Lab Instructions - session 12

Feature-based Image Alignment, RANSAC

Robust estimation with RANSAC

Remember how SIFT feature points were matched in two images. Look at the code below to remind yourself. Run the file and see the result.

File: **sift_match_ransac.py**

```python
import numpy as np
import cv2

I1 = cv2.imread('obj3.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

# detect keypoints and compute their descriptor vectors
keypoints1, desc1 = sift.detectAndCompute(G1, None); # opencv 3
keypoints2, desc2 = sift.detectAndCompute(G2, None); # opencv 3

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

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

# stop here!!
exit() # comment this line out to move on!

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

# 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)
```

```
# apply RANSAC
# points1 = [keypoints1[m.queryIdx].pt for m in good_matches]
# points1 = np.array(points1,dtype=np.float32)

# points2 = [keypoints2[m.trainIdx].pt for m in good_matches]
# points2 = np.array(points2,dtype=np.float32)
# H, mask = cv2.findHomography(points1, points2, cv2.RANSAC,5.0) # 5 pixels margin
# mask = mask.ravel().tolist()
# print(mask)

good_matches = [m for m,msk in zip(good_matches,mask) if msk == 1]

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

cv2.imshow('sift_keypoints1',I)
cv2.waitKey(0)
```

As you can see, even after removing a number of false matches using the distance ratio test (`m1.distance < alpha *m2.distance`), some of the mismatches still persist. We want to remove them using RANSAC. Since we know that the transformation between the source and target objects is a **homography,** we can use the function `cv2.findHomography` with the option `cv2.RANSAC` to remove the outliers and estimate a homography `H`. Uncomment the following lines from the code. It redefines the list **good_matches** by further removing the outliers using RANSAC. Rerun the code to draw the new "good_matches" and see the result.

```
# apply RANSAC
points1 = [keypoints1[m.queryIdx].pt for m in good_matches]
points1 = np.array(points1,dtype=np.float32)

points2 = [keypoints2[m.trainIdx].pt for m in good_matches]
points2 = np.array(points2,dtype=np.float32)

H, mask = cv2.findHomography(points1, points2, cv2.RANSAC,5.0)

mask = mask.ravel().tolist()

good_matches = [m for m,msk in zip(good_matches,mask) if msk == 1]
```

- Remember, for every member **m** of the list **good_matches**, **m.queryIdx** and **m.trainIdx** were the indices of the key points in the first and second images respectively. What is `keypoints1[m.queryIdx].pt`? What are the shapes of `points1` and `points2`?
- Print the variable `mask`. What does it represent?
- What does the following line do?
  ```
  good_matches = [m for m,msk in zip(good_matches,mask) if msk == 1]
  ```

Task 1: Image alignment

We want to transform the first image to its location in the second image. This can be done using the **cv2.warpPerspective** function you used before and the transformation matrix **H** obtained using RANSAC. Complete the following code to do this. The code alternatingly displays the second image and the transformed source object. In the current code, the homography **H** is just the identity matrix. You need to set it to the correctly estimated homography.

File: **task1.py**

```python
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); # opencv 3

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); # opencv 3

    H = np.eye(3,dtype=np.float32) # this needs to be changed!!

    J = cv2.warpPerspective(I1, H, (I2.shape[1],I2.shape[0]) )

    # alternatingly show images I2 and J
    ind = 0;
    imgs = [I2, J]
    while 1:
        ind = 1-ind

        cv2.imshow('Reg',imgs[ind])
        key =  cv2.waitKey(800)

        if key & 0xFF == ord('q'):
            exit()
        elif key & 0xFF != 0xFF:
            break
```

Task 2: Draw object outline

In this task, we want to draw the outline of each object in the scene. The outline (a quadrilateral) can be drawn by drawing four lines with the **cv2.line** function. The four corners of the first image can be transformed to their locations in the second image using the function **cv2.perspectiveTransform** (look at the code). Notice that this function accepts the points in the form of **(x,y)** and not **(y,x)**. Complete the following to perform Task 2.

File: **task2.py**

```python
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); # opencv 3

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

    I1 = cv2.imread(fname)
    G1 = cv2.cvtColor(I1,cv2.COLOR_BGR2GRAY)
```

```python
    keypoints1, desc1 = sift.detectAndCompute(G1, None); # opencv 3

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

    # for each descriptor in desc1 find its
    # two nearest neighbors in desc2
    matches = bf.knnMatch(desc1,desc2, k=2)

    # distance ratio test
    alpha = 0.75
    good_matches=[m1 for m1,m2 in matches if m1.distance < alpha *m2.distance]

    points1 = [keypoints1[m.queryIdx].pt for m in good_matches]
    points1 = np.array(points1,dtype=np.float32)

    points2 = [keypoints2[m.trainIdx].pt for m in good_matches]
    points2 = np.array(points2,dtype=np.float32)

    H, mask = cv2.findHomography(points1, points2, cv2.RANSAC,5.0)

    H = np.eye(3,dtype=np.float32) # this needs to be changed
    pts = np.float32([[100,100],
                      [100,400],
                      [400,400],
                      [400,100] ]).reshape(-1,1,2) # this needs to be changed

    dst = cv2.perspectiveTransform(pts,H).reshape(4,2)

    J = I2.copy()
    cv2.line(J, (dst[0,0], dst[0,1]), (dst[1,0], dst[1,1]), (255,0,0),3)
    cv2.line(J, (dst[1,0], dst[1,1]), (dst[2,0], dst[2,1]), (255,0,0),3)
    cv2.line(J, (dst[2,0], dst[2,1]), (dst[3,0], dst[3,1]), (255,0,0),3)
    cv2.line(J, (dst[3,0], dst[3,1]), (dst[0,0], dst[0,1]), (255,0,0),3)


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

    cv2.imshow('keypoints',I)

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

- What is the purpose of using the `cv2.perspectiveTransform()` function in this code?
- Why is the SIFT feature detector used in this task, and what are its advantages over other methods?
- What is the difference between `queryIdx` and `trainIdx` in a `cv2.DMatch` object?