens from what you learned in the class?
- Can you tell now why some keypoints are located in the flat regions of the image? Notice that the circle drawn for each feature is only proportional to the scale region for which sift keypoint is detected. The actual region might span a larger area.

A low threshold in keypoint localization. Large area so at the edges of region there might be a keypoint.

To get information about the key points, you can access the fields of the keypoint object. Uncomment the for loop in your code to print some information about each keypoint:

```
#! print keypoint properties
for kp in keypoints:
   print('-'*40)
   print('location=(%.2f,%.2f)'%(kp.pt[0], kp.pt[1]))
   print('orientation angle=%1.1f'%kp.angle)
   print('scale=%f'%kp.size)
```

- You can see that the location of each keypoint (`kp.pt`) does not have integer coordinates (the feature is not exactly located at a pixel). Can you tell why from what you learned in the class?

It's doing keypoint localization, by fitting a quadratic curve to the points in scale-space, then finding the exterma.

# Towards feature matching

We have taken pictures of a book from two different perspectives (*book1.jpg* and *book2.jpg*). As you can see, the orientation and scale of the book are different in the two images. Let us detect the SIFT features and compare them.


Run the following piece of code to compare the sift features in the book.
File: **sift_compare.py**

```
import numpy as np
import cv2

I1 = cv2.imread('book1.jpg')
G1 = cv2.cvtColor(I1,cv2.COLOR_BGR2GRAY)

I2 = cv2.imread('book2.jpg')
G2 = cv2.cvtColor(I2,cv2.COLOR_BGR2GRAY)

#sift = cv2.FeatureDetector_create("SIFT") # opencv 2.x.x
sift = cv2.xfeatures2d.SIFT_create() # opencv 3.x.x
# if the above fails use "sift = cv2.SIFT()"

keypoints1 = sift.detect(G1,None)
keypoints2 = sift.detect(G2,None)

cv2.drawKeypoints(G1,keypoints1,I1, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
cv2.drawKeypoints(G2,keypoints2,I2, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

# concatenate the two images
I = np.concatenate((I1,I2), axis=1)

cv2.imshow('sift_keypoints1',I)
cv2.waitKey()
```

- Can you detect feature points in both images whose location, scale and orientation match?
- Can you detect keypoints with two or more orientations such that only one of the orientations matches with a keypoint in the other image? Look at the following patch for example:

# Feature description and matching

Now, we want to do what you did with your eyes in the previous example automatically. First, run the following code and see the output.

File: **sift_match.py**

```
import numpy as np
import cv2

I1 = cv2.imread('book2.jpg')
G1 = cv2.cvtColor(I1,cv2.COLOR_BGR2GRAY)

I2 = cv2.imread('scene.jpg')
G2 = cv2.cvtColor(I2,cv2.COLOR_BGR2GRAY)

sift = cv2.xfeatures2d.SIFT_create() # opencv 3
# use "sift = cv2.SIFT()" if the above fails

keypoints1 = sift.detect(G1,None)
keypoints2 = sift.detect(G2,None)

cv2.drawKeypoints(G1,keypoints1,I1, \
    flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
cv2.drawKeypoints(G2,keypoints2,I2, \
    flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

print("No. of keypoints1 =", len(keypoints1))
print("No. of keypoints2 =", len(keypoints2))

# compute descriptor vectors
keypoints1, desc1 = sift.compute(G1, keypoints1); # opencv 3
keypoints2, desc2 = sift.compute(G2, keypoints2); # opencv 3

print("Descriptors1.shape =", desc1.shape)
print("Descriptors2.shape =", desc2.shape)

# pause here!!
input("Press ENTER to continue...")

# brute-force matching
bf = cv2.BFMatcher(crossCheck=False)

# for each descriptor in desc1 find its
# two nearest neighbors in desc2
matches = bf.knnMatch(desc1,desc2, k=2)
good_matches = []
alpha = 0.75
for m1,m2 in matches:
    # m1 is the best match
    # m2 is the second best match
    if m1.distance < alpha *m2.distance:
```

```
        good_matches.append(m1)

I = cv2.drawMatches(I1,keypoints1,I2,keypoints2, good_matches, None)

cv2.imshow('sift_keypoints1',I)
cv2.waitKey()
```

What is the output of the following lines?

```
print("No. of keypoints1 =", len(keypoints1))
print("No. of keypoints2 =", len(keypoints2))
```

- How many keypoints have been found in each image?

The 2D numpy arrays desc1 and desc2 represent the set of descriptors for each image. The following line prints their dimensions.

```
print("Descriptors1.shape =", desc1.shape)
print("Descriptors2.shape =", desc2.shape)
```

- How do you interpret the shapes of desc1 and desc2?

The line `matches = bf.knnMatch(desc1,desc2, k=2)` for each keypoint descriptor in the first image finds its *best match* and its *second best match* among the descriptors of the second image (`desc2`). The line `if m1.distance < alpha * m2.distance` makes the algorithm only accept the best match if the distance to the first match is smaller than **alpha** times distance to the second match, where alpha = 0.75 here.

- Change the value of **alpha** and see the result. Explain how changing alpha affects the result.

Reducing the alpha, reduces the number of matches.
The ratio test helps ensure that the match m1 is significantly better than the next best alternative m2.

Instead of

```
keypoints1 = sift.detect(G1,None)
keypoints1, desc1 = sift.compute(G1, keypoints1);
```

you can combine the two steps and write:

```
keypoints1, desc1 = sift.detectAndCompute(G1,None)
```

- Why do you think SIFT detection and description are also implemented as separate functions? Where `sift.detect` or `sift.compute` can be useful?

It gives us modularity and freedom to mix and match different descriptors and keypoint detection methods.
For example we may not want to use SIFT keypoint detection but instead Harris

# Task 1

We have pictures of a bunch of objects (files *obj1.jpg, obj2.jpg, …, obj9.jpg*). Your task is to find out if the object exists in the scene image (file **scene.jpg**). The algorithm is to find the matched feature points between each object and the scene image. If the number of good matches is greater than a certain threshold (e.g. 30 matches) you decide that the object is present in the scene. You need to complete the following code to accomplish your task.

file: **task1.py**

```
import numpy as np
import cv2
import glob

sift = cv2.xfeatures2d.SIFT_create() # opencv 3
# use "sift = cv2.SIFT()" if the above fails

I2 = cv2.imread('scene.jpg')
G2 = cv2.cvtColor(I2,cv2.COLOR_BGR2GRAY)
keypoints2, desc2 = sift.detectAndCompute(G2, None);

fnames = glob.glob('obj?.jpg')
fnames.sort()
for fname in fnames:

    I1 = cv2.imread(fname)
    G1 = cv2.cvtColor(I1,cv2.COLOR_BGR2GRAY)
    keypoints1, desc1 = sift.detectAndCompute(G1, None);
    cv2.drawKeypoints(G1,keypoints2,I1)

    good_matches = []

    I = cv2.drawMatches(I1,keypoints1,I2,keypoints2,good_matches, None)

    no_matches = len(good_matches)
    if no_matches > 30:
        txt = "Object found! (matches = %d)"%no_matches
    else:
        txt = "Object not found! (matches = %d)"%no_matches

    cv2.putText(I,txt,(20,40),cv2.FONT_HERSHEY_SIMPLEX, 1,(255,0,0),3)

    cv2.imshow('keypoints',I)

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

# References

- OpenCV-Python Tutorials - SIFT