Kewen Peng

CSC Project 2 Report

In this project, I use my laptop camera to capture video, and extract edges and corners from the frame. After that, I superimposed the features on the frame.

I also use OpenCV to extract key points from a video frame and compare them. I examined the performance of SIFT descriptors relative to translation, rotation, scale and angles.

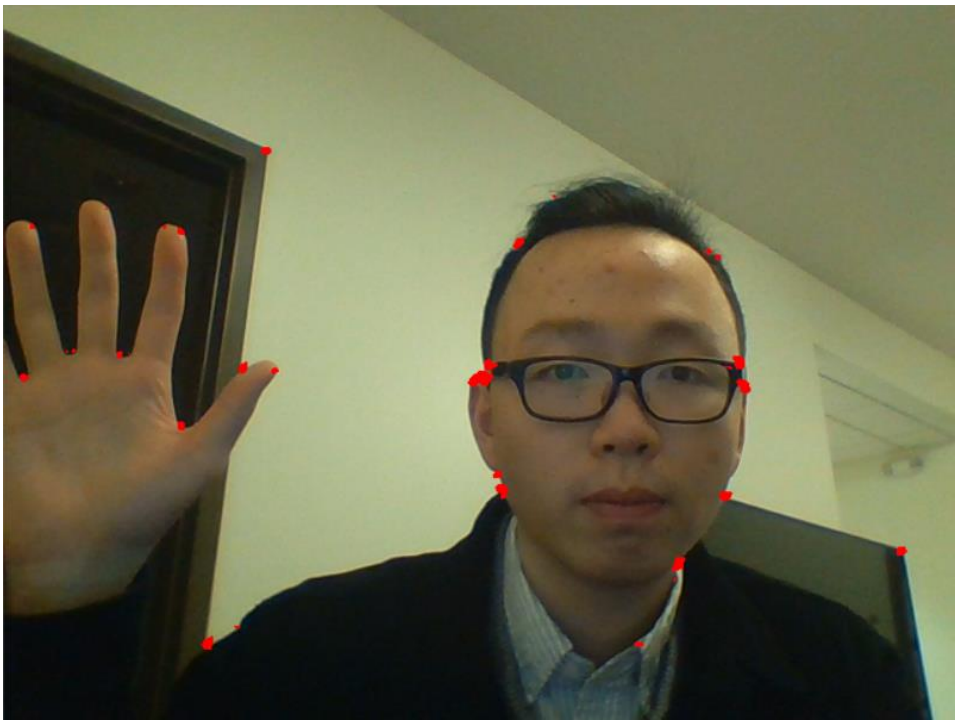
**Part 1 Corner detector**

```python
import numpy as np
import cv2
cap = cv2.VideoCapture(0)
# ########################## cornerHarris
while(cap.isOpened()):
    ret, frame = cap.read()
    if ret==True:
        img=cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
        corner=cv2.cornerHarris(img,7,7,0.04)
        img2=img.copy()
        frame[corner > 0.04 * corner.max()] = [0, 0, 255]
        # out.write(frame)

        cv2.imshow('frame',frame)
        # use Esc as keyboard interrupt
        if cv2.waitKey(1) == 27:
            break
    else:
        break

cap.release()
cv2.destroyAllWindows()

```

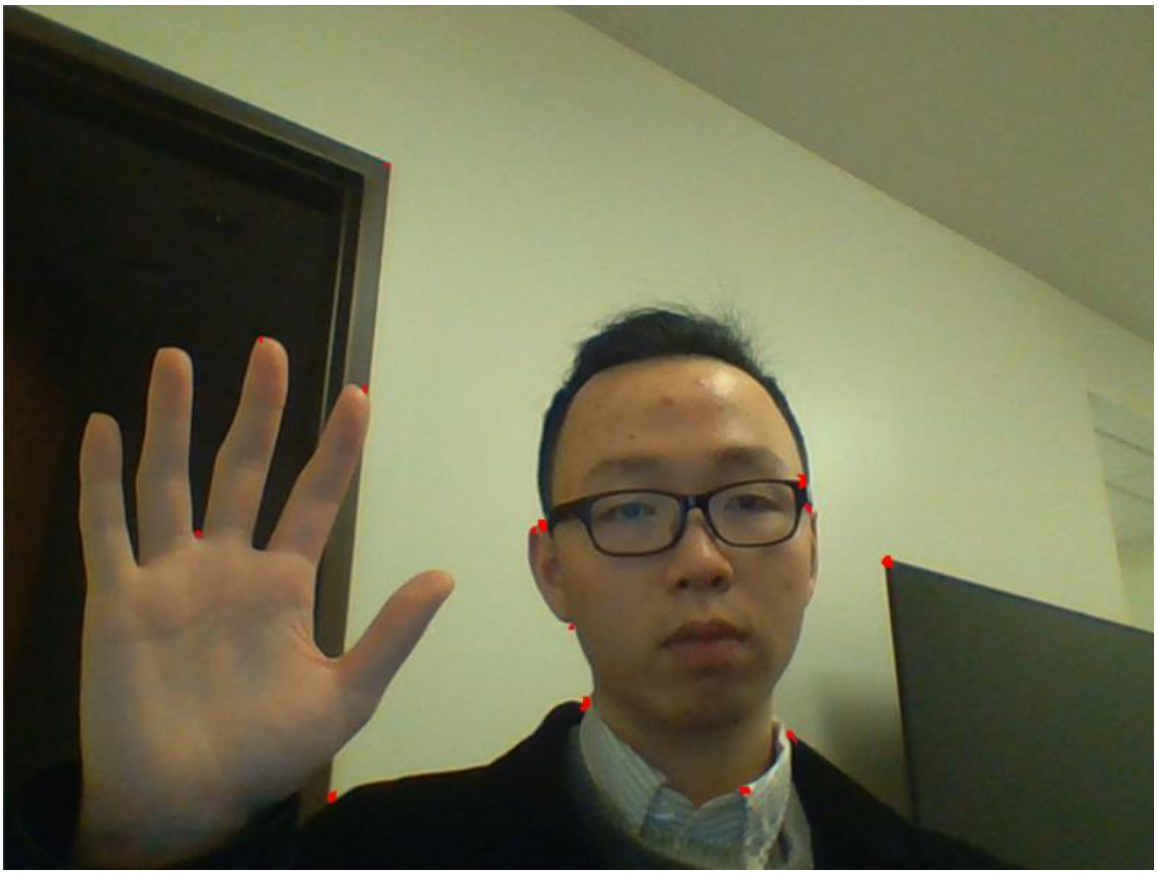


(Good parameter)

Kewen Peng

I use cv2.cornerHarris to detect corners in the frame. The first parameter is the input image. The second parameter 7 is the block size, which means the detector will consider the 7x7 neighborhood of each pixel. The third parameter 7 is the k-size used by the Sobel derivative. The fourth one 0.04 is the free parameter: with a larger k, the method tends to only detect larger/more obvious corners.

The following is when k=0.1, and the method finds fewer corners from a similar scene.



(Under-fitting parameters)

In general, I found cv2.cornerHarris(img,7,7,0.04) to be appropriate parameters because it can sufficiently find valid corners and does not get influenced by lighting, distance, or blurring.

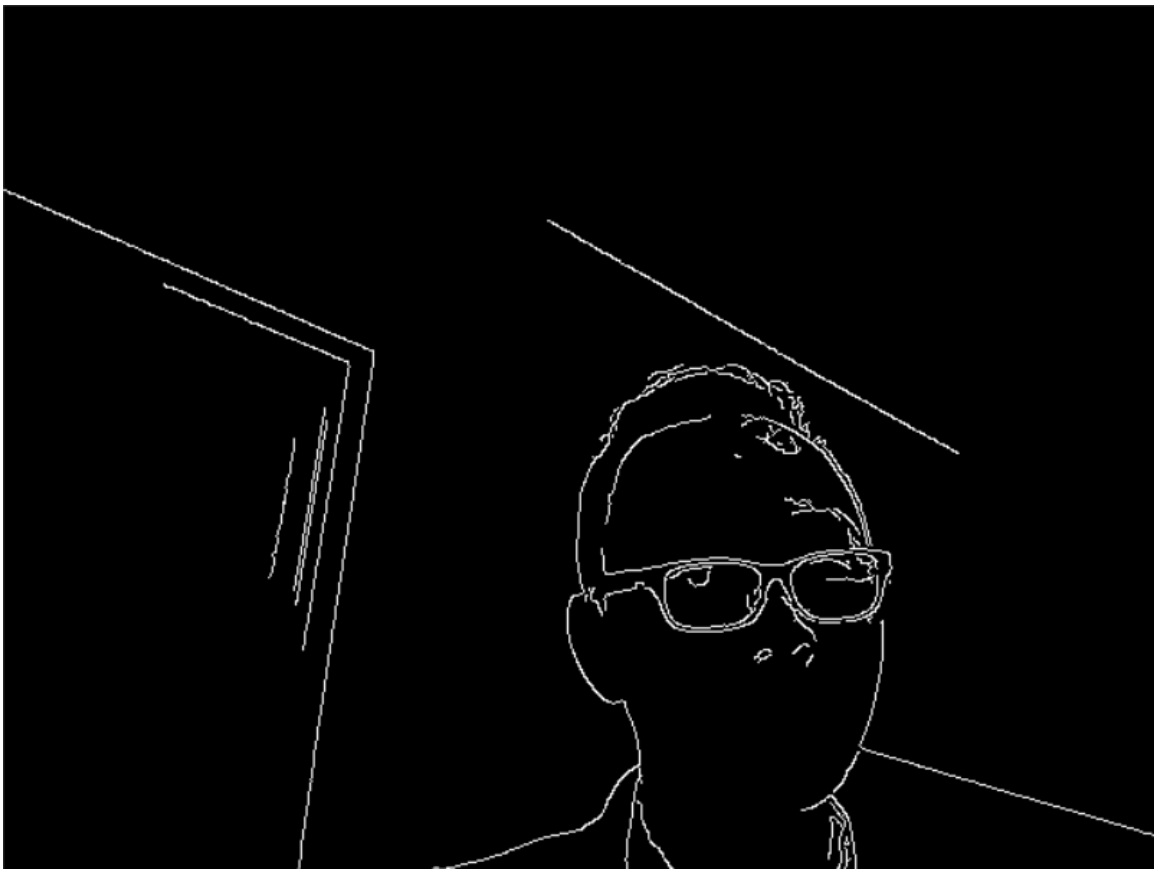
Kewen Peng

**Part 1 Canny edge detection**

```
24      # ########################## canny edge detection
25      while(cap.isOpened()):
26          ret, frame = cap.read()
27          if ret==True:
28              frame = cv2.Canny(frame,100,200)
29              cv2.imshow('frame',frame)
30              # use Esc as keyboard interrupt
31              if cv2.waitKey(1) == 27:
32                  break
33          else:
34              break
35
36      cap.release()
37      cv2.destroyAllWindows()
38
```

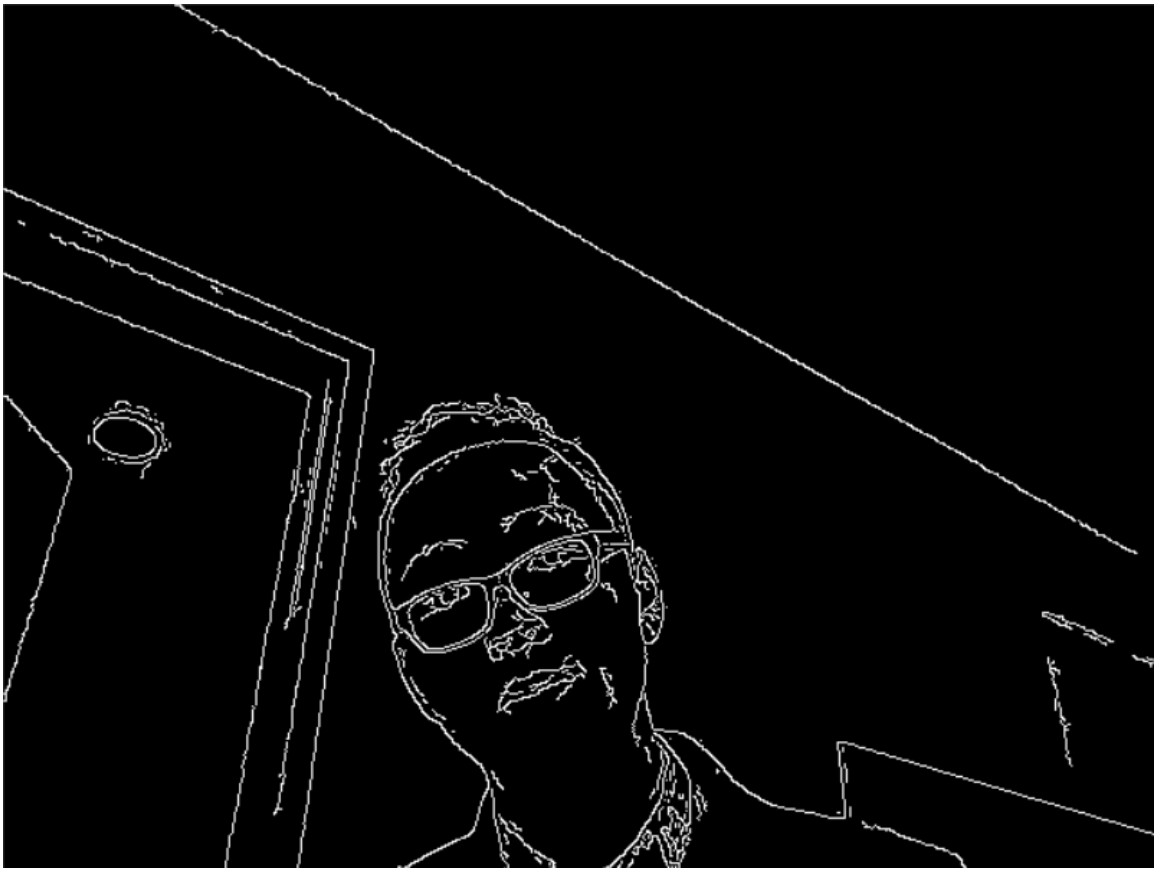
The first parameter is input image; the last 2 are the 2 thresholds used to determine edges.

Good parameters: cv2.Canny(frame,100,200)



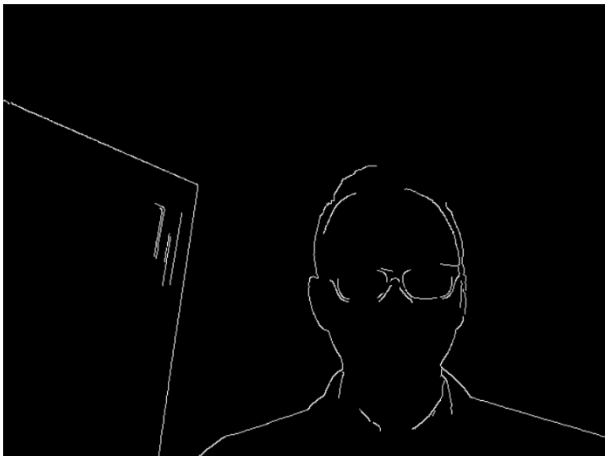
Kewen Peng

Better parameters: cv2.Canny(frame,50,100)

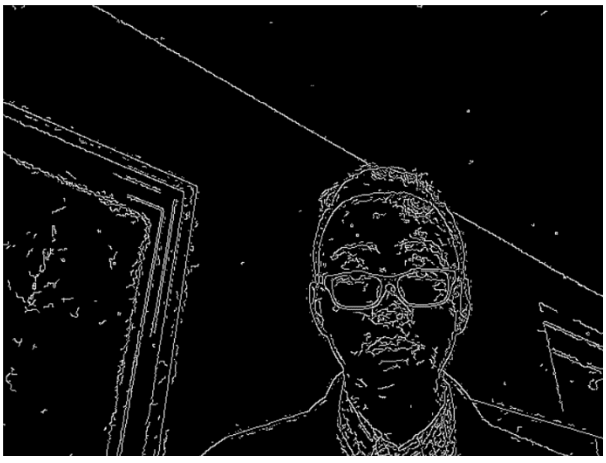


A lower threshold would display more edges, but we need to avoid displaying every discontinuity from the frame.

Not-so-good parameters:



Threshold = (200, 300)



Threshold = (30, 50)

Kewen Peng

**Part 2 SIFT**

**2.1**

I applied cv2.xfeatures2d.SIFT_create() to the image I found online. The method cv.xfeatures2d.SIFT_create(nfeatures, nOctaveLayers, contrastThreshold, edgeThreshold, sigma) has 5 parameters whose meaning I found in the official documentation: https://docs.opencv.org/3.4/d5/d3c/classcv_1_1xfeatures2d_1_1SIFT.html.

**Parameters**

| | |
|---|---|
| nfeatures | The number of best features to retain. The features are ranked by their scores (measured in SIFT algorithm as the local contrast) |
| nOctaveLayers | The number of layers in each octave. 3 is the value used in D. Lowe paper. The number of octaves is computed automatically from the image resolution. |
| contrastThreshold | The contrast threshold used to filter out weak features in semi-uniform (low-contrast) regions. The larger the threshold, the less features are produced by the detector. |
| edgeThreshold | The threshold used to filter out edge-like features. Note that the its meaning is different from the contrastThreshold, i.e. the larger the edgeThreshold, the less features are filtered out (more features are retained). |
| sigma | The sigma of the Gaussian applied to the input image at the octave #0. If your image is captured with a weak camera with soft lenses, you might want to reduce the number. |

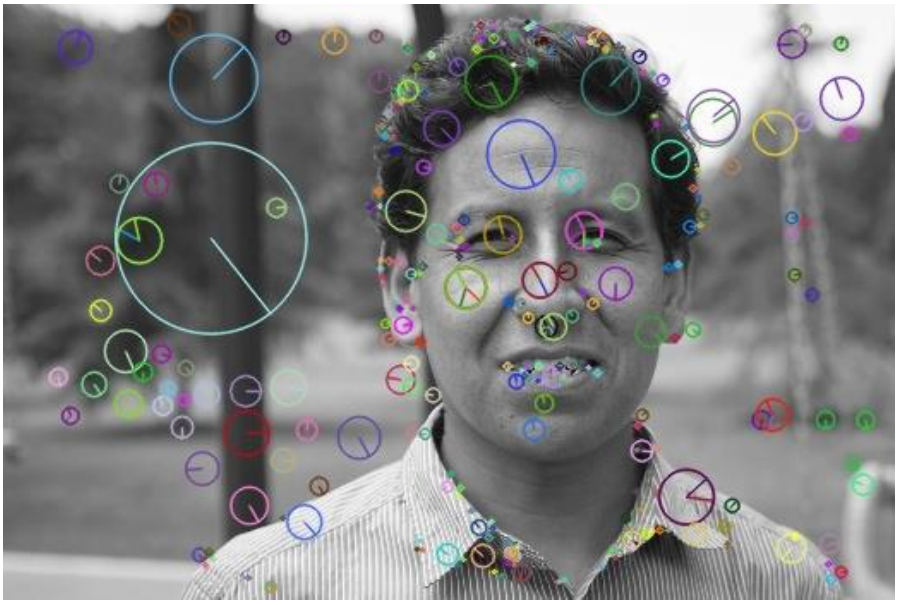
```
39  # ########################## SIFT
40  sift=cv2.xfeatures2d.SIFT_create(0,3,0.05,5,1.6)
41  bfmatcher=cv2.BFMatcher_create(cv2.NORM_L2,crossCheck=True)
42
43  img=cv2.imread('test.JPG')
44  img=cv2.resize(img, (0, 0), fx=0.25, fy=0.25)
45  gray=cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
46  kp=sift.detect(gray,None)
47  kp,dsc=sift.compute(gray,kp)
48
49  img2=cv2.drawKeypoints(gray,kp,img,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
50  cv2.imshow('kp',img2)
51  cv2.waitKey(0)
52  cv2.imwrite('kp.jpg',img2)
53  cv2.destroyAllWindows()
```

A set of good parameters is able to find out key points such as edges and corners, and will show good performance when scaling, rotation, translation applied to the image.

The following images are generated by different parameters. Personally I believe the first set shows the best performance because it returns obvious corners and edges while ignoring other trivial discontinuity in the picture.

Kewen Peng

(0, 3, 0.05, 5, 1.6)



(0, 3, 0.05, 10, 1.6)

(0, 3, 0.1, 5, 1.6)

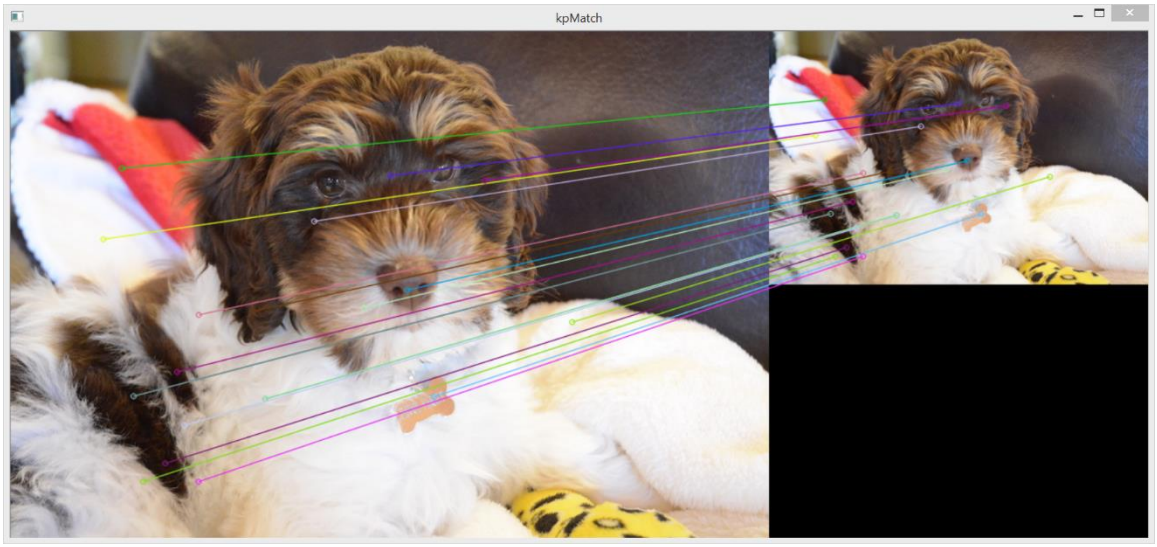
Kewen Peng

**2.2 Scale**

I applied SIFT of the same parameter values on the gray-scaled puppy image. The sift.detect() method returns key points extracted from the image. And sift.compute() returns both the key points and descriptors.

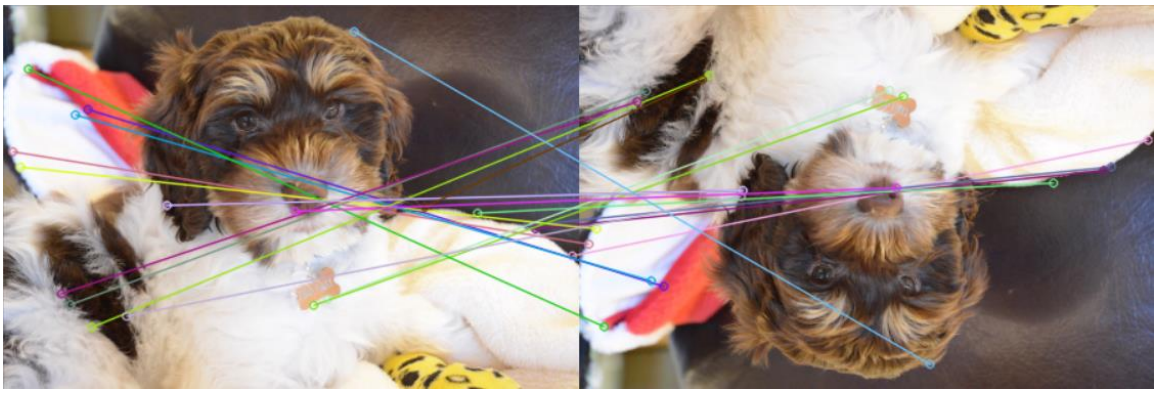
After that, I used the brute-force matcher method to match the key points between the original grey image and the downscaled one. After sorting the matcher in the order of the distance between descriptors (lower distance means better match), I superimposed the first 20 key points from the matcher list using cv2.drawMatches().

```
39  # ########################## SIFT scaling
40  sift=cv2.xfeatures2d.SIFT_create(0,3,0.05,10,1.6)
41  bfmatcher=cv2.BFMatcher_create(cv2.NORM_L2,crossCheck=True)
42
43  img=cv2.imread('DSC_9259.JPG')
44  #img=cv2.resize(img, (0, 0), fx=0.5, fy=0.5)
45  gray=cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
46
47  img2=cv2.resize(img, (0, 0), fx=0.5, fy=0.5)
48  gray2=cv2.cvtColor(img2, cv2.COLOR_RGB2GRAY)
49
50  kp=sift.detect(gray,None)
51  kp,dsc=sift.compute(gray,kp)
52  kp2 = sift.detect(gray2, None)
53  kp2,dsc2=sift.compute(gray2,kp2)
54
55  matcher = bfmatcher.match(dsc,dsc2)
56  matcher = sorted(matcher, key= lambda x: x.distance)
57  img3=cv2.drawMatches(img,kp,img2,kp2,matcher[:20],None,flags=2)
58  cv2.imshow('kpMatch',img3)
59  cv2.waitKey(0)
60  cv2.destroyAllWindows()
```



## 2.3 Rotation

```
55  # ########################## SIFT rotation
56  sift=cv2.xfeatures2d.SIFT_create(0,3,0.05,10,1.6)
57  bfmatcher=cv2.BFMatcher_create(cv2.NORM_L2,crossCheck=True)
58
59  img=cv2.imread('DSC_9259.JPG')
60  img=cv2.resize(img, (0, 0), fx=0.5, fy=0.5)
61  gray=cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
62
63  img2=cv2.flip(img,0)
64  gray2=cv2.cvtColor(img2, cv2.COLOR_RGB2GRAY)
65
66  kp=sift.detect(gray,None)
67  kp,dsc=sift.compute(gray,kp)
68  kp2 = sift.detect(gray2, None)
69  kp2,dsc2=sift.compute(gray2,kp2)
70
71  matcher = bfmatcher.match(dsc,dsc2)
72  matcher = sorted(matcher, key= lambda x: x.distance)
73  img3=cv2.drawMatches(img,kp,img2,kp2,matcher[:20],None,flags=2)
74  cv2.imshow('kpMatch',img3)
75  cv2.waitKey(0)
76  cv2.destroyAllWindows()
```



In general, the SIFT works fine when rotating or scaling the image as the key points from 2 images can be matched by the brute-force matcher.

Kewen Peng

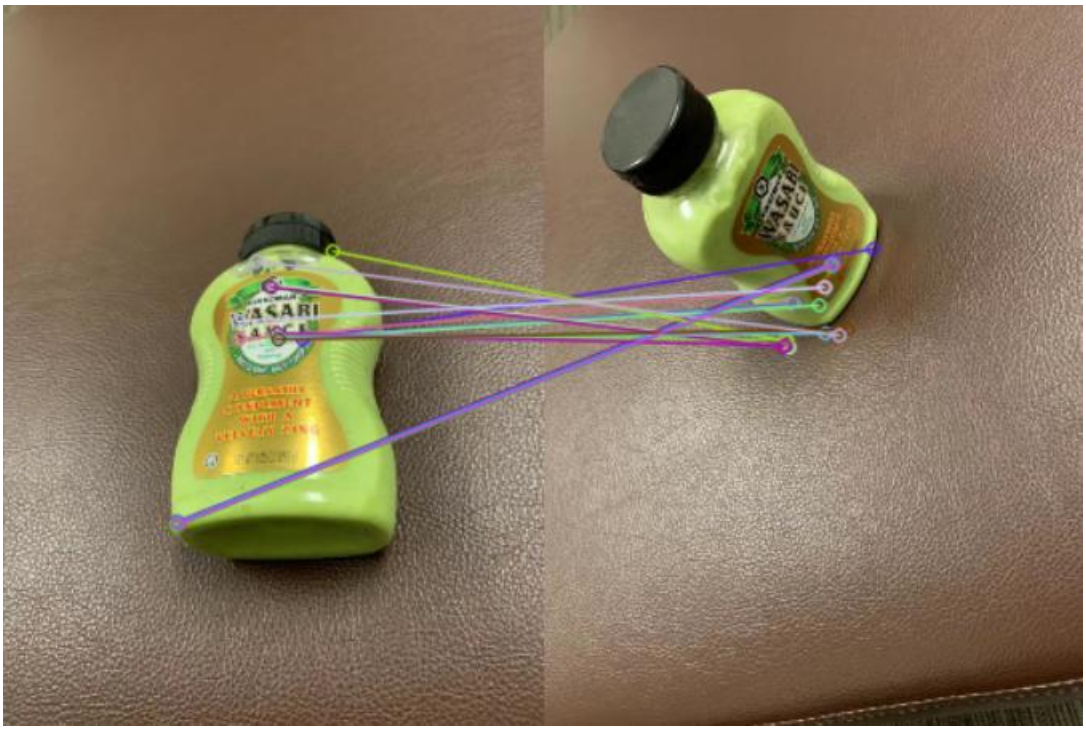
**Part 3 Key points and matching**

In this part I tried to match key points from images of the same scene taken from different angle and distance. I used the dectect()method from the SIFT instance I created via cv2. And then I used the corners detected by cornerHarris() as key points to compute the descriptors of each image. Finally, using the bfmatcher, I drew the top 20 matching key points between the 2 images.

```
# ######################### matching by corners
sift=cv2.xfeatures2d.SIFT_create(0,3,0.05,10,1.6)
bfmatcher=cv2.BFMatcher_create(cv2.NORM_L2,crossCheck=True)
img=cv2.imread('compare.JPG')
img=cv2.resize(img, (0, 0), fx=0.25, fy=0.25)
gray=cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)

img2=cv2.imread('compare1.JPG')
img2=cv2.resize(img2, (0, 0), fx=0.25, fy=0.25)
gray2=cv2.cvtColor(img2, cv2.COLOR_RGB2GRAY)
corners=cv2.cornerHarris(gray,7,7,0.1)
corners2=cv2.cornerHarris(gray2, 7, 7, 0.1)
kpsCorners=np.argwhere(corners > 0.1*corners.max())
kpsCorners = [cv2.KeyPoint(pt[1],pt[0],2) for pt in kpsCorners]
kpsCorners, dstCorners = sift.compute(gray,kpsCorners)
kpsCorners2=np.argwhere(corners > 0.1*corners2.max())
kpsCorners2 = [cv2.KeyPoint(pt[1],pt[0],2) for pt in kpsCorners2]
kpsCorners2, dstCorners2 = sift.compute(gray2,kpsCorners2)
matcher = bfmatcher.match(dstCorners,dstCorners2)
matcher = sorted(matcher, key= lambda x: x.distance)
img4=cv2.drawMatches(img,kpsCorners,img2,kpsCorners2,matcher[:20],None,flags=2)
cv2.imshow('cornerMatch',img4)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Kewen Peng



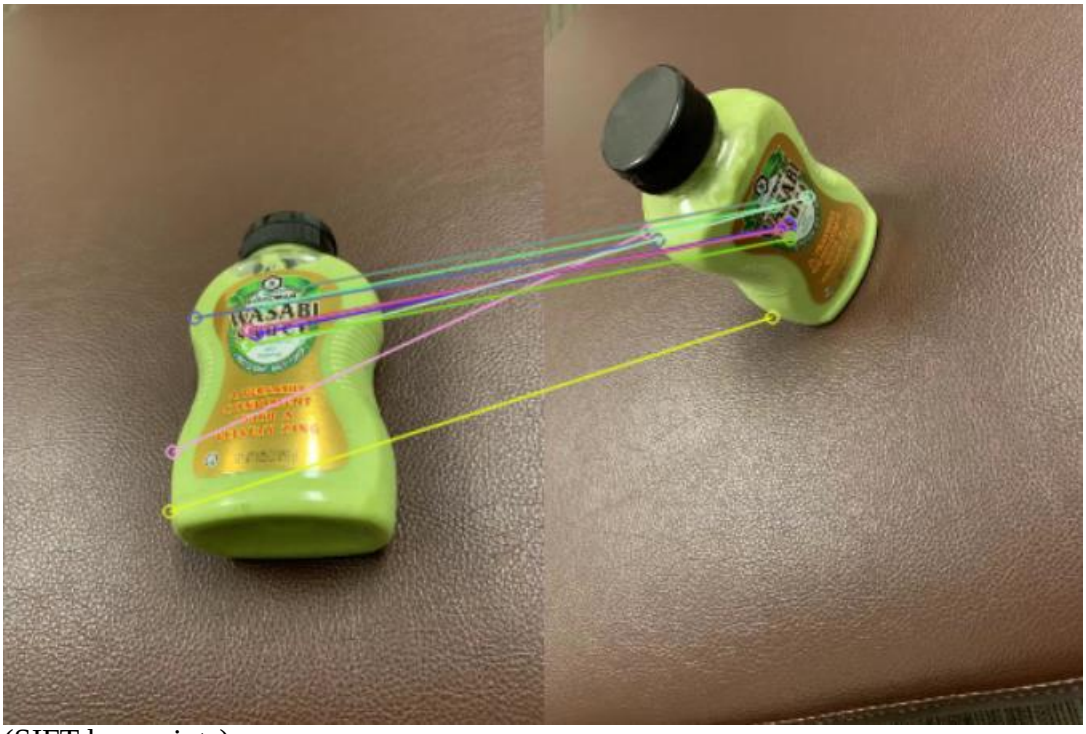
(Corner key points)

```
# ########################## matching by SIFT
sift=cv2.xfeatures2d.SIFT_create(0,3,0.05,10,1.6)
bfmatcher=cv2.BFMatcher_create(cv2.NORM_L2,crossCheck=True)
img=cv2.imread('compare.JPG')
img=cv2.resize(img, (0, 0), fx=0.25, fy=0.25)
gray=cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)

img2=cv2.imread('compare1.JPG')
img2=cv2.resize(img2, (0, 0), fx=0.25, fy=0.25)
gray2=cv2.cvtColor(img2, cv2.COLOR_RGB2GRAY)
kp=sift.detect(gray,None)
kp,dsc=sift.compute(gray,kp)
kp2 = sift.detect(gray2, None)
kp2,dsc2=sift.compute(gray2,kp2)

matcher = bfmatcher.match(dsc,dsc2)
matcher = sorted(matcher, key= lambda x: x.distance)
img3=cv2.drawMatches(img,kp,img2,kp2,matcher[:10],None,flags=2)
cv2.imshow('SIFTMatch',img3)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Kewen Peng



(SIFT key points)

The first part is generated using corners as key points, and the second one is generated using SIFT key points. By observation, SIFT seems to have a better performance on matching key points. This is because in most cases corners are not efficient key points. By just using corners as key points, the model takes a larger risk of mismatching. I would assume that corner key points can have better performance when the image is mainly made of distinguishable corners.