ECE 66100 HW4 Report

Zhengxin Jiang
(jiang839@purdue.edu)

September 29, 2022

1 Theory Question

Approximate LoG by DoG

Since the Gaussian depends on σ, we let $ff(x, y, \sigma)$ represent the σ-smoothed version of an image. Hence we have the already proven equation

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

Since $\nabla^2 ff(x, y, \sigma)$ is the LoG of $f(x, y)$, we have

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

It is now implied that LoG can be approximated by taking the subtraction of two smoothed versions with different σ.

Efficiency of DoG

DoG is computationally much more efficient for the same value of σ because DoG uses smaller operator than LoG. For example, with $\sigma = \sqrt{2}$, LoG uses a 13×13 operator while DoG uses a 9 element 1D operator.

2 Task 1.1: Harris Corner Detector

The Harris Corner Detection algorithm finds the interest points in an image by trying to find the points with significant gray level variations in at least two different directions.

The first step is to apply Haar filter at scale σ on the image to compute d_x and d_y. The size N of the Haar filter is the smallest even integer larger than 4σ.

Then we construct a matrix for exch pixel based on their $5\sigma \times 5\sigma$ neighborhood

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

We use the Harris Response

$$R = det(C) - k(Tr(C)^2)$$

$$k = \frac{det(C)}{Tr(C)^2}$$

to determine if the pixel is a corner. In my implementation I set the threshold to be the largest 10 percent in R. The point with R value over the threshold also need to be the local maximum to be determined as a corner.

3 Task 1.2: Points Correspondence Using SSD and NCC

Since the interest points generated by Harris Corner Detector do not have descriptor vectors, we find the correspondences by comparing the $(m+1) \times (m+1)$ window of the two points. The distances are calculated by the following formulas:

For SSD

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

For NCC

$$NCC = \frac{\sum_i \sum_j ((f_1(i,j) - m_1) - (f_2(i,j) - m_2))}{\sqrt{(\sum_i \sum_j (f_1(i,j) - m_1))^2 (\sum_i \sum_j (f_2(i,j) - m_2))^2}}$$

4 Task 2: SIFT

For this homework assignment I choose to test SIFT on the image pairs. The SIFT algorithm contains following steps:

1. Constructing the DoG pyramid and find all the local extrema. Each point in the DoG pyramid is compared to its $3 \times 3 \times 3$ neighborhood.

2. Since the image becomes coarse when the level goes higher in the pyramid, we localize the extrema with better accuracy by estimating the second-order derivatives at the sampling points. We have

$$D(x) \approx D(x_0) + J^T(x_0)x + \frac{1}{2}x^T H(x_0)x$$

where J is the gradient vector and H is the Hessian. The location of extrema is given by

$$x = -H^{-1}(x_0)J(x_0)$$

3. We eliminate the weak extrama by setting a threshold of $|D(x)| < 0.03$. Points with $|D(x)| < 0.03$ are rejected.

4. We assign each interest point with a dominant orientation and a 128-dimensional vector. The vector can be used for finding point correspondences.

5 Results of all Tasks

Shown below are all original images used for this assignment.

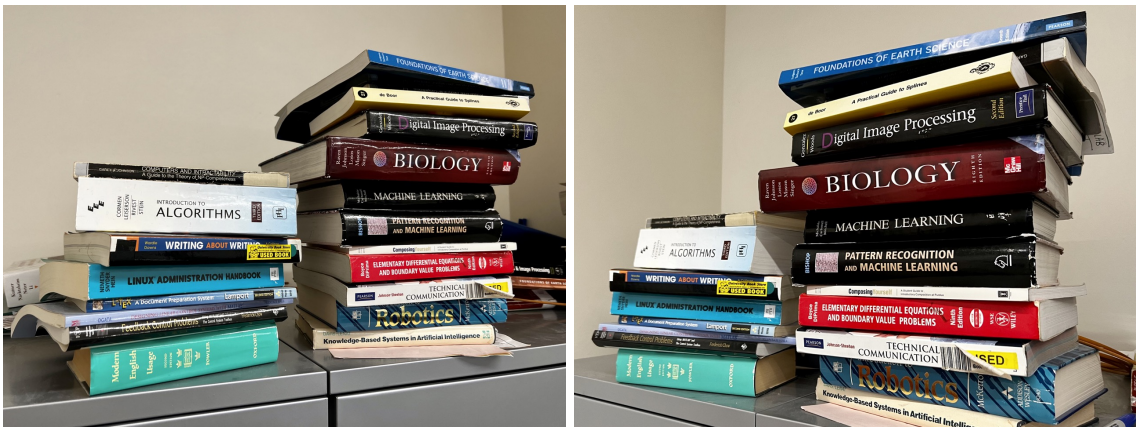

Figure 1: Image pair of books

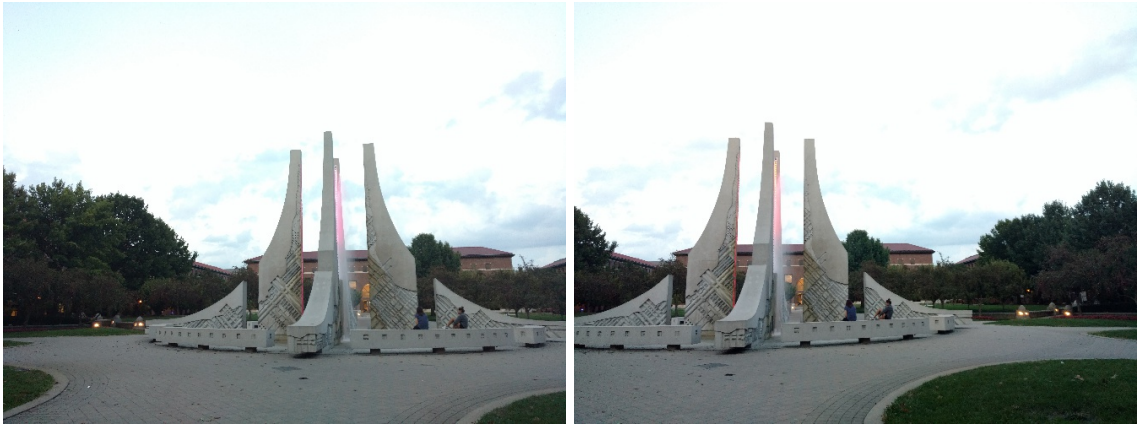

Figure 2: Image pair of a fountain

Figure 3: Image pair of a table

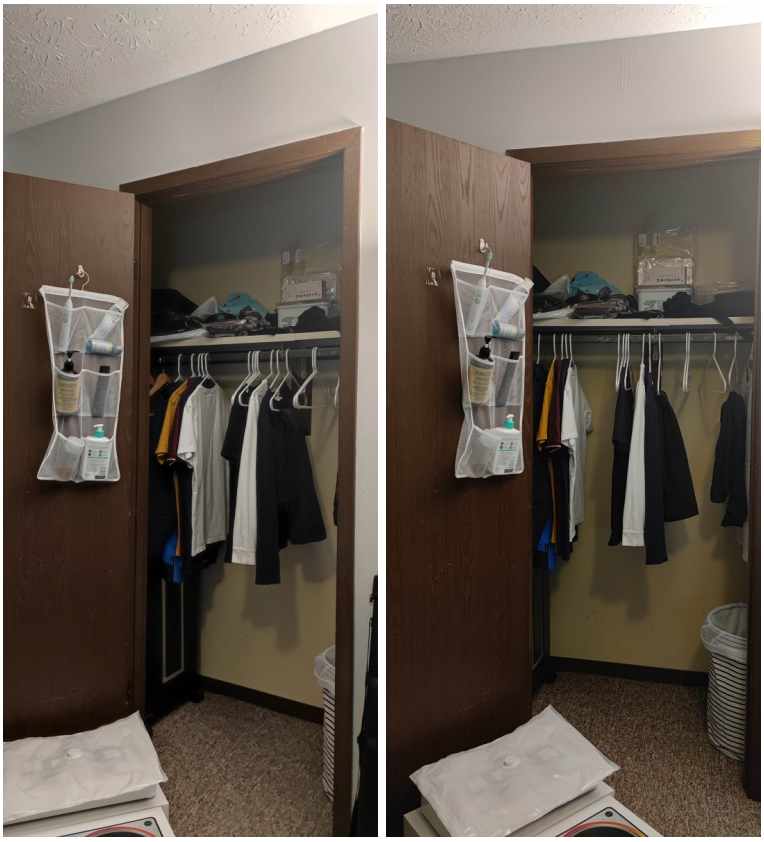

Figure 4: Image pair of clothes

Image pair of books

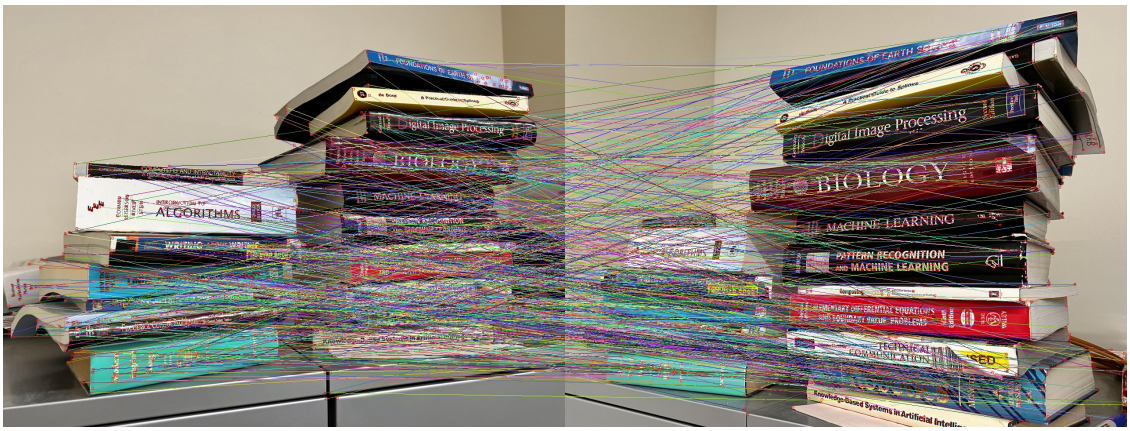

Figure 5: Harris detected corners and correspondences with ssd, $\sigma = 1.8$

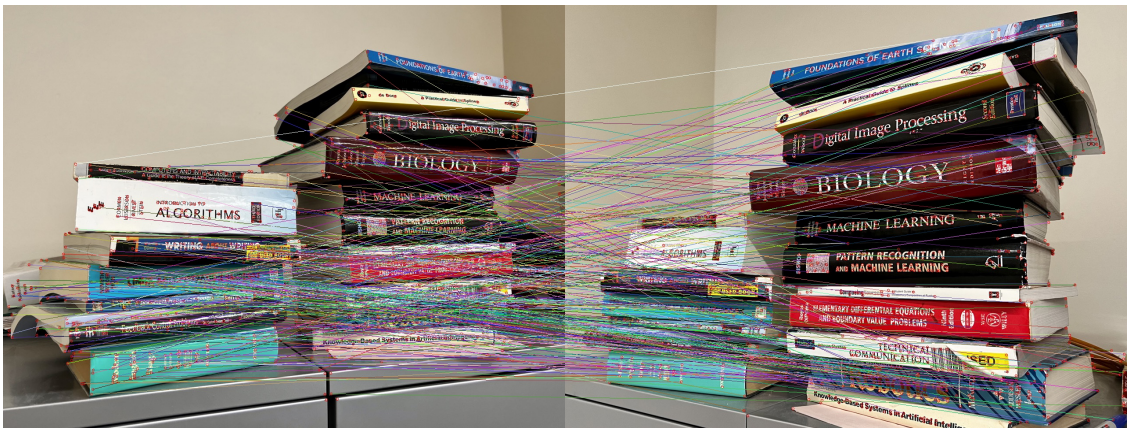

Figure 6: Harris detected corners and correspondences with ncc, $\sigma = 1.8$

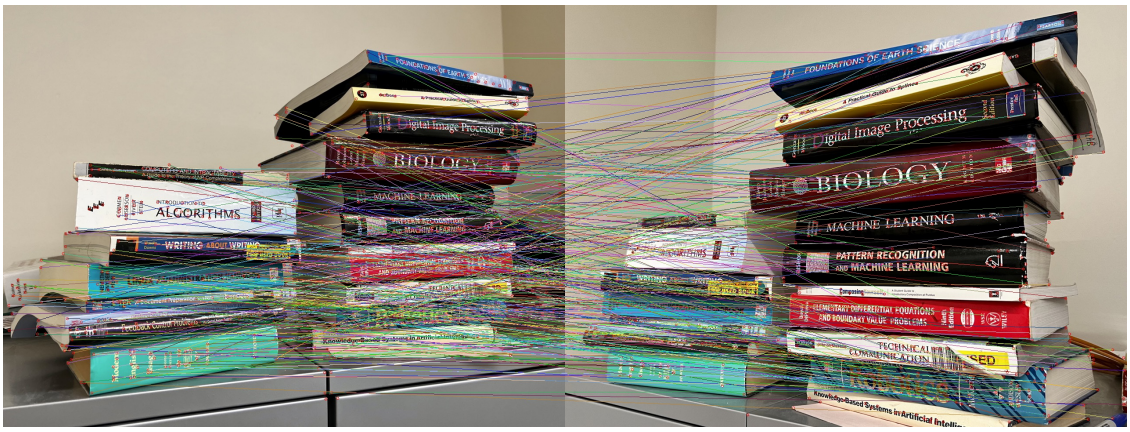

Figure 7: Harris detected corners and correspondences with ssd, $\sigma = 2.2$

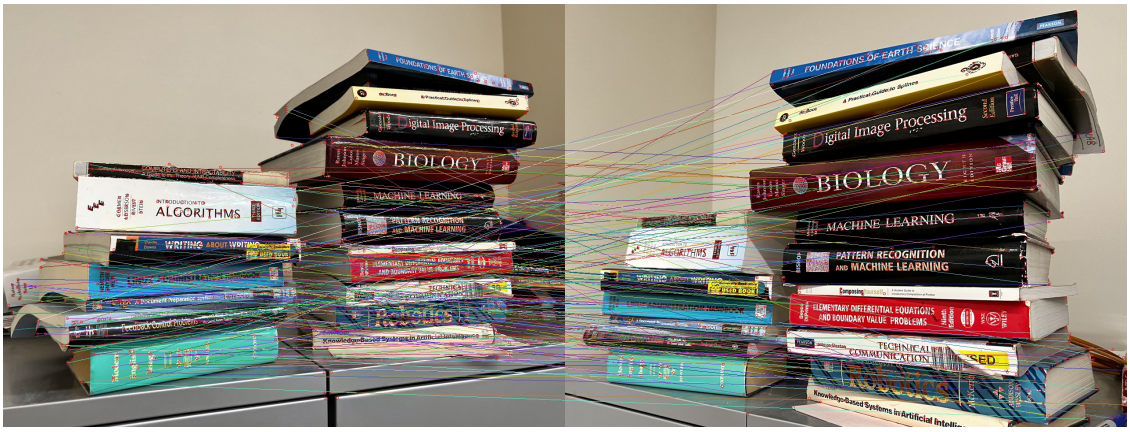

Figure 8: Harris detected corners and correspondences with ncc, $\sigma = 2.2$

Figure 9: Harris detected corners and correspondences with ssd, $\sigma = 2.6$

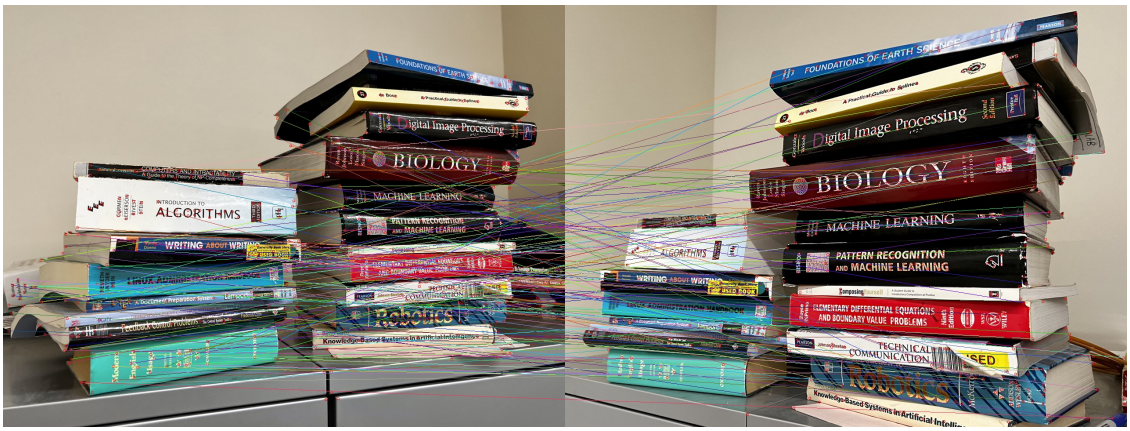

Figure 10: Harris detected corners and correspondences with ncc, $\sigma = 2.6$

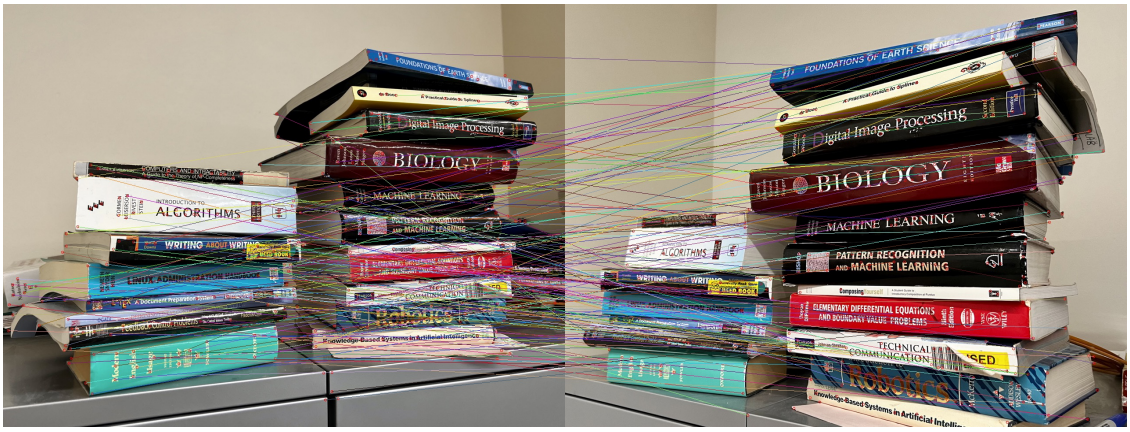

Figure 11: Harris detected corners and correspondences with ssd, $\sigma = 3.0$

Figure 12: Harris detected corners and correspondences with ncc, $\sigma = 3.0$

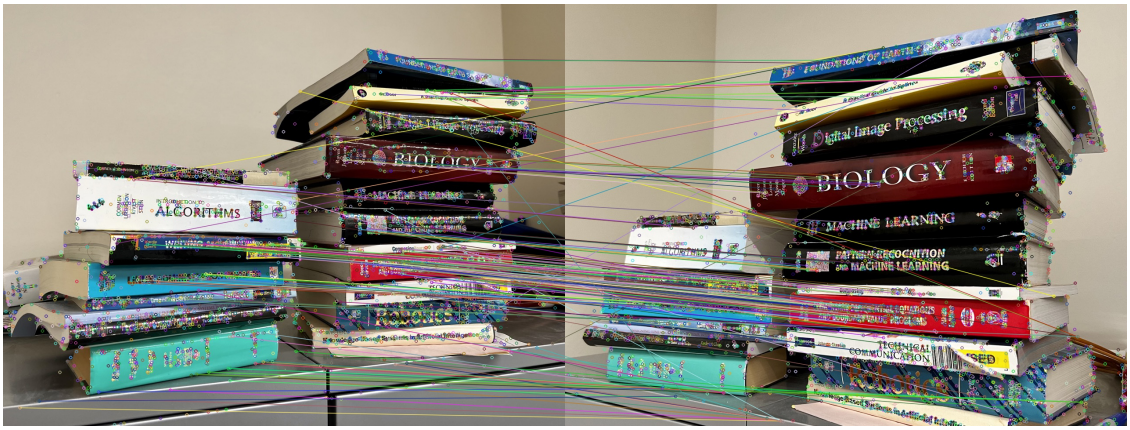

Figure 13: SIFT detected points and correspondences

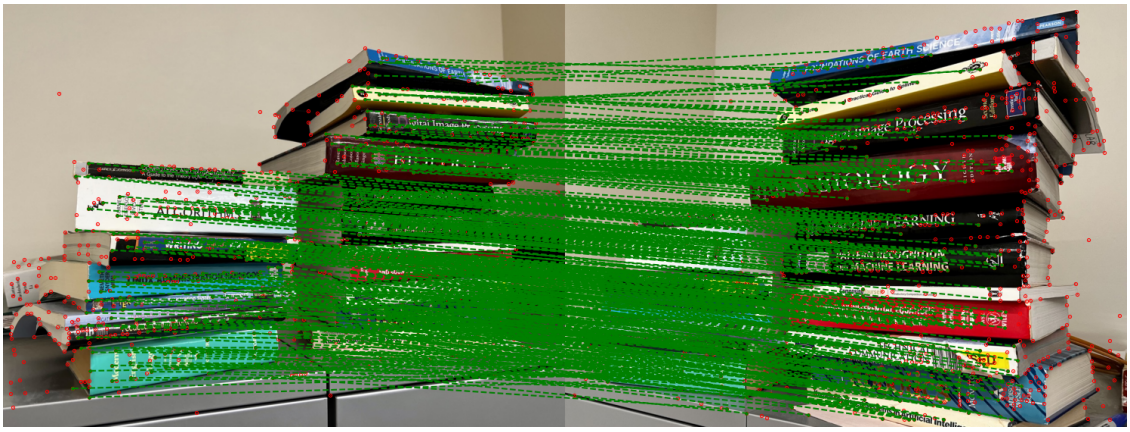

Figure 14: SuperGlue detected points and correspondences

Image pair of the fountain

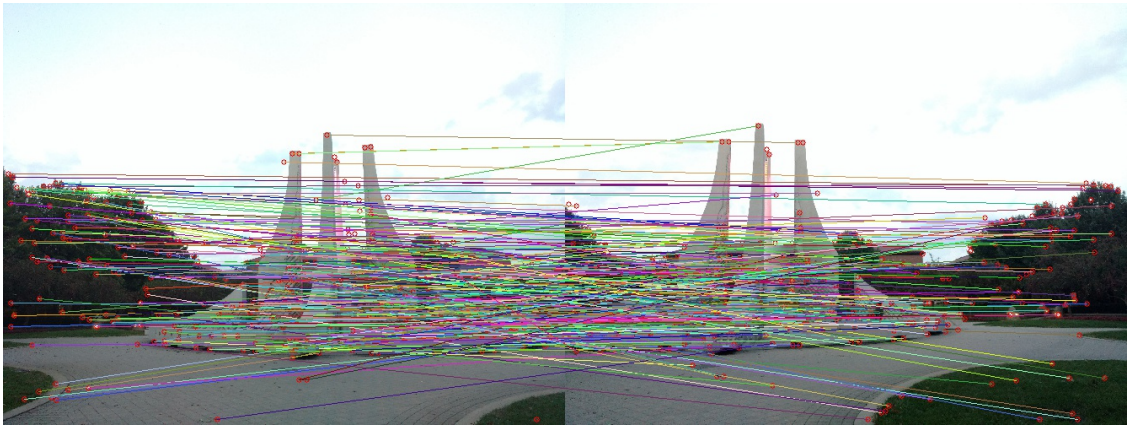

Figure 15: Harris detected corners and correspondences with ssd, $\sigma = 1.8$

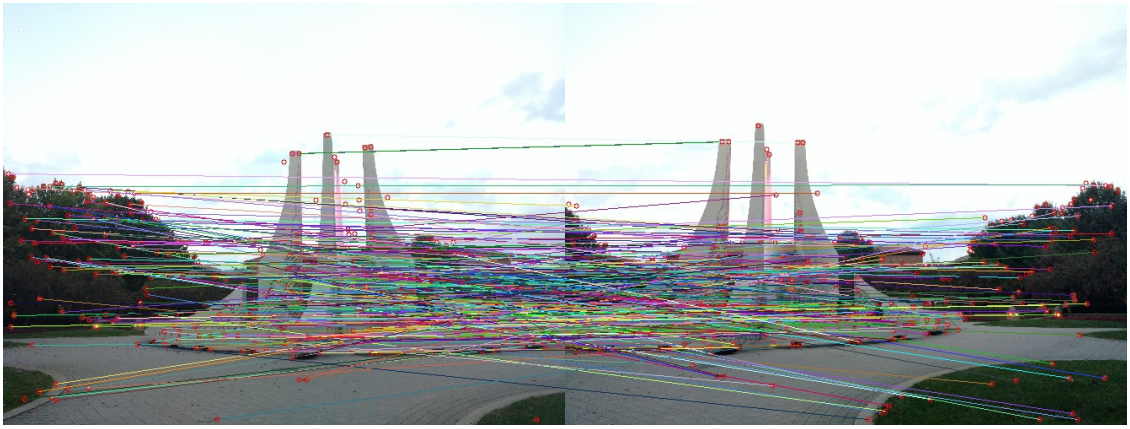

Figure 16: Harris detected corners and correspondences with ncc, $\sigma = 1.8$

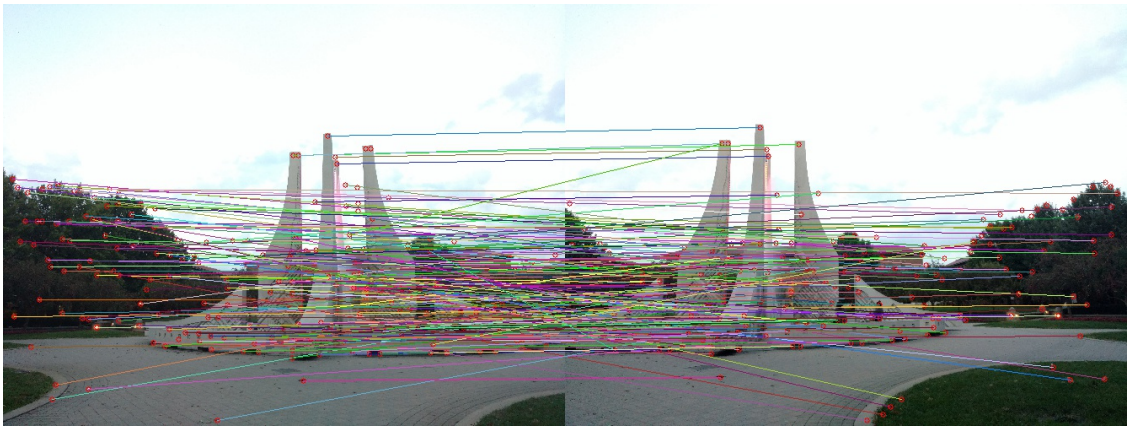

Figure 17: Harris detected corners and correspondences with ssd, $\sigma = 2.2$

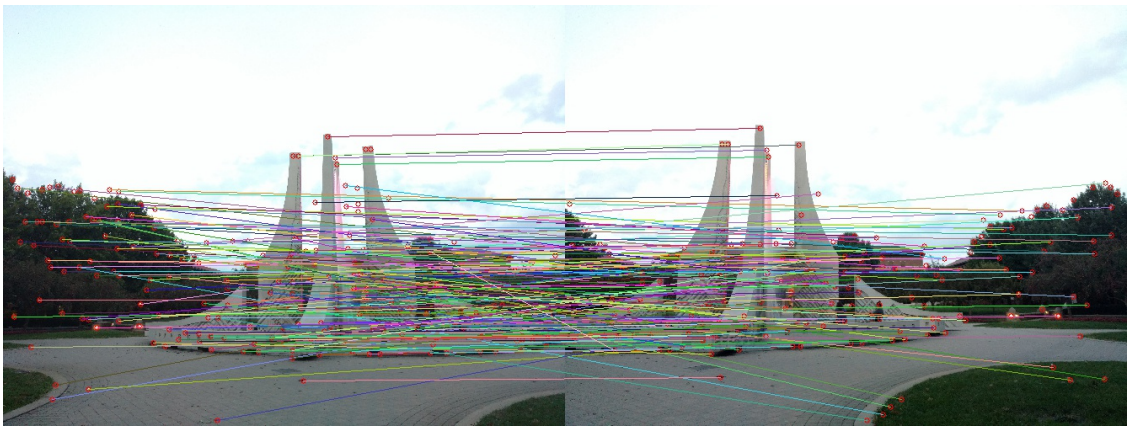

Figure 18: Harris detected corners and correspondences with ncc, $\sigma = 2.2$

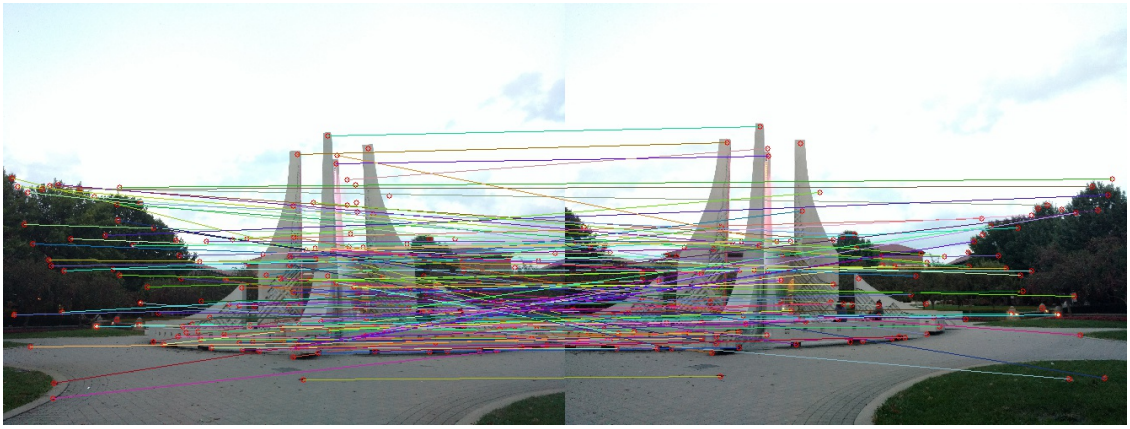

Figure 19: Harris detected corners and correspondences with ssd, $\sigma = 2.6$

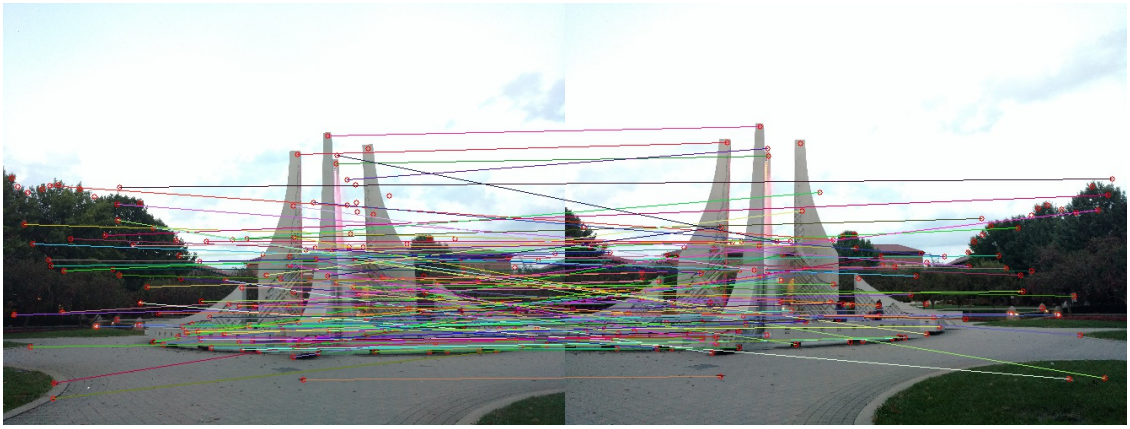

Figure 20: Harris detected corners and correspondences with ncc, $\sigma = 2.6$

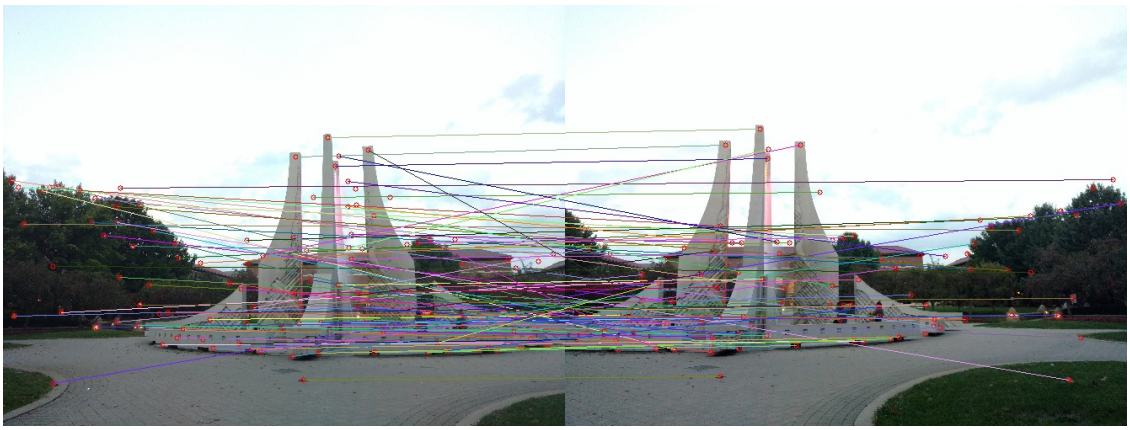

Figure 21: Harris detected corners and correspondences with ssd, $\sigma = 3.0$

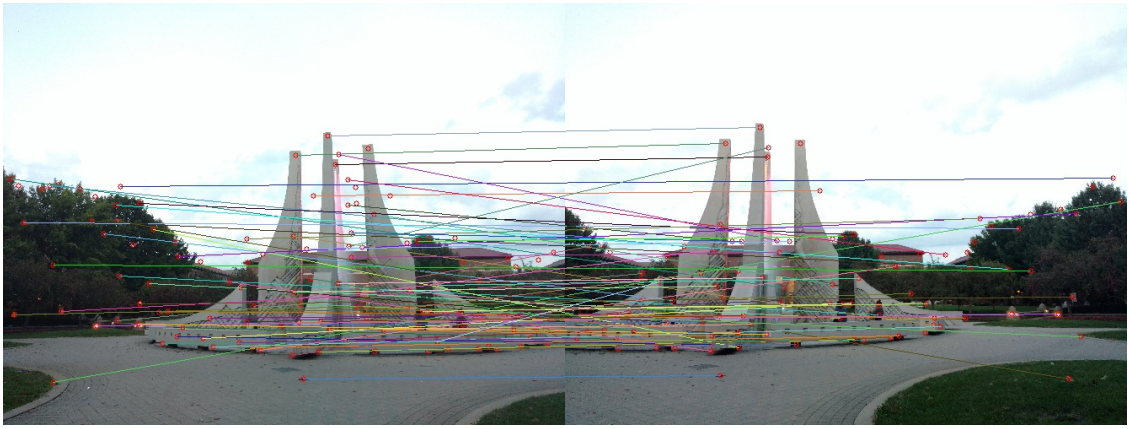

Figure 22: Harris detected corners and correspondences with ncc, $\sigma = 3.0$

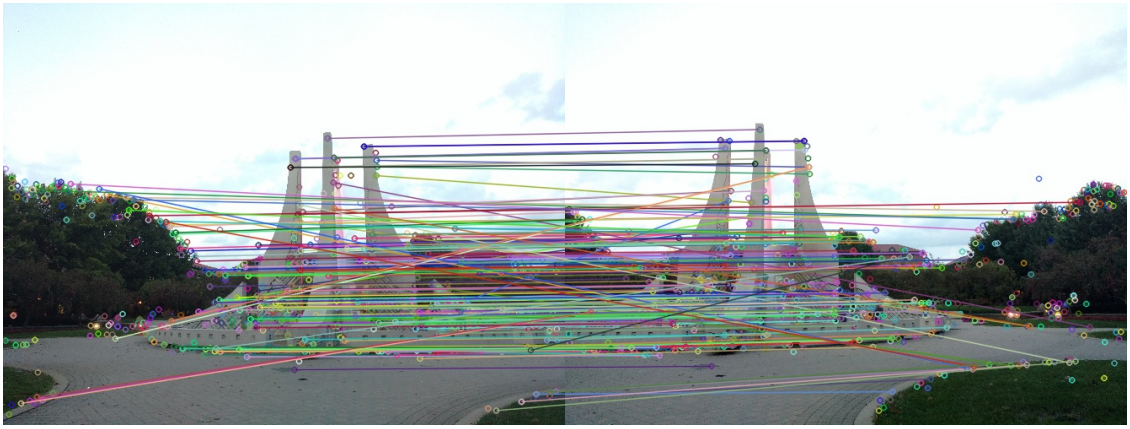

Figure 23: SIFT detected points and correspondences

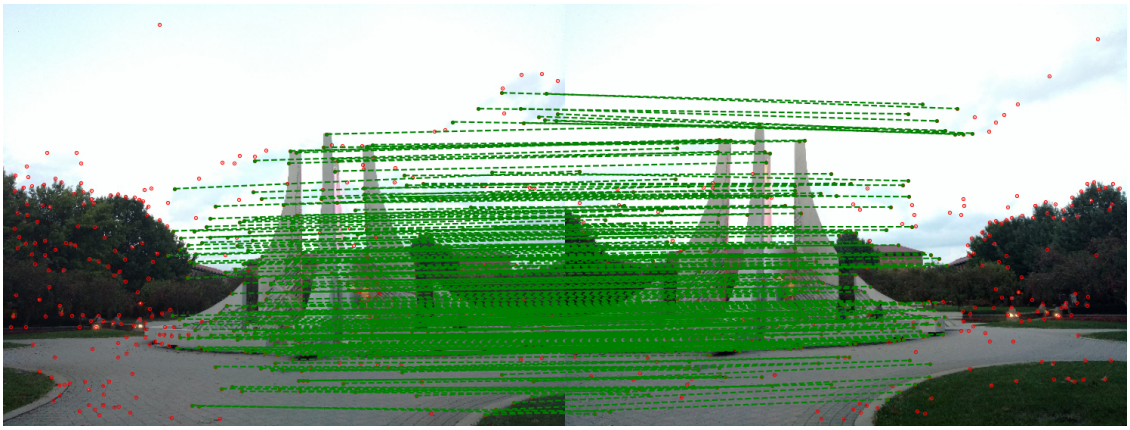

Figure 24: SuperGlue detected points and correspondences

Self-took image pair of a table

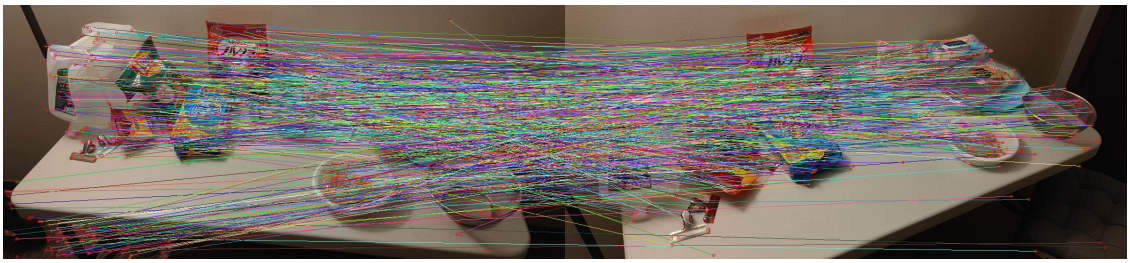

Figure 25: Harris detected corners and correspondences with ssd, $\sigma = 1.8$

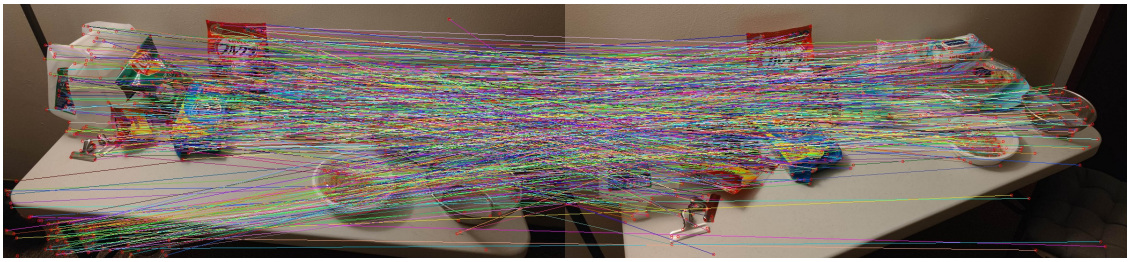

Figure 26: Harris detected corners and correspondences with ncc, $\sigma = 1.8$

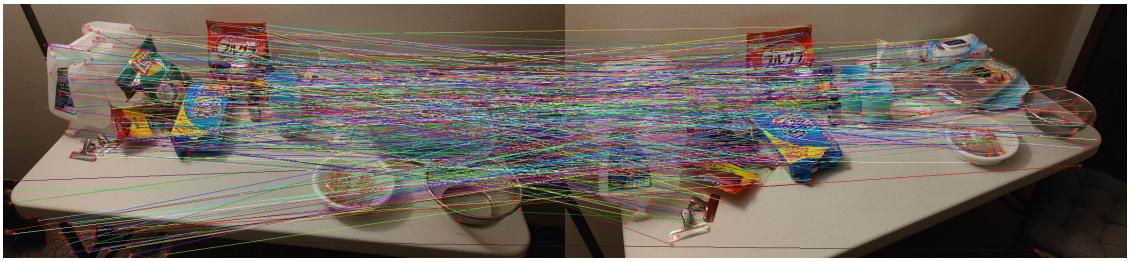

Figure 27: Harris detected corners and correspondences with ssd, $\sigma = 2.2$

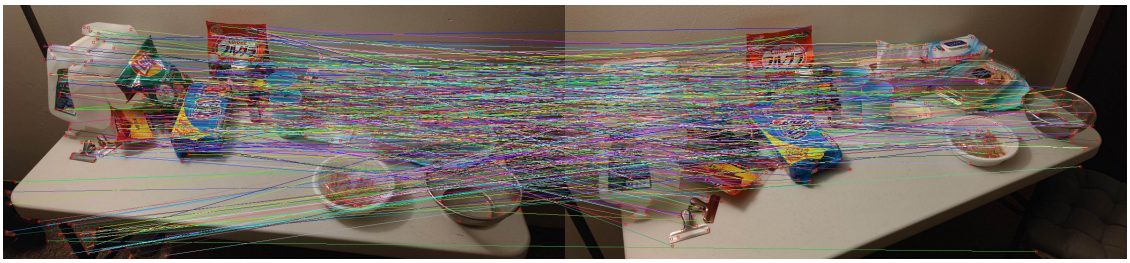

Figure 28: Harris detected corners and correspondences with ncc, $\sigma = 2.2$

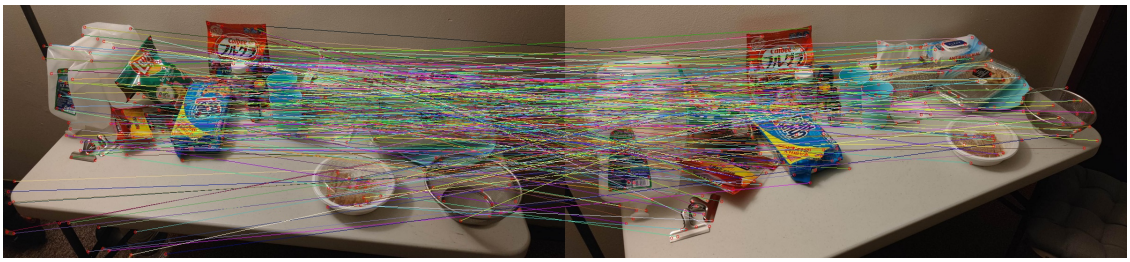

Figure 29: Harris detected corners and correspondences with ssd, $\sigma = 2.6$

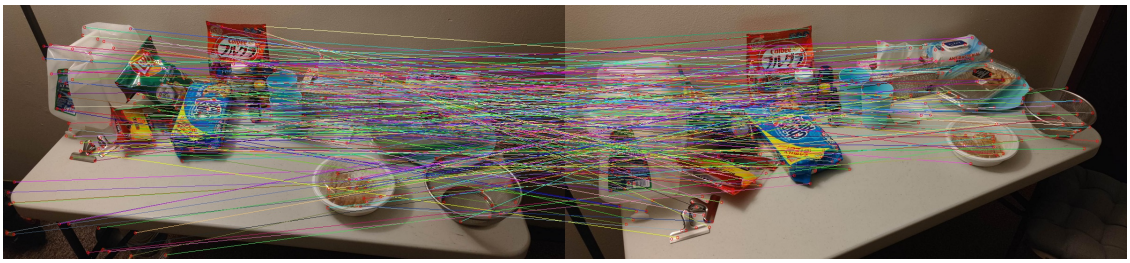

Figure 30: Harris detected corners and correspondences with ncc, $\sigma = 2.6$

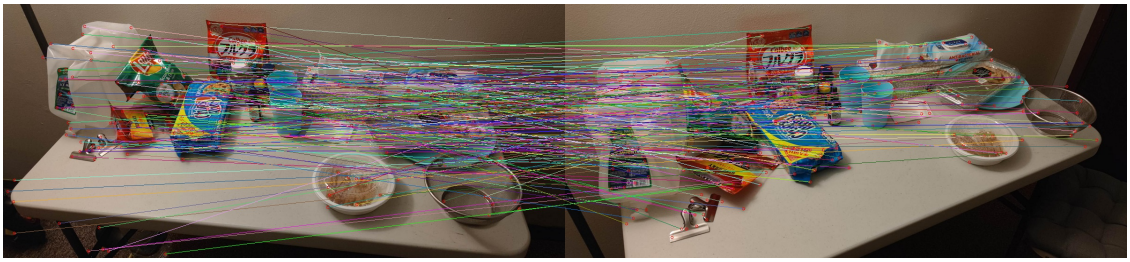

Figure 31: Harris detected corners and correspondences with ssd, $\sigma = 3.0$

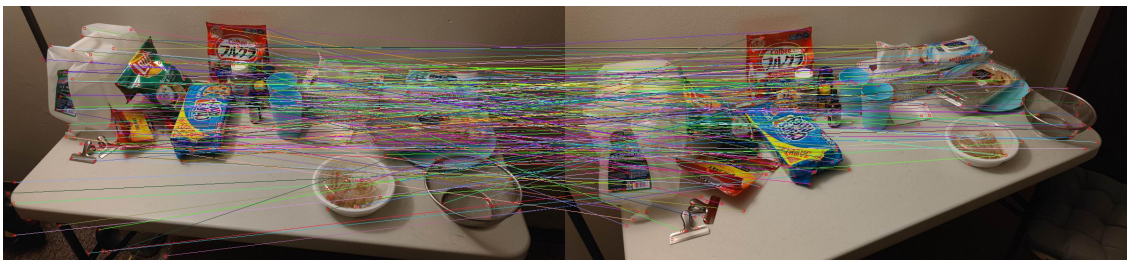

Figure 32: Harris detected corners and correspondences with ncc, $\sigma = 3.0$

Figure 33: SIFT detected points and correspondences

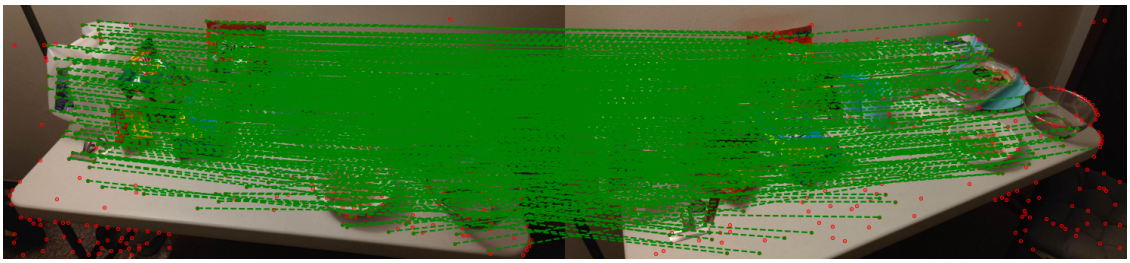

Figure 34: SuperGlue detected points and correspondences

Self-took image pair of some clothes

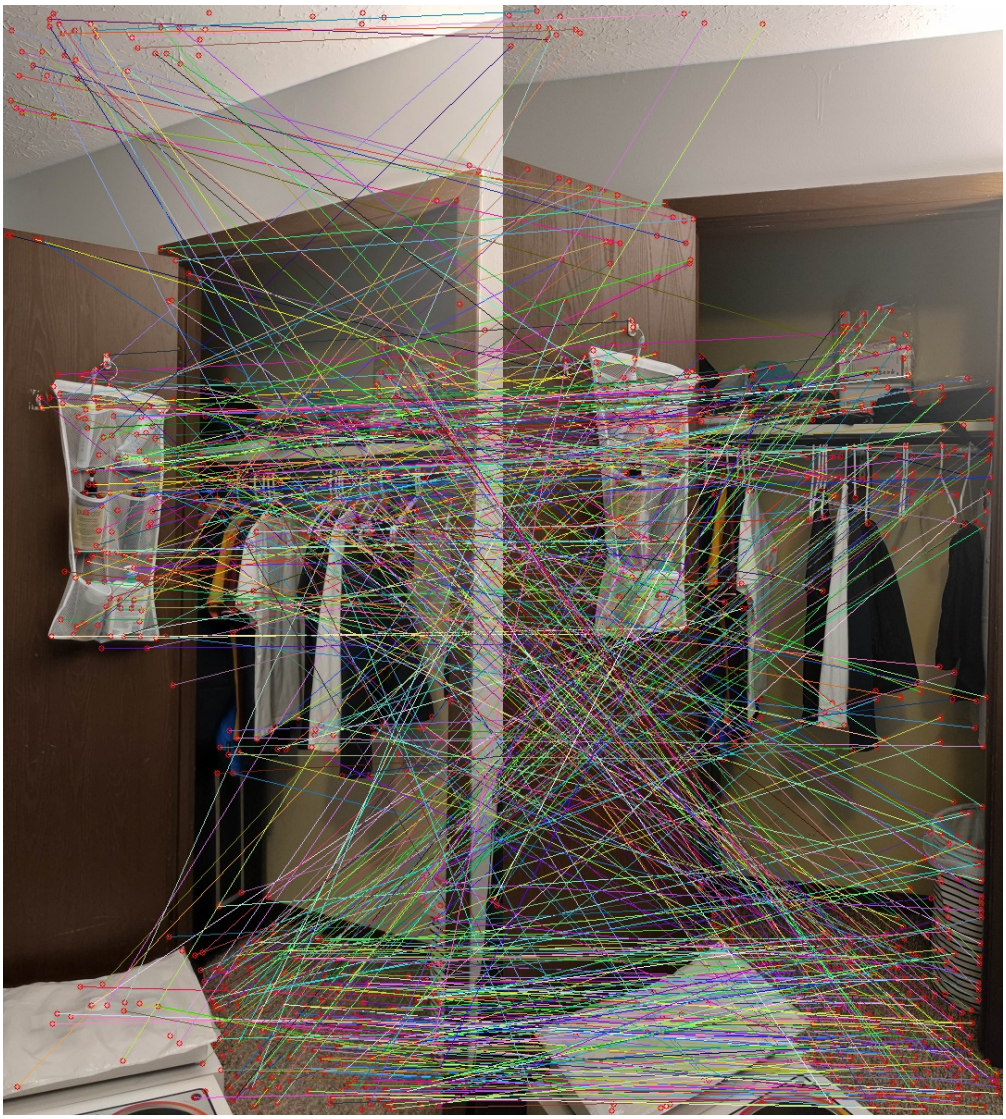

Figure 35: Harris detected corners and correspondences with ssd, $\sigma = 1.8$

Figure 36: Harris detected corners and correspondences with ncc, $\sigma = 1.8$

Figure 37: Harris detected corners and correspondences with ssd, $\sigma = 2.2$

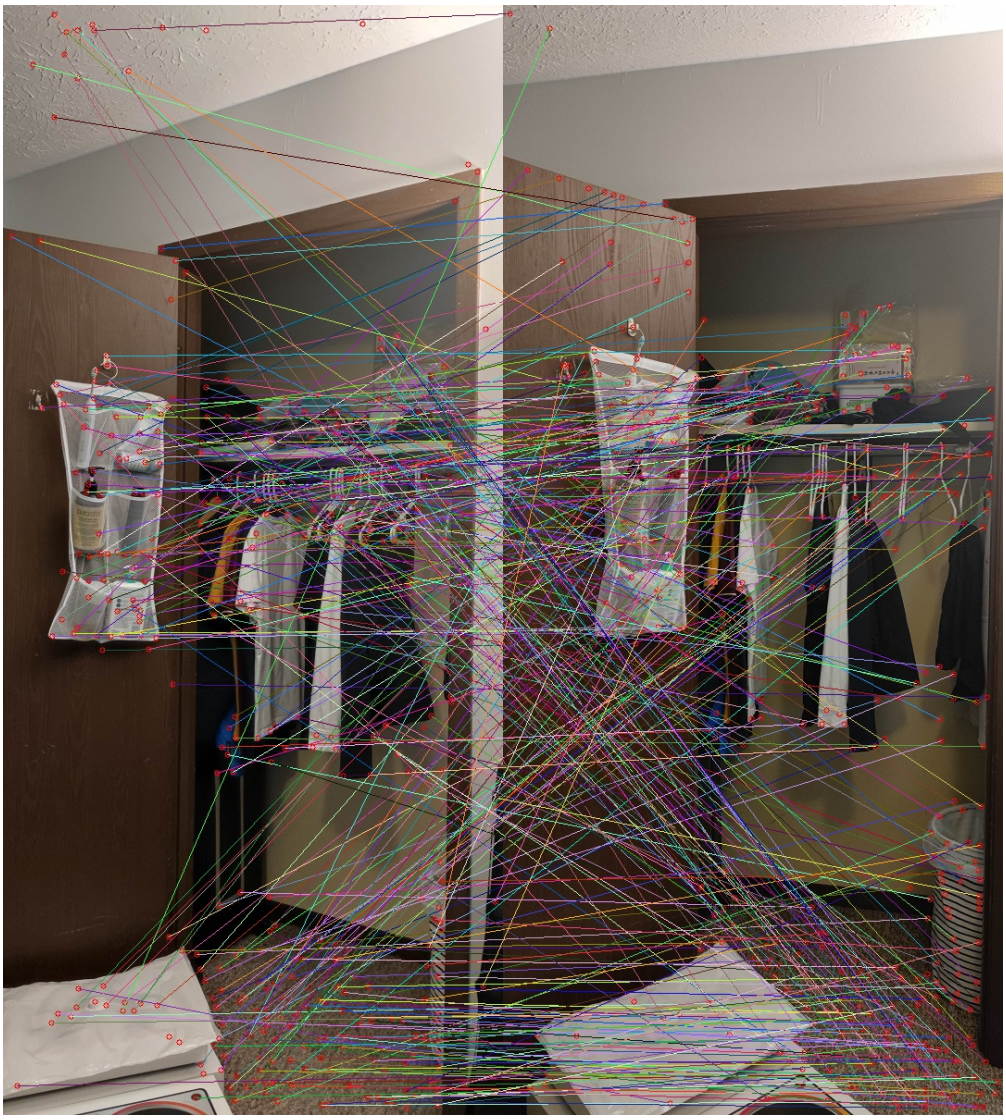

Figure 38: Harris detected corners and correspondences with ncc, $\sigma = 2.2$

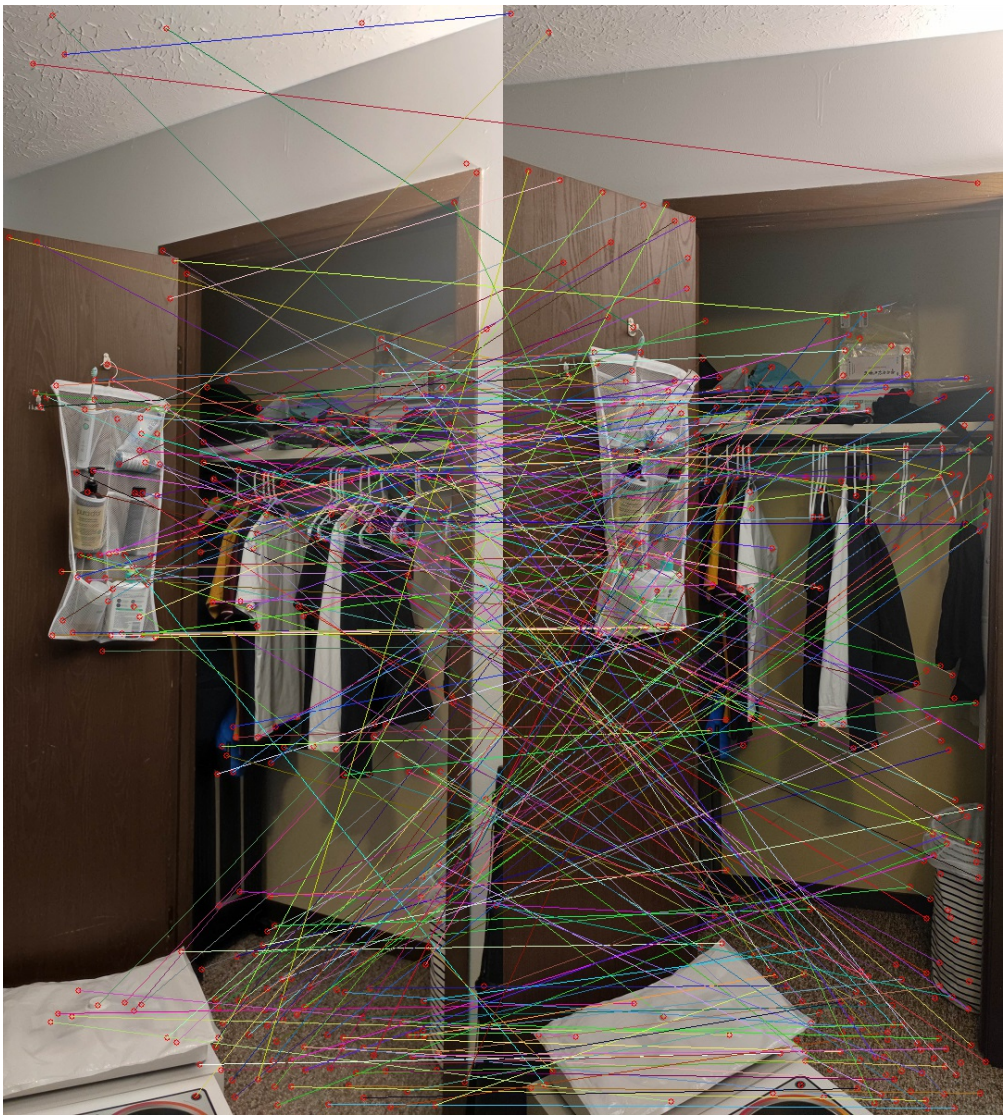

Figure 39: Harris detected corners and correspondences with ssd, $\sigma = 2.6$

Figure 40: Harris detected corners and correspondences with ncc, $\sigma = 2.6$

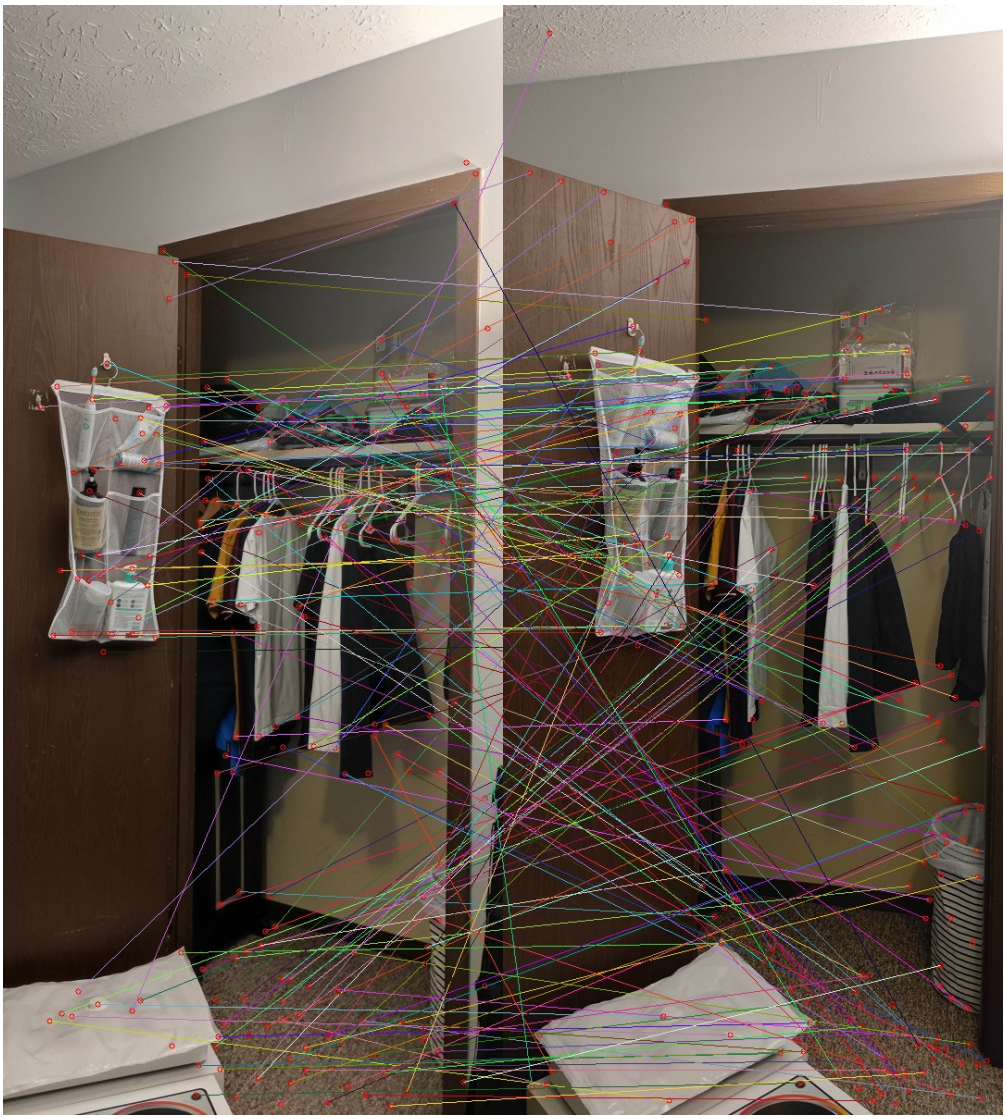

Figure 41: Harris detected corners and correspondences with ssd, $\sigma = 3.0$

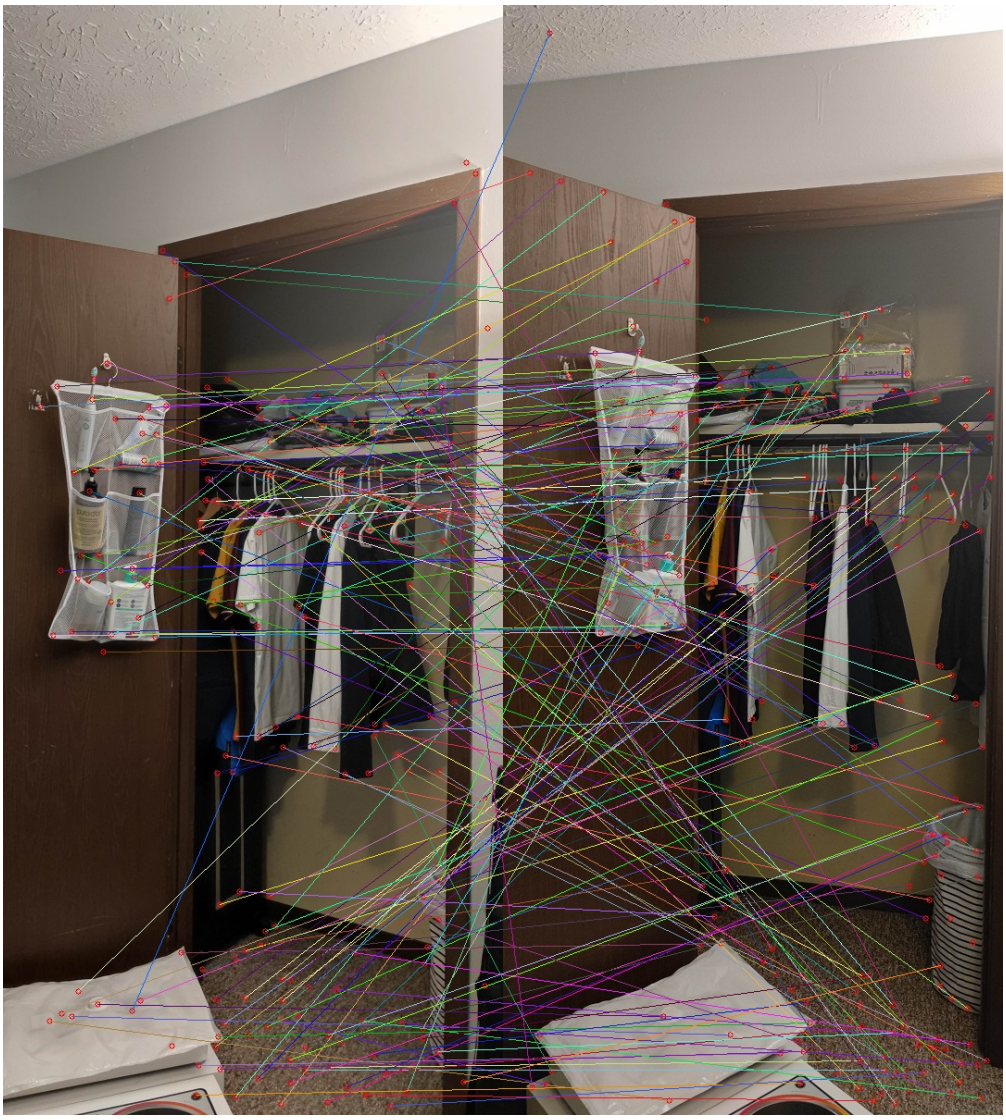

Figure 42: Harris detected corners and correspondences with ncc, $\sigma = 3.0$

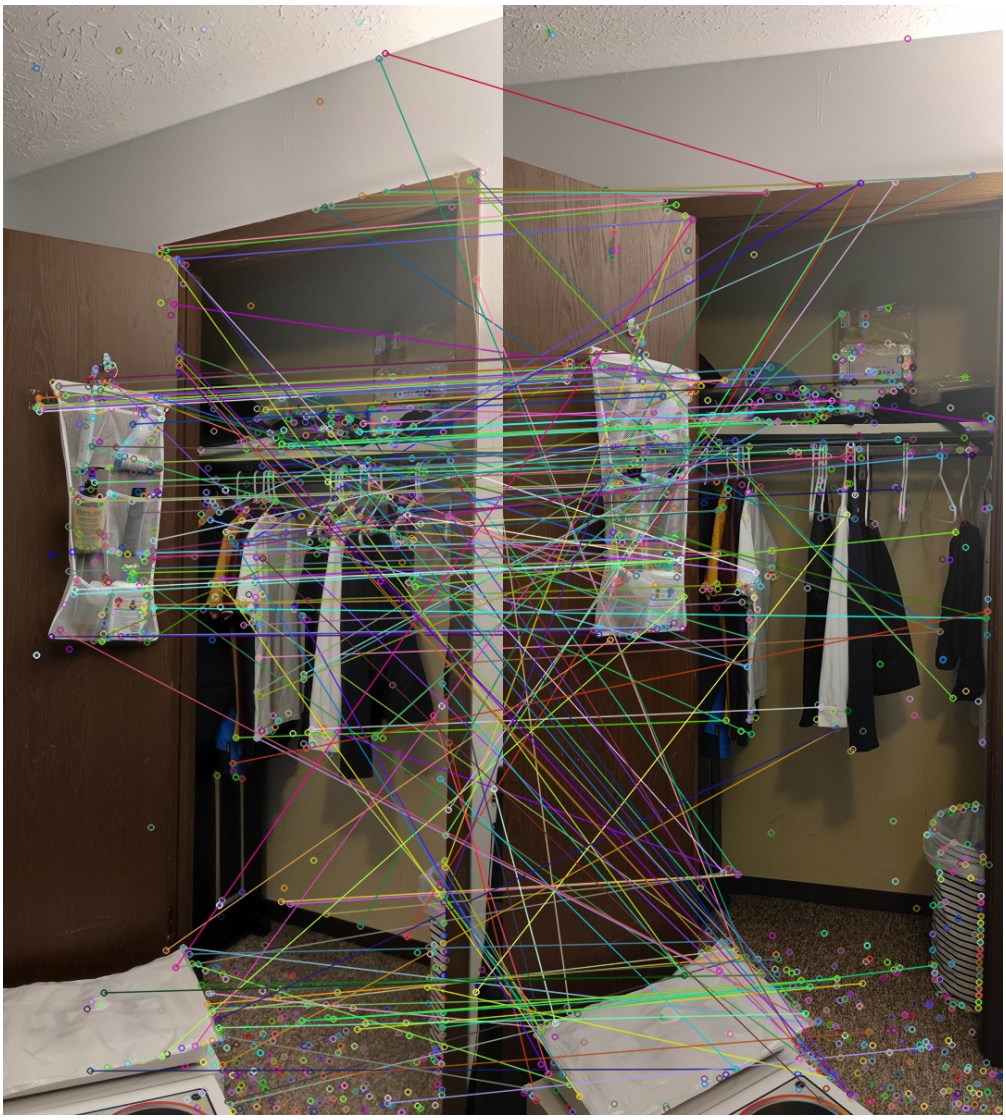

Figure 43: SIFT detected points and correspondences

Figure 44: SuperGlue detected points and correspondences

6 Observations of the Results

For the Harris Corner Detector with different scales, we can observe that more interest points can be detected with lower σ value, and less interest points with higher σ value.

For the output quality, SIFT detects more interest points and has better points matching than Harris. We can observe the matching quality by watching the overall cross and parallel lines since cross lines usually mean mismatch. The SSD and NCC give similar matching quality.

The SuperGlue gives significantly better points matching result than other methods.

The best result of Harris Corner Detector uses the parameter of $\sigma = 2.6$ and the threshold of the highest 10 percent values of R. For SSD and NCC, the window size is $M = 11$. The threshold for SSD is 3 and for NCC is 0.5.

7 Source code

```
# ECE661 HW4
# Zhengxin Jiang
# jiang839

import cv2
import numpy as np
import matplotlib . pyplot as plt
import math
import random


### function definetion ###

# The function turn a image into grayscale and normalize it
def imgToGray(img):

    img_ = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    img_ = img_.astype(np.float64)
    img_ -= img_.min()
    img_ /= img_.max()

    return img_

# The function run harris corner detection on an image
# and return the indices of interest points and the marked image
def harris(img, sigma):

    # get the gray scale image and normolize it
    img_ = imgToGray(img)

    height = img_.shape[0]
    width = img_.shape[1]
    matrix_R = np.zeros((height,width))

    # The haar kernel
    N_haar = math.ceil(sigma*2)*2

    haar_x = np.ones((N_haar, N_haar))
    haar_x[:, :int(N_haar/2)] = -1

    haar_y = np.ones((N_haar, N_haar))
    haar_y[:int(N_haar/2), :] = -1

    # derivatives
    dx = cv2.filter2D(img_, ddepth=-1, kernel=haar_x)
    dy = cv2.filter2D(img_, ddepth=-1, kernel=haar_y)

    dx2 = np.square(dx)
    dy2 = np.square(dy)
    dxy = dx*dy

    # Take the sum
```

```
    N_sum = math.ceil(sigma*5)
    if N_sum%2 == 0:
        N_sum += 1
    sum_kernel = np.ones((N_sum, N_sum))
    sum_offset = N_sum // 2

    sum_dx2 = cv2.filter2D(dx2, ddepth=-1, kernel=sum_kernel)
    sum_dy2 = cv2.filter2D(dy2, ddepth=-1, kernel=sum_kernel)
    sum_dxy = cv2.filter2D(dxy, ddepth=-1, kernel=sum_kernel)

    # Calculate the R matrix
    trace = sum_dx2 + sum_dy2
    det = sum_dx2*sum_dy2 - sum_dxy*sum_dxy

    k_vals = det / (trace ** 2 + 0.001)
    k = np.sum(k_vals) / (height*width)

    R = det - k*trace**2
    thresh = np.sort(R.flatten())[-int(len(R.flatten())*0.1)]

    idx = []

    # find corners
    for i in range(sum_offset, width - sum_offset):
        for j in range(sum_offset, height - sum_offset):

            window = R[j-sum_offset: j+sum_offset, i-sum_offset: i+sum_offset]
            if R[j, i] == window.max() and R[j, i]>thresh:
                idx.append([j, i])

    # mark corners on the image
    img_mark = img.copy()

    for idx_ in idx:
        cv2.circle(img_mark,(idx_[1],idx_[0]),3,(0,0,255))

    return idx, img_mark


# The function finds correspondence points between two marked images
def harris_corr(img_1, coord1, img_2, coord2, M = 11, method = 'ssd'):

    img1 = imgToGray(img_1)
    img2 = imgToGray(img_2)
    img_comb = np.concatenate((img_1, img_2), axis=1)

    offset = M//2
    height1 = img1.shape[0]
    width1 = img1.shape[1]
    height2 = img2.shape[0]
    width2 = img2.shape[1]

    # find correspondences
    for c1 in coord1:
```

```
        if c1[0]>offset and c1[0]<height1-offset and c1[1]>offset and c1[1]<width1-offset:
            dist_arr = []
            window1 = img1[c1[0]-offset: c1[0]+offset, c1[1]-offset: c1[1]+offset]

            # for each point in image1 find the best match in image2
            for c2 in coord2:
                if c2[0]>offset and c2[0]<height1-offset and c2[1]>offset and c2[1]<width1-
                    offset:
                    window2 = img2[c2[0]-offset: c2[0]+offset, c2[1]-offset: c2[1]+offset]

                    # compare two windows based on selected method
                    if method == 'ssd':
                        dist = np.sum((window1-window2)**2)

                    elif method == 'ncc':
                        m1 = window1.mean()
                        m2 = window2.mean()

                        t1 = np.sum((window1 - m1) * (window2 - m2))
                        t2 = np.sum((window1 - m1)**2)
                        t3 = np.sum((window2 - m2)**2)

                        dist = 1 - t1/np.sqrt(t2*t3)

                    dist_arr.append(dist)

                else:
                    dist_arr.append(9999)

            # coordinate in the combined image
            c2_match = coord2[np.argsort(dist_arr)[0]]
            c2_match[1] += width1

            # draw line with random color
            if (min(dist_arr) < 3 and method == 'ssd') or (min(dist_arr) < 0.5 and method
                == 'ncc'):
                r = random.randint(0,255)
                g = random.randint(0,255)
                b = random.randint(0,255)
                cv2.line(img_comb,(c1[1],c1[0]),(c2_match[1],c2_match[0]),(b,g,r))


    return img_comb


# The SIFT method
def sift(img1, img2):

    sift1 = cv2.xfeatures2d.SIFT_create()
    sift2 = cv2.xfeatures2d.SIFT_create()

    kp1, des1 = sift1.detectAndCompute(img1,None)
    kp2, des2 = sift2.detectAndCompute(img2,None)
```

```
    bf = cv2.BFMatcher()
    matches = bf.match(des1,des2)
    matches = sorted(matches, key = lambda x:x.distance)

    img_res = cv2.drawMatches(img1,kp1,img2,kp2,matches[:200],None,-1)

    return img_res


if __name__ == '__main__' :

    result_path = 'C:/Users/jzx/OneDrive␣-␣purdue.edu/ECE661/hw4/result␣images/'

    # images for task 1
    img_a1 = cv2.imread('HW4-Images/books_1.jpeg')
    img_a2 = cv2.imread('HW4-Images/books_2.jpeg')

    img_b1 = cv2.imread('HW4-Images/fountain_1.jpg')
    img_b2 = cv2.imread('HW4-Images/fountain_2.jpg')

    selfimg_a1 = cv2.imread('HW4-Images/table1.jpg')
    selfimg_a2 = cv2.imread('HW4-Images/table2.jpg')

    selfimg_b1 = cv2.imread('HW4-Images/cloth1.jpg')
    selfimg_b2 = cv2.imread('HW4-Images/cloth2.jpg')


    ### Task 1.1 ###
# sigma = 2.6

# harris_coord_a1, img_a1_harris = harris(img_a1, sigma)
# harris_coord_a2, img_a2_harris = harris(img_a2, sigma)
# harris_coord_b1, img_b1_harris = harris(img_b1, sigma)
# harris_coord_b2, img_b2_harris = harris(img_b2, sigma)

# img_a = np.concatenate((img_a1_harris, img_a2_harris), axis=1)
# cv2.imwrite(result_path + '1.1_a_' + str(sigma) + '.jpg', img_a)
# img_b = np.concatenate((img_b1_harris, img_b2_harris), axis=1)
# cv2.imwrite(result_path + '1.1_b_' + str(sigma) + '.jpg', img_b)

# img_a = cv2.cvtColor(img_a,cv2.COLOR_BGR2RGB)
# plt.figure()
# plt.imshow(img_a)
# img_b = cv2.cvtColor(img_b,cv2.COLOR_BGR2RGB)
# plt.figure()
# plt.imshow(img_b)


    ### Task 1.2 ###
# method = 'ncc'

# img_a_corr = harris_corr(img_a1_harris, harris_coord_a1, img_a2_harris, harris_coord_a2
    )
```

```
# cv2.imwrite(result_path + '1.2_a_' + str(sigma) +'_'+method+ '.jpg', img_a_corr)

# img_a_corr = cv2.cvtColor(img_a_corr,cv2.COLOR_BGR2RGB)
# plt.figure()
# plt.imshow(img_a_corr)

# img_b_corr = harris_corr(img_b1_harris, harris_coord_b1, img_b2_harris, harris_coord_b2
    )
# cv2.imwrite(result_path + '1.2_b_' + str(sigma) +'_'+method+ '.jpg', img_b_corr)

# img_b_corr = cv2.cvtColor(img_b_corr,cv2.COLOR_BGR2RGB)
# plt.figure()
# plt.imshow(img_b_corr)


    ### Task 1.3 SIFT ###
# img_a_sift = sift(img_a1, img_a2)
# cv2.imwrite(result_path + '1.3_a_sift' + '.jpg', img_a_sift)

# img_b_sift = sift(img_b1, img_b2)
# cv2.imwrite(result_path + '1.3_b_sift' + '.jpg', img_b_sift)


    ### Task 2.1 ###
    sigma = 3.0

    harris_coord_a1, img_a1_harris = harris(selfimg_a1, sigma)
    harris_coord_a2, img_a2_harris = harris(selfimg_a2, sigma)
    harris_coord_b1, img_b1_harris = harris(selfimg_b1, sigma)
    harris_coord_b2, img_b2_harris = harris(selfimg_b2, sigma)

# img_a = np.concatenate((img_a1_harris, img_a2_harris), axis=1)
# cv2.imwrite(result_path + '2.1_a_' + str(sigma) + '.jpg', img_a)
# img_b = np.concatenate((img_b1_harris, img_b2_harris), axis=1)
# cv2.imwrite(result_path + '2.1_b_' + str(sigma) + '.jpg', img_b)

# img_a = cv2.cvtColor(img_a,cv2.COLOR_BGR2RGB)
# plt.figure()
# plt.imshow(img_a)
# img_b = cv2.cvtColor(img_b,cv2.COLOR_BGR2RGB)
# plt.figure()
# plt.imshow(img_b)


    ### Task 2.2 ###
    method = 'ssd'

    img_a_corr = harris_corr(img_a1_harris, harris_coord_a1, img_a2_harris,
        harris_coord_a2)
    cv2.imwrite(result_path + '2.2_a_' + str(sigma) +'_'+method+ '.jpg', img_a_corr)

    img_a_corr = cv2.cvtColor(img_a_corr,cv2.COLOR_BGR2RGB)
    plt.figure()
    plt.imshow(img_a_corr)
```

```
    img_b_corr = harris_corr(img_b1_harris, harris_coord_b1, img_b2_harris,
        harris_coord_b2)
    cv2.imwrite(result_path + '2.2_b_' + str(sigma) +'_'+method+ '.jpg', img_b_corr)

    img_b_corr = cv2.cvtColor(img_b_corr,cv2.COLOR_BGR2RGB)
    plt.figure()
    plt.imshow(img_b_corr)


    ### Task 2.3 SIFT ###
# img_a_sift = sift(selfimg_a1, selfimg_a2)
# cv2.imwrite(result_path + '2.3_a_sift' + '.jpg', img_a_sift)

# img_b_sift = sift(selfimg_b1, selfimg_b2)
# cv2.imwrite(result_path + '2.3_b_sift' + '.jpg', img_b_sift)
```