ECE661 Computer Vision HW4

Chengjun Guo
guo456@purdue.edu

28 September 2022

1 Theory question

Question:What is the theoretical reason for why the LoG of an image can be computed as a DoG. Also explain in your own words why computing the LoG of an image as a DoG is computationally much more efficient for the same value of σ.

The result from the scale-space theory is:

$$\frac{\partial}{\partial\sigma}ff(x,y,\sigma) = \sigma\nabla^2 ff(x,y,\sigma) \tag{1}$$

which gives:

$$LoG(f(x,y)) = \frac{\partial}{\partial\sigma}ff(x,y,\sigma) \tag{2}$$

This implies that the LoG of image f(x,y) can be approximated by

$$\frac{ff(x,y,\sigma+\Delta\sigma) - ff(x,y,\sigma)}{\Delta\sigma} \tag{3}$$

which is the difference between images smoothed at two different scale.
The reason that DoG is much more efficient is that Gaussian is separable in x and y. 2-D smoothing can be replaced with 1-D smoothing in x and y separately. However, the LoG operator is not separable. For example, when picking $\sigma = \sqrt{2}$, LoG needs a 13 by 13 kernal for the convolution where DoG only needs two 9 element kernals in x and y.

2 Logic

The code used are mostly the same for both tasks. The only difference are the variable names and values for points and input images.

For the Deep Learning based method, I created a new virtual environment

superglue. After activating the virtual environment, it is executed manually with bash commands instead of the .sh file.

For my own point detector and matching, I defined Haar_wavelet_filter to calculate the image intensity's gradient along x and y. The harris_corner_detector is defined to search for the interest points in one image. The correspondences is defined to pairing the interest points from two images in SSD mode or NCC mode.

For the cv2 method point searching and matching, I defined sift for point searching and matching.

2.1 Algorithms and implementation

2.1.1 Harris Corner detector

The first part of the Harris Corner detector is Haar wavelet filter. It is used to calculate the x-gradient I_x and y-gradient I_y. The size of the kernal is determined by σ. After I_x and I_y are found, we use a window of size $5\sigma \times 5\sigma$ to find the relationship between pixels and the surrounding pixels. The matrix is constructed by:

$$C = \begin{bmatrix} \sum d_x^2 & \sum d_x d_y \\ \sum d_x d_y & \sum d_y^2 \end{bmatrix} \tag{4}$$

The Harris response

$$R = det(C) - k(Tr(C))^2 \tag{5}$$

is used to determine the 'curvature'. In case that the large amount of interest points, I didn't threshold the points by top 10% or 30%. Top 500 points would be selected for non maximum suppression. The suppression is to select the pixel with highest R to avoid excessive point for one corner.

2.1.2 Points matching with SSD and NCC

With the interest points as input, the matching function then use two possible metrics to find the best matching pairs. Windows of size 21 by 21 surrounding each pair of points is considered for the evaluation. Every point in the first image is looped with the points in the second image to find the lowest distance.

For Sum of Squared Difference(SSD) method:

$$SSD = \sum_i \sum_j \left(f_1(i,j) - f_2(i,j)\right)^2$$

The Euclidean distance between the descriptor vectors are calculated where the best SSD value is 1.

For Normalized Cross Correlation(NCC) method:

$$NCC = \frac{\sum\sum\left(f_1(i,j) - mean1\right)\left(f_2(i,j) - mean2\right)}{\sqrt{\left(\sum\sum\left(f_1(i,j) - mean1\right)^2\right)\left(\sum\sum\left(f_2(i,j) - mean2\right)^2\right)}}$$

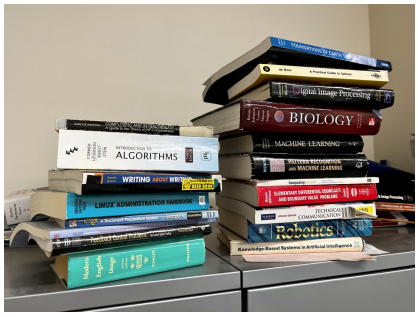
where mean denotes the mean value of the windows and f_1, f_2 denote the value of the point for image 1 and 2.
Since the best NCC value is 1, the distance is calculated by absolute value of 1-NCC_distance.

2.1.3 Sift overview

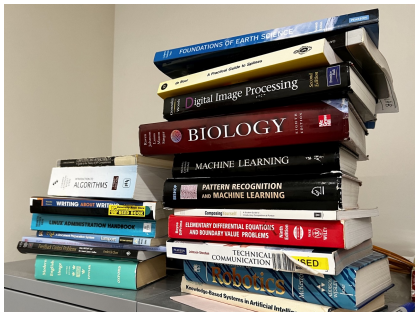
The first step is constructing DoG pyramid. The image is convolved with an initial scale. Then downsample by a factor of two, double the scale. Within each octave, take the difference of successive layers.
The second step is to find the local extrema in each octave. Each pixel are compared with 8 surrounding pixels and 18 pixels in 3 by 3 neighbourhood above and below.
In the third step, the extremum in a higher octave may be coarse as the σ increase. second order Taylor series expansion is used around it. Jacobian and Hessian are used for the first and second partial derivatives.
Then the extrema is thresholded by 0.03. A dominant local orientation and an 128 dimensional vector are calculated for each extremum. These feature descriptor are used for point matching.

3 Data and images:

Input images:



(a) book1



(b) book2

(c) fountain1

(d) fountain2

(a) mouse1

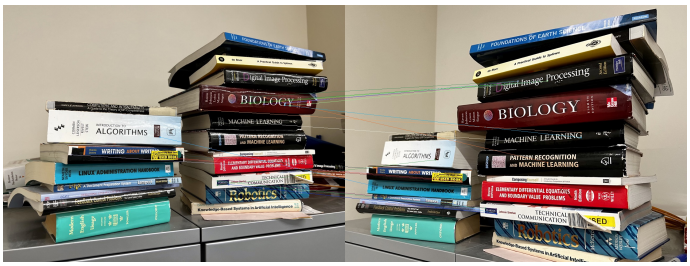
(b) mouse2

(c) filter1

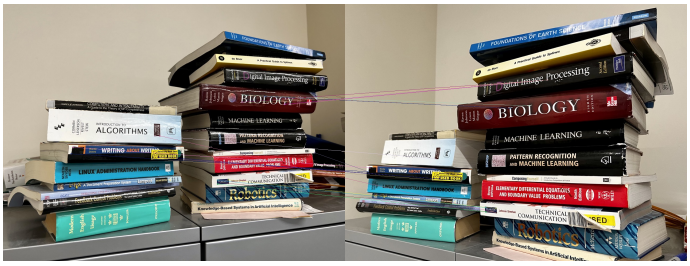
(d) filter2

3.1 book outputs

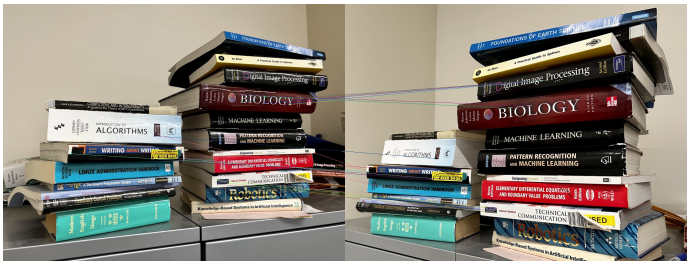
For SSD output $\sigma = 1, 1.2, 1.4, 4$:



(a) book SSD 1



(b) book SSD 1.2

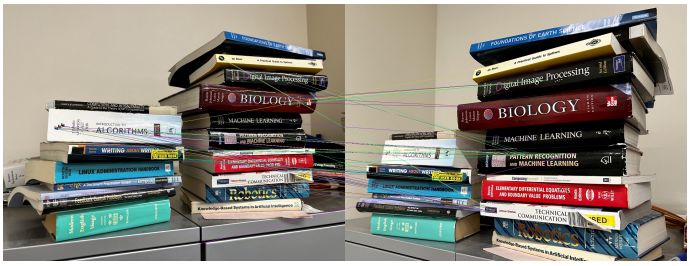


(c) book SSD 1.4

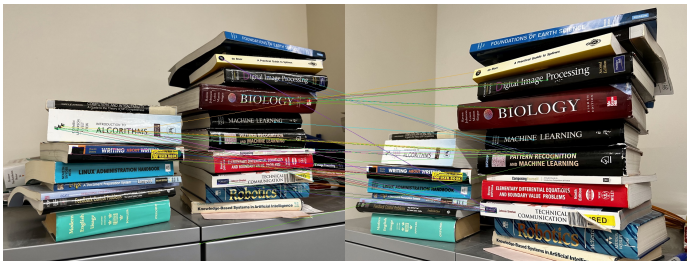
(d) book SSD 4

For NCC output $\sigma = 1, 1.2, 1.4, 4$:

(a) book NCC 1



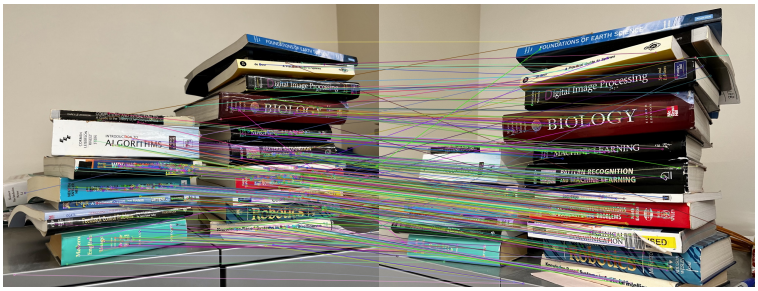
(b) book NCC 1.2



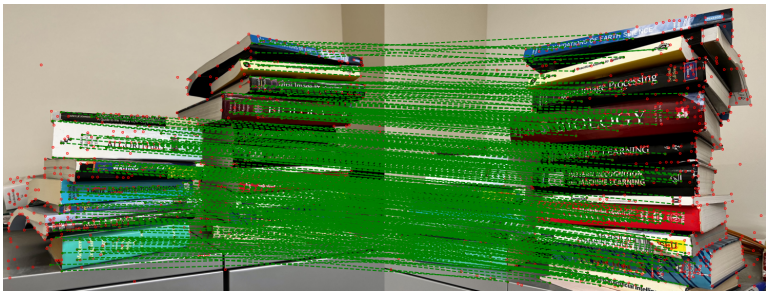
(c) book NCC 1.4

(d) book NCC 4

cv2 Sift output and superglue output:



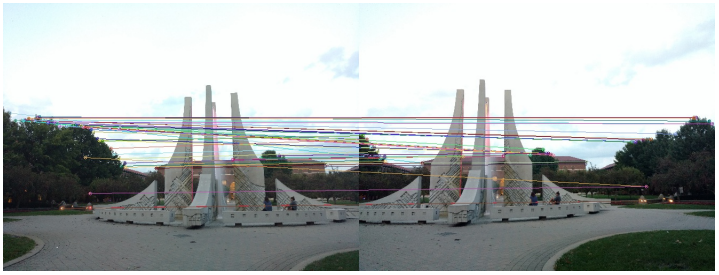
(a) book sift



(b) book superglue

3.2 fountain outputs

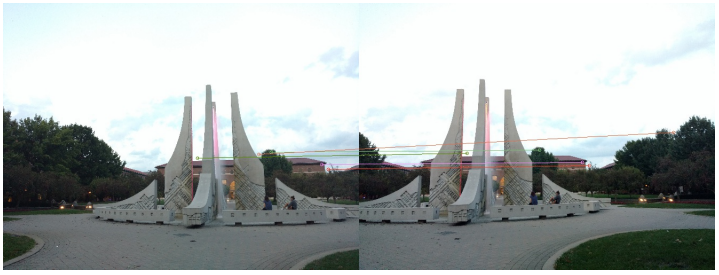
For SSD output $\sigma = 1, 1.2, 1.4, 4$:



(a) fountain SSD 1

(b) fountain SSD 1.2

(c) fountain SSD 1.4



(d) fountain SSD 4

For NCC output $\sigma = 1, 1.2, 1.4, 4$:

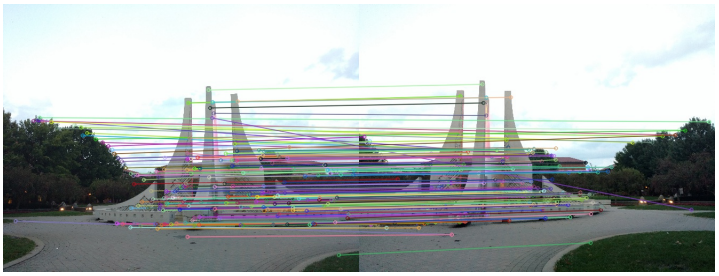
(a) fountain NCC 1

(b) fountain NCC 1.2

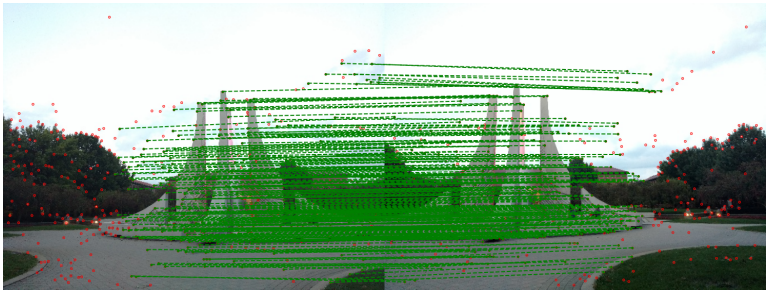
(c) fountain NCC 1.4

(d) fountain NCC 4

cv2 Sift output and superglue output:



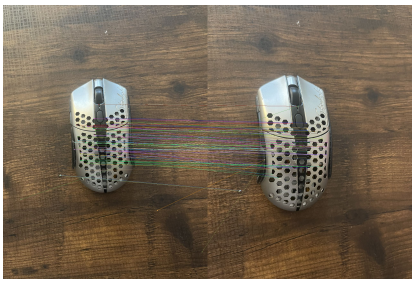
(a) fountain sift



(b) fountain superglue

3.3 mouse outputs

For SSD output $\sigma = 1, 1.2, 1.4, 4$:



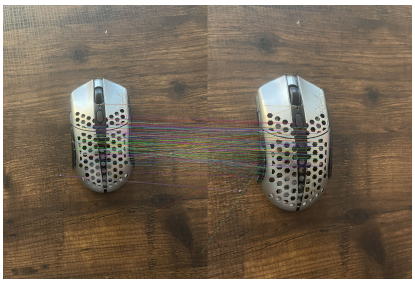
(a) mouse SSD 1

(b) mouse SSD 1.2

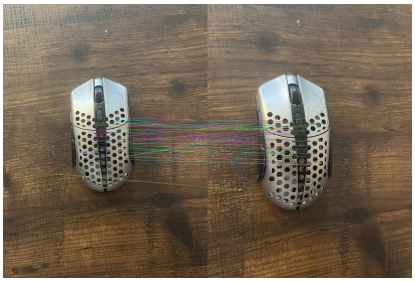
(c) mouse SSD 1.4

(d) mouse SSD 4

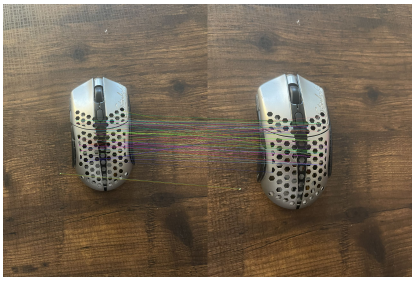
For NCC output $\sigma = 1, 1.2, 1.4, 4$:



(a) mouse NCC 1



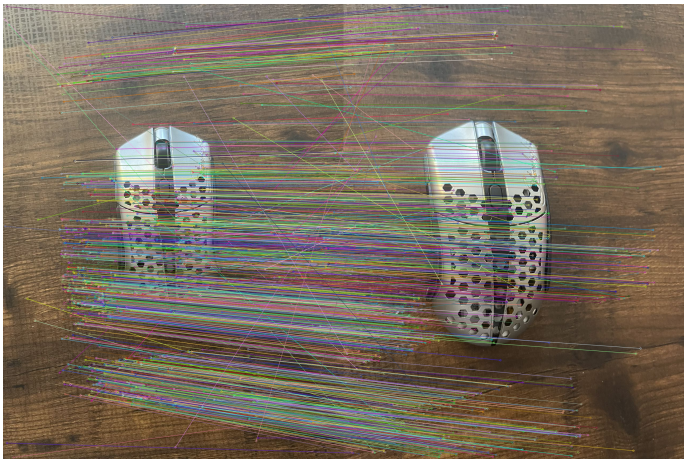
(b) mouse NCC 1.2



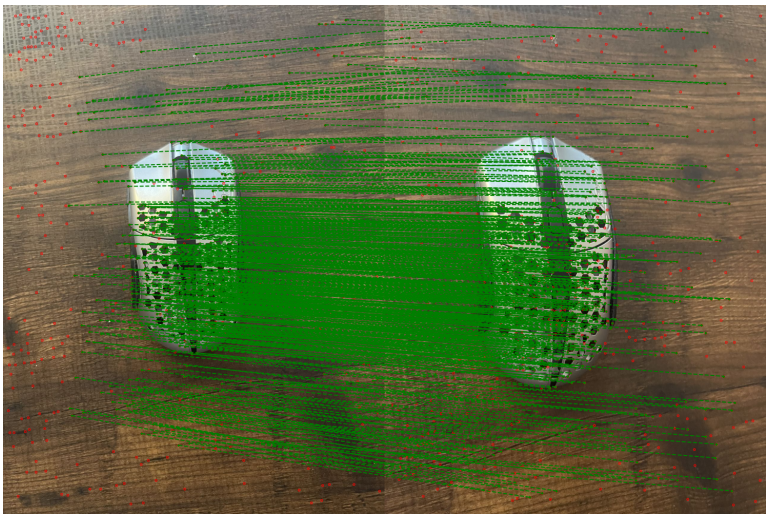
(c) mouse NCC 1.4

(d) mouse NCC 4

cv2 Sift output and superglue output:



(a) mouse sift



(b) mouse superglue

3.4 filter outputs

For SSD output $\sigma = 1, 1.2, 1.4, 4$:

(a) filter SSD 1

(b) filter SSD 1.2

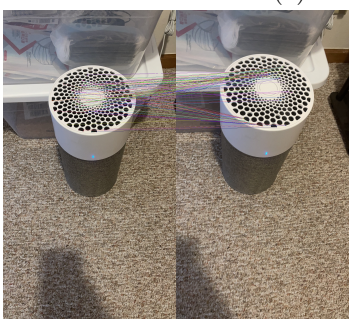
(c) filter SSD 1.4

(d) filter SSD 4

For NCC output $\sigma = 1, 1.2, 1.4, 4$:

(a) filter NCC 1

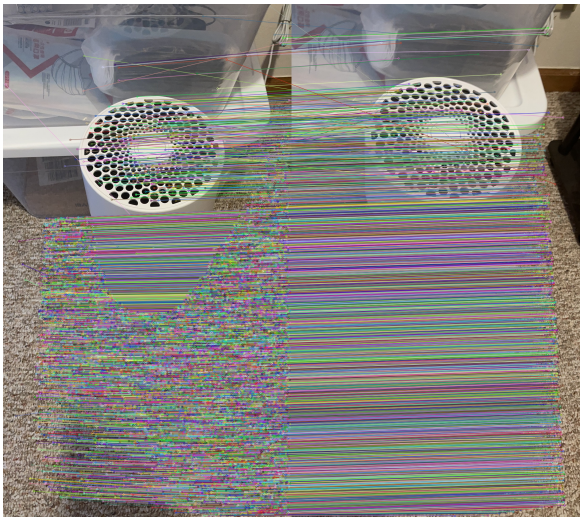
(b) filter NCC 1.2



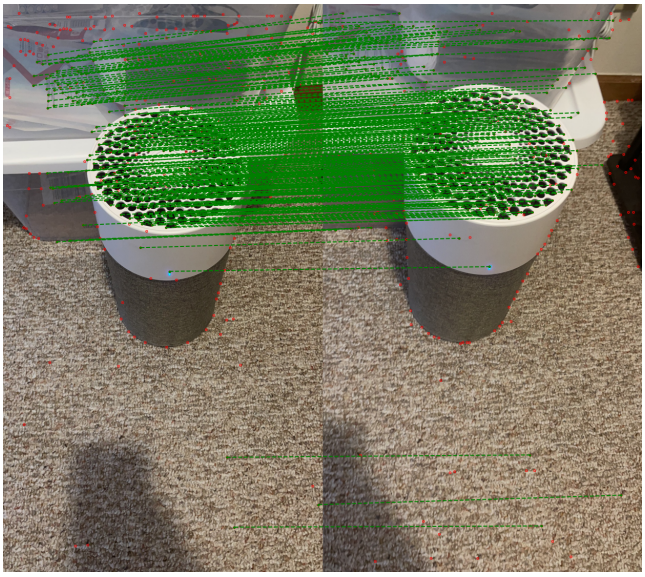
(c) filter NCC 1.4

(d) filter NCC 4

cv2 Sift output and superglue output:



(a) filter sift



(b) filter superglue

The thresholds:

Harris Corner Detector	Scale	σ
	Corner response threshold	top 500
SSD	Matching window size	21
	SSD score threshold	25
NCC	Matching window size	21
	NCC score threshold	25

4 Observations

4.1 Observation on interest points at various scales using Harris

The performance of the interest point detection has best performance as the sigma is at 1. This is because the sigma controls the smoothing. As the sigma increase, the points would be less and it would be a little bit off the corner.

4.2 Observation on the output quality Harris vs Sift

For given input such as book, sift have better performance. To reduce the execution time, I only select top 500 points and pass these points to the non maximum suppression for Harris. Several lines are paired with high vertical error while sift lines are mostly parallel and the vertical errors are low.
For my own input such as mouse, Harris have better performance. The selected sigma helped to ignore the background noises and focus on the point matching of my mouse. One possible reason is that sift have higher sigma.

4.3 Observation on NCC and SSD

For my own input filter, SSD have better performance. There are several faulty lines are eliminated by SSD such as the yellow line from right to top left in NCC. Other than those several lines, most lines are similar to each other.

4.4 Observation on output quality of superpoint+superglue

For given input book, Deep Learning based method have better performance. There are several lines paired with bottom point and top point are eliminated. Some points from the edge of the desk paired with the other side of the points of desk are corrected too.
For my own input such as filter, DL based method also have better performance. Sift would spend lots of points on carpet due to the noise. However, DL method would ignore the noises and find interest point in the box. The neural network is more human-like and eliminate errors with the same pattern.

5 Code

```
import numpy as np
import cv2
import sys
import math
import matplotlib.pyplot as plt

def Haar_wavelet_filter(sigma):
    M = math.ceil(4*sigma)
    M += M % 2
    kernel = np.ones((M,M))
    kernel[:,:M//2] = -1
    return kernel, -kernel.T

def harris_corner_detector(image,sigma):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    image = image/255
    kernel_x, kernel_y = Haar_wavelet_filter(sigma)
    dx = cv2.filter2D(image, -1, kernel=kernel_x)
    dy = cv2.filter2D(image, -1, kernel=kernel_y)
    dx_sq = dx**2   # space of dx square
    dy_sq = dy**2   # space of dy square
    dx_dy = dx*dy    # space of dxdy
    N = 2 * int((5 * sigma)//2) + 1
    window = np.ones((N,N))
    sum_dx_sq = cv2.filter2D(dx_sq,-1,kernel=window)
    sum_dy_sq = cv2.filter2D(dy_sq, -1, kernel=window)
    sum_dx_dy = cv2.filter2D(dx_dy, -1, kernel=window)
    detC = sum_dx_sq * sum_dy_sq - sum_dx_dy ** 2
    traceC = sum_dx_sq + sum_dy_sq
    # R =detC / (traceC ** 2 + 0.0001)
    R = detC - 0.05 * traceC ** 2
    R[R<0] = 0
    # threshold = np.percentile(np.sort(R), 90)
    # print(threshold)
    threshold = np.sort(R.flatten())[-500]
    # non maximum suppression
    new = np.zeros(R.shape)
    # N=21
    for i in range(N//2, image.shape[1] - N//2):
        for j in range(N//2, image.shape[0] - N//2):
            region = R[j-N//2 :j+N//2+1, i-N//2:i+N//2+1]
```

```
            R_max = np.amax(region)
            if R_max==R[j,i] and R[j,i]>threshold:
                new[j,i] = R_max
    coordinates = np.where(new>0)
    return coordinates


def correspondences(img1,img2,coordinates1,coordinates2,metric='SSD',N=10):
    point_pair = []
    img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY) / 255
    img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY) / 255
    for pt1 in list(zip(coordinates1[0], coordinates1[1])):
        distance_list = []
        for pt2 in list(zip(coordinates2[0], coordinates2[1])):
            distance_list.append(compute_distance(img1,img2,pt1,pt2,metric,N))
        if np.min(distance_list) < 25:
            best_point = list(zip(coordinates2[0], coordinates2[1]))[np.argmin(distance_list
            point_pair.append((pt1,best_point))
    return point_pair


def compute_distance(img1,img2,point1,point2,metric,N=10):
    img1 = cv2.copyMakeBorder(img1,N,N,N,N,cv2.BORDER_REPLICATE)
    img2 = cv2.copyMakeBorder(img2,N,N,N,N,cv2.BORDER_REPLICATE)
    window1 = img1[point1[0]:point1[0]+int(2*N+1),point1[1]:point1[1]+int(2*N+1)]
    window2 = img2[point2[0]:point2[0]+int(2*N+1),point2[1]:point2[1]+int(2*N+1)]
    distance = np.inf
    if metric == 'SSD':
        distance = np.sum((window1 - window2) ** 2)
    elif metric == 'NCC':
        mean1 = window1.mean()
        mean2 = window2.mean()
        corr = np.sum((window1-mean1)*(window2-mean2))
        norm1 = np.sum((window1-mean1) ** 2)
        norm2 = np.sum((window2-mean2) ** 2)
        distance = 1 - corr/np.sqrt(norm1*norm2)
    return distance

def sift(img1,img2):
    # img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
    # img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
    sift = cv2.xfeatures2d.SIFT_create()
    kp1, des1 = sift.detectAndCompute(img1,None)
    kp2, des2 = sift.detectAndCompute(img2,None)
    bf = cv2.BFMatcher()
    # image1_kp = cv2.drawKeypoints(img1, kp1, outImage=np.array([]), flags=cv2.DRAW_MATCHES
```

```
    matches = bf.knnMatch(des1,des2,k=2)
    good = []
    for m, n in matches:
        if m.distance < 0.75 * n.distance:
            good.append([m])
    ans = cv2.drawMatchesKnn(img1,kp1,img2,kp2,good,None,flags=2)
    return ans

if __name__ == '__main__':
    image1_name = 'mouse_1'
    image2_name = 'mouse_2'
    mode = 'SSD'
    img1 = cv2.imread(image1_name+'.jpg')
    img2 = cv2.imread(image2_name+'.jpg')
    for sigma in [1,1.2,1.4,4]:
        # print(harris_corner_detector(img1, 2))
        coordinates1 = harris_corner_detector(img1, sigma)
        coordinates2 = harris_corner_detector(img2, sigma)
        # print('point got',len(coordinates1[0]),len(coordinates2[0]))
        # for x, y in list(zip(coordinates1[0], coordinates1[1])):
        #     img1 = cv2.circle(img1, (y, x), 3, (0, 255, 0))
        # cv2.imshow('points', img1)
        point_pairs = correspondences(img1,img2,coordinates1,coordinates2,mode)
        # print(point_pairs)
        offset = img2.shape[1]
        final = np.concatenate((img1,img2),axis=1)
        for pointpair in point_pairs:
            color = list(np.random.random(size=3) * 256)
            cv2.circle(final, (pointpair[0][1],pointpair[0][0]), 3, color, 1)
            cv2.circle(final, (pointpair[1][1]+offset,pointpair[1][0]), 3, color, 1)
            # d = np.array(pointpair[0])-np.array(pointpair[1])
            # sd = np.dot(d.T,d)
            # print(sd)
            # if sd < 625:
            cv2.line(final,(pointpair[0][1],pointpair[0][0]),(pointpair[1][1]+offset,pointpa
        cv2.imwrite(image1_name+'_'+image2_name+'_'+mode+'_'+str(sigma)+'.jpg', final)

    ans = sift(img1,img2)
    cv2.imwrite(image1_name+'_'+image2_name+'_'+'sift.jpg',ans)

    cv2.waitKey()
    cv2.destroyAllWindows()
```