Assignment 1(Programming)

ID	Name
1953902	高杨帆

1. Scale Invariant Detection

a. Ideas & Results

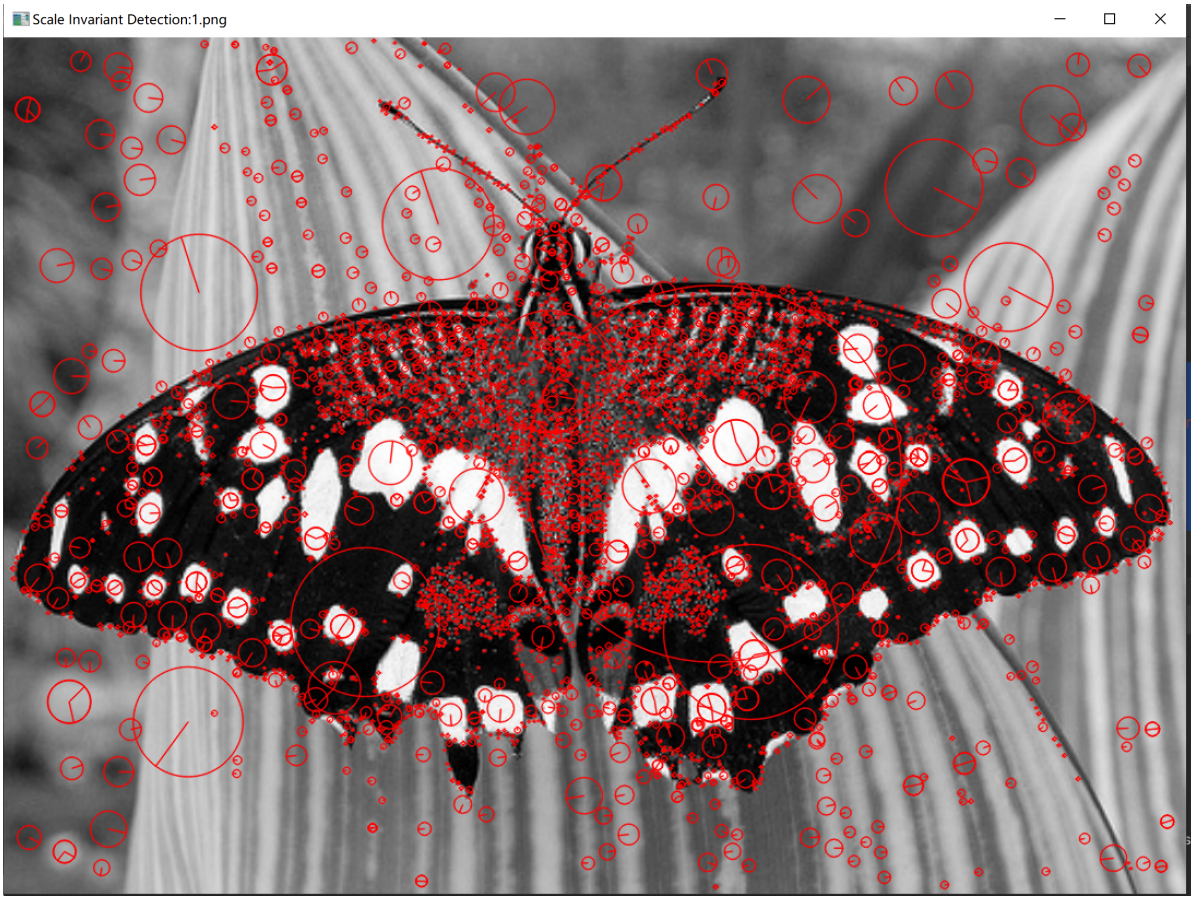
I have provided 2 solutions to this problem:

Using OpenCV API

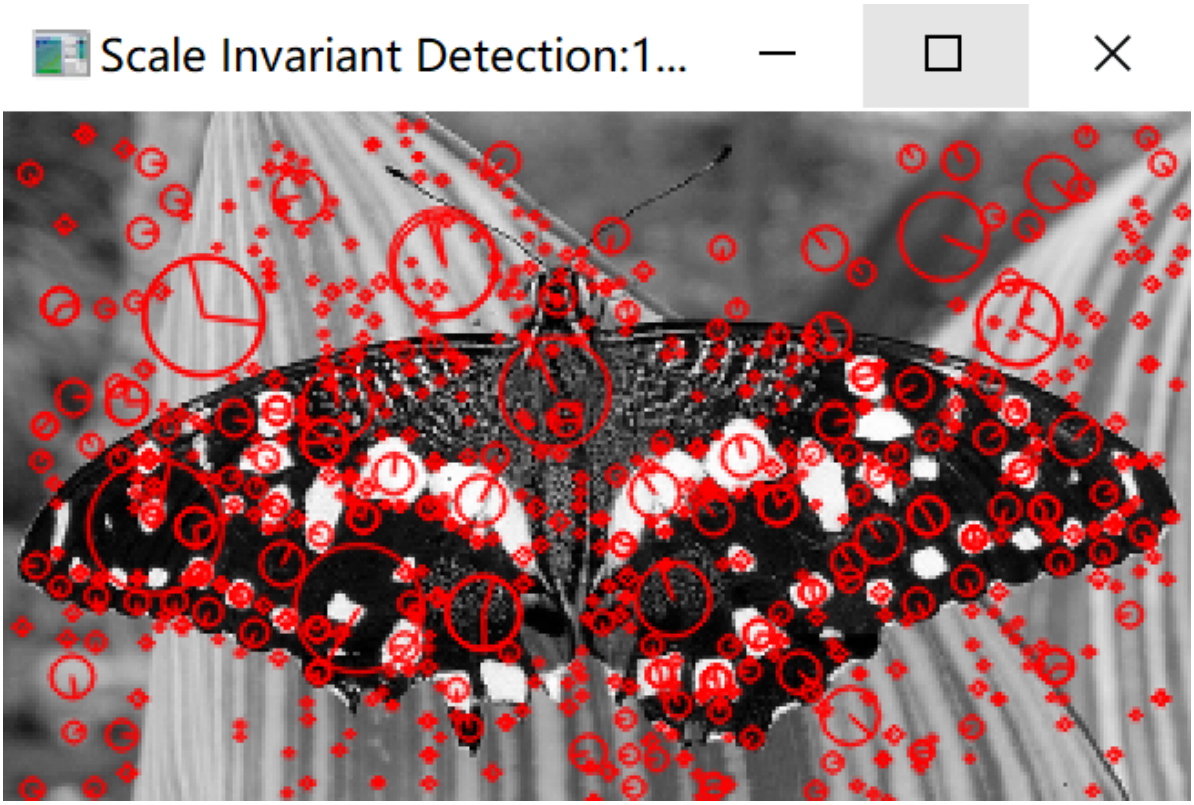
The idea is to use OpenCV API to do key point detection and draw the key points(with the dominant orientation) on the image:

```
sift = cv2.xfeatures2d.SIFT_create()
        keypoint, descriptor = sift.detectAndCompute(img, None)
        img = cv2.drawKeypoints(image=img, keypoints=keypoint, outImage=img,
color=(0, 0, 255), flags=cv2.
                                    DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
        # cv2.imwrite('./img/' + img_name + '_sid.png', img)
        cv2.imshow(f'Scale Invariant Detection:{img_name}', img)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
```

the result is like this:



when the image is scaled:



Using my own implementation

This is my own implementation of SIFT key point detection, which is a little bit different from what Prof. Zhang mentioned in the lecture. Here are the main steps:

1. construct the DoG scale space by the method introduced in https://www.cnblogs.com/zyly/p/9519011.html
2. detect the scale invariant pixel by pixel using the method mentioned in the lecture

3. draw the scale invariant points on the image using OpenCV API

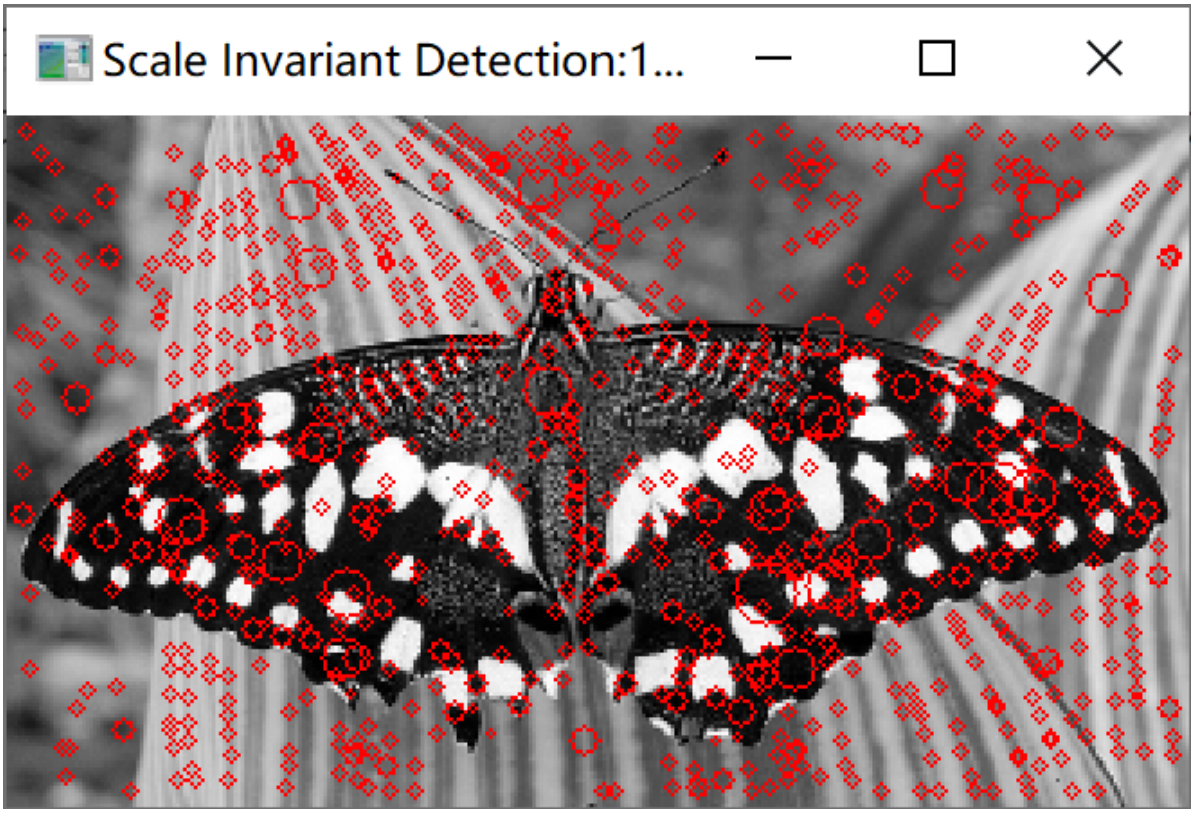
Source code of these steps is in [main.py]

result of my own implementation seems a little bit weird:

when the image is scaled:

(2022/4/6 update) I have done some more research on this topic and I discovered that I have implemented the convolution part wrongly(I just considered it as element-wise production of matrices). But after re-implemented that, the result is even worse than the previous version:

(2022/4/8 update) I slightly modified the implementation of convolution and DoG construction, and the result became better:



for the unscaled one:

That is more similar to the

b. How to run this

1. `cd scale-invariant-detect`
2. `pip install -r requirements.txt`
3. `python main.py name_of_your_image` **note that the images shall be stored in** `./img` **directory I provided**

Optional arguments:

`--use_cv` use the OpenCV API if true, else use my implementation. Default `True`

`--sigma0` default `1.52` , you can tune this parameter to test the influence of different sigma values(i.e., standard deviation) on the detection

example:

`python main.py 1.png --use_cv false`

2. Panorama Stitching

a. Ideas & Results

just do as what Prof. Zhang mentioned in the lecture:

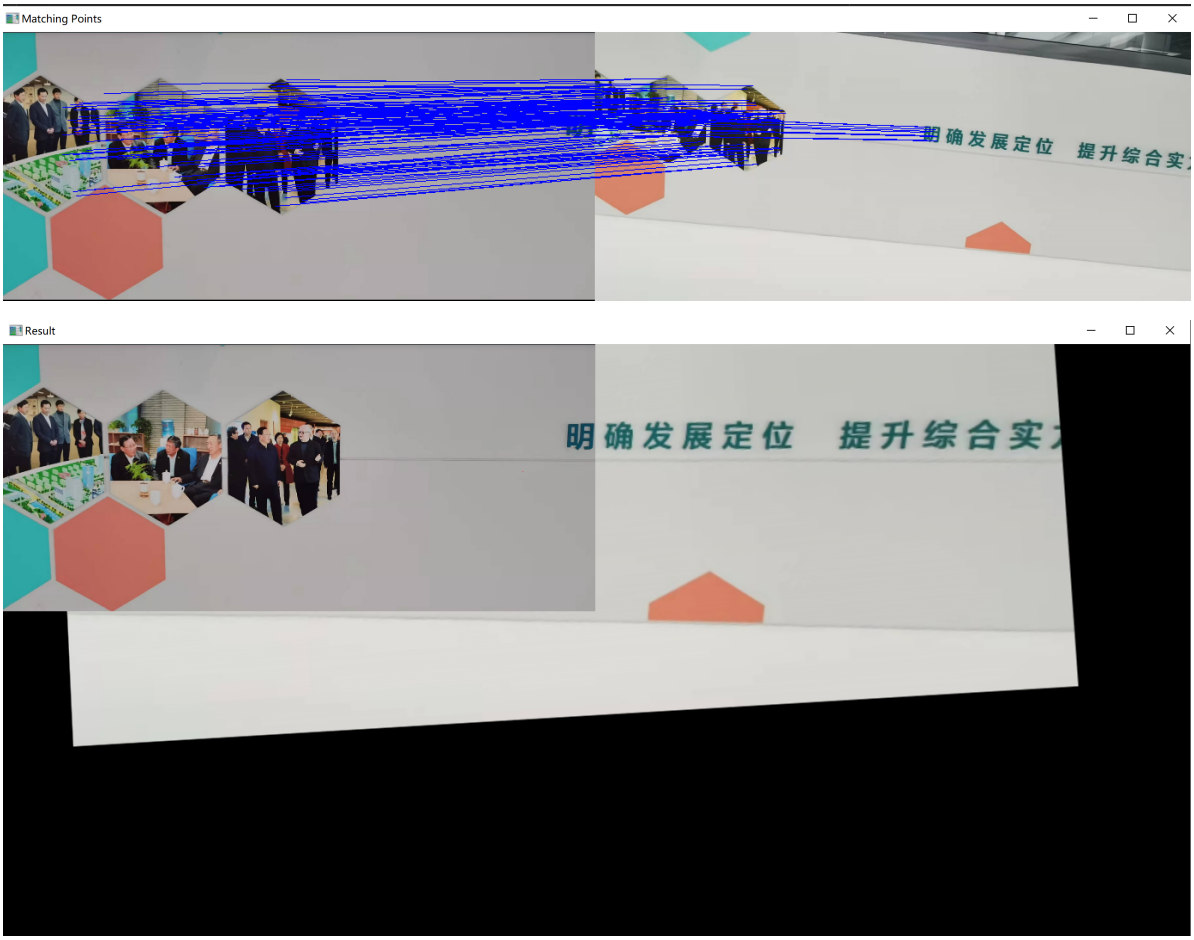
1. find key points
2. construct descriptors for all key points
3. build correspondence for key points
4. solve H
5. apply H^{-1} to image 2
6. stitch them together

Note that in step 3 I used KNN algorithm to build correspondence of points in 2 images.

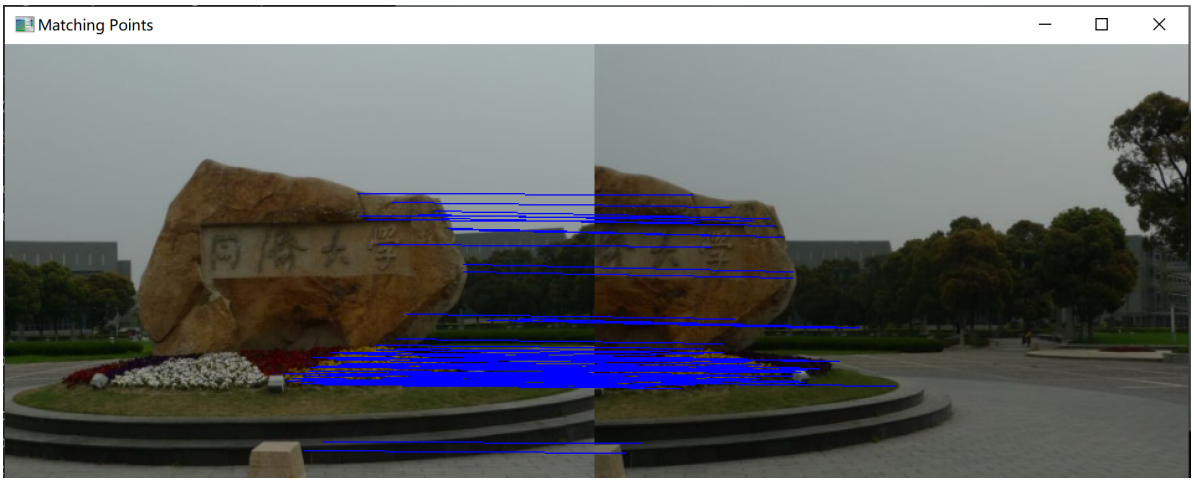
Concrete code can be found in [main.py]

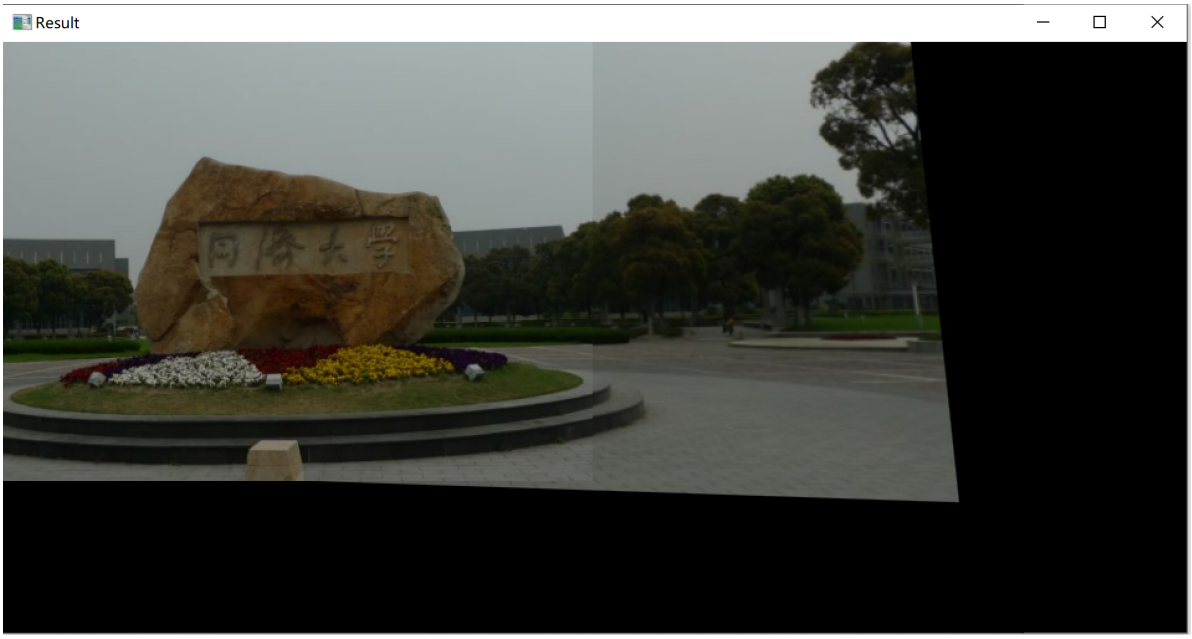
Results of different examples in the lecture:

1.

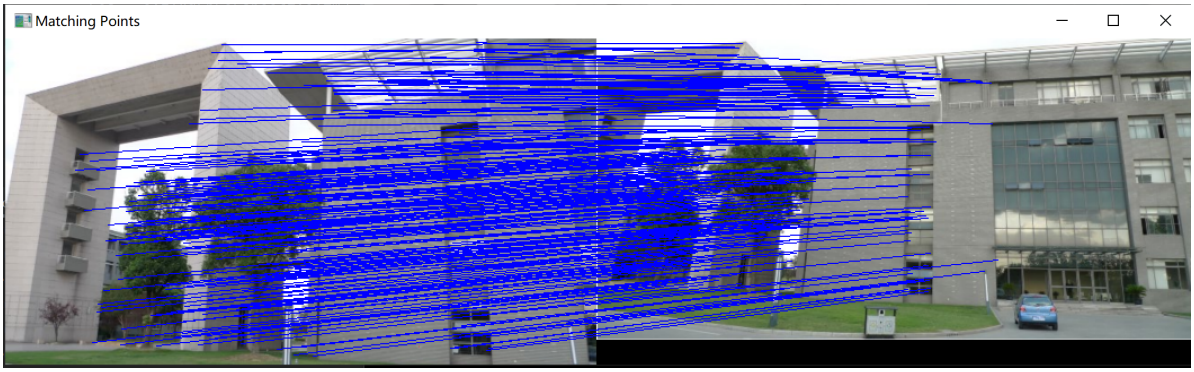


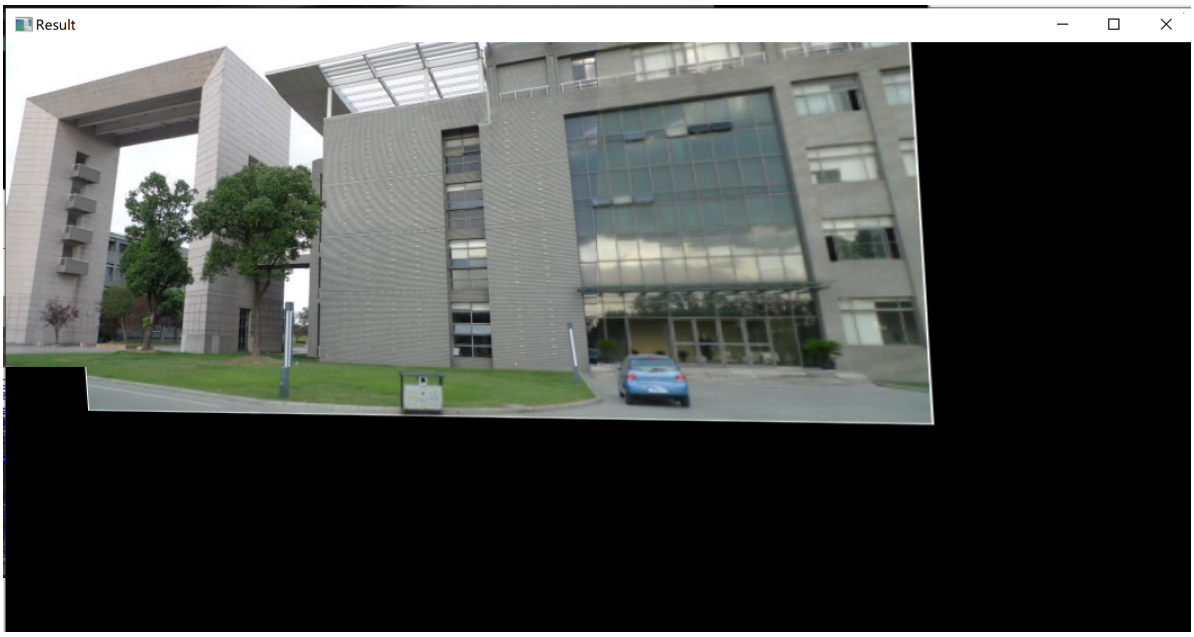
2.





3.





b. How to run this

1. `cd panorama-stitching`
2. `pip install -r requirements.txt`
3. `python main.py name_of_1st_image name_of_2nd_image` **note that the images shall be stored in** `./img` **directory I provided**

Optional arguments:

`--show_match` Show the correspondence of point if true. Default `False`

example:

`python main.py '3-1.png' '3-2.png' --show_match true`

Please name the image that is on the left of the result image lexicographically lower