ECE 661 Homework 4

Weichen Xu

xu1363@purdue.edu

1 Theory Question

The Laplacian of Gaussian (LoG) and Difference of Gaussian (DoG) for an image are written as

$$LoG[f(x,y)] = \nabla^2 ff(x,y,\sigma)$$

$$DoG[f(x,y)] = \frac{\partial}{\partial\sigma} ff(x,y,\sigma)$$

Since we know that $\nabla^2 ff(x,y,\sigma) = f(x,y) * h(x,y,\sigma)$, while $h(x,y,\sigma) = \frac{-1}{2\pi\sigma^4}(2 - \frac{x^2+y^2}{\sigma^2})e^{-\frac{x^2+y^2}{\sigma^2}}$.

Then we can write DoG as following simplified form,

$$\frac{\partial}{\partial\sigma} ff(x,y,\sigma) = \frac{-\sigma}{2\pi\sigma^4}\iint f(x',y')\left[2 - \frac{(x-x')^2+(y-y')^2}{\sigma^2}\right] e^{-\frac{(x-x')^2+(y-y')^2}{2\sigma^2}} dx'dy'$$

$$= \sigma f(x,y) * h(x,y,\sigma)$$

So, we can compute DoG to obtain LoG, instead of computing LoG directly.

The benefit of computing DoG is more computationally efficient than computing LoG. The main reason for this is computing the DoG of an image can be separated to two directional 1D smoothing in x y directions, while LoG is not separable. Also in the digital implementation, for the same value σ, the DoG tend to have smaller operator size than the LoG.

2 Harris Corner Detector and SSD NCC Metrics

The Harris corner detector will typically detect the significant variations of grayscale pixel values in x and y directions in the image.

Step 1: The detector works at different scales. The first part is choosing a proper scale (σ). Once the scale is chosen, we can build the Haar filters in x y directions, the filter is of the size of $M \times M$, where M is the smallest even integer regarding to 4σ. Take $\sigma = 1.5$ as an example.

$$H_x(\sigma = 1.5) = \begin{bmatrix} -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \end{bmatrix}$$

$$H_y(\sigma = 1.5) = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

Step 2: Haar filters in two directions are convolved with the image to get the derivatives d_x $d_y$in x y directions.

Step 3: For each location in the image, build a C matrix from the d_x d_y with a $N \times N$ neighborhood, where N is the nearest odd integer of 5σ.

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

Step 4: We compute the Harris response of C with $R = \det(C) - trace(C)^2 * k$, where k is constant, usually set as 0.05.

Step 5: After we calculate the Harris response for every location in the image, we will get the map of response, while only the positive values are kept, the negative values are set to 0. Finally, we will use two metrics to decide the correspondence pairs. The first metric is Sum of Squared Differences (SSD), the second metric is Normalized Cross Correlation (NCC), while certain thresholds are applied to filter potential good pairs.

$$SSD = \sum_x \sum_y (f_1(x,y) - f_2(x,y))^2$$

$$NCC = \frac{\sum_x \sum_y (f_1(x,y) - m_1)(f_2(x,y) - m_2)}{\sqrt{\left[\sum_x \sum_y (f_1(x,y) - m_1)^2\right]\left[\sum_x \sum_y (f_2(x,y) - m_2)^2\right]}}$$

While f_1 is the image 1, m_1 is the mean value of image patch of f_1 in grayscale within window size of $(M+1) \times (M+1)$, similarly for f_2 and m_2.

To find the best matching pairs of interest points, we will use two metrics to decide.

For SSD metric, I calculate the SSD result for two neighborhoods around interest points. Then setting the threshold to be the three times of the minimal SSD values for all pairs, pairs with SSD value smaller than the threshold will be treated as valid pairs.

For NCC metric, I calculate the NCC result for two neighborhoods around interest points. Then setting the threshold to be the 0.8 times of the maximal NCC values for all pairs, pairs with NCC value greater than the threshold will be treated as valid pairs.

3 Experiment Results of Harris Corner Detector and SSD NCC Metric

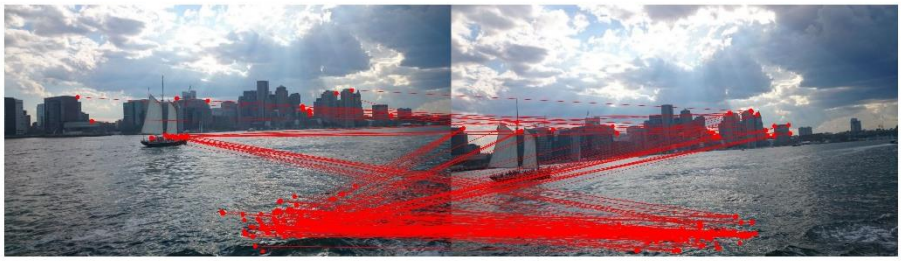

Figure 1. SSD on image pair 1 with $\sigma = 0.5$

Figure 2. NCC on image pair 1 with $\sigma = 0.5$

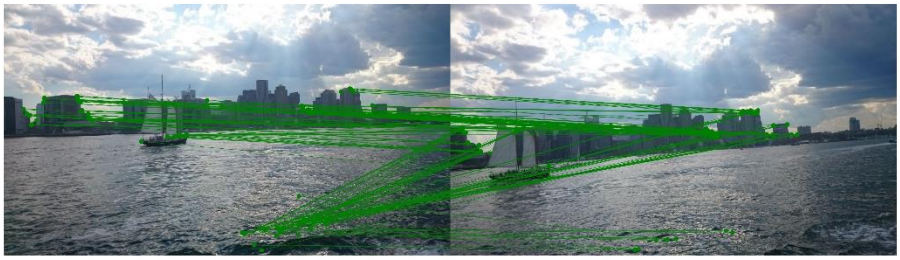

Figure 3. SSD on image pair 1 with $\sigma = 1.0$

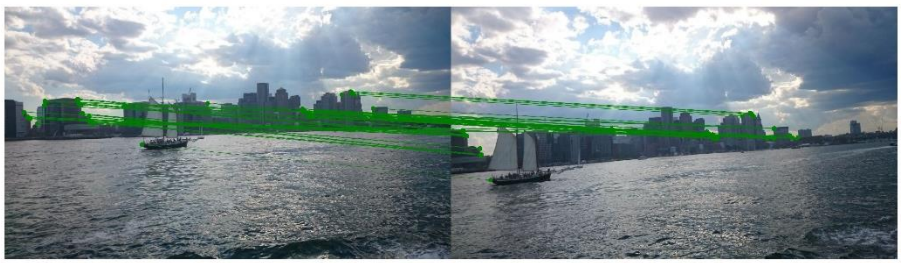

Figure 4. NCC on image pair 1 with $\sigma = 1.0$

Figure 5. SSD on image pair 1 with $\sigma = 1.5$

Figure 6. NCC on image pair 1 with $\sigma = 1.5$

Figure 7. SSD on image pair 1 with $\sigma = 2.0$

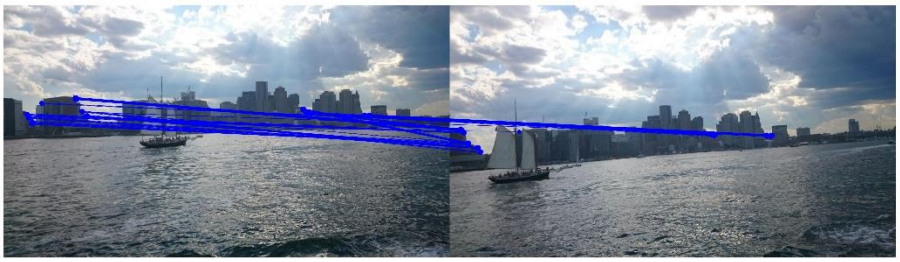

Figure 8. NCC on image pair 1 with $\sigma = 2.0$

Figure 9. SSD on image pair 2 with $\sigma = 0.5$

Figure 10. NCC on image pair 2 with $\sigma = 0.5$

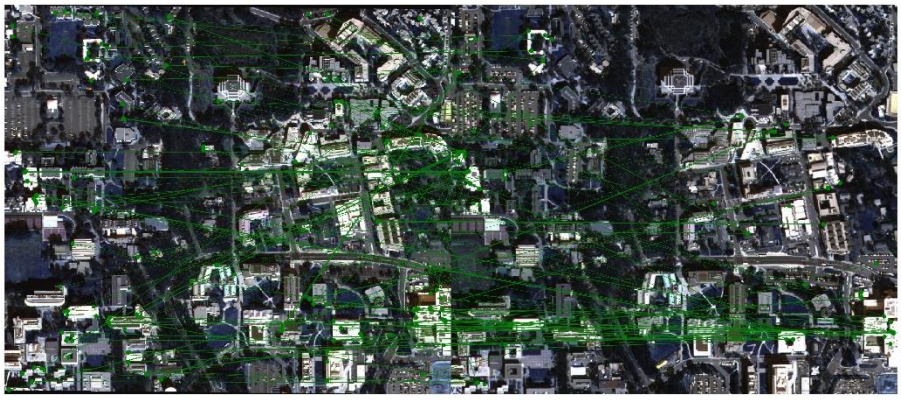

Figure 11. SSD on image pair 2 with $\sigma = 1.0$

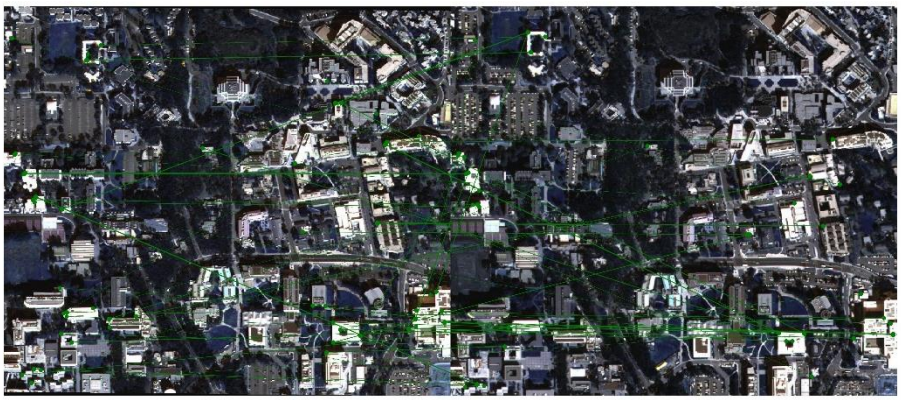

Figure 12. NCC on image pair 2 with $\sigma = 1.0$

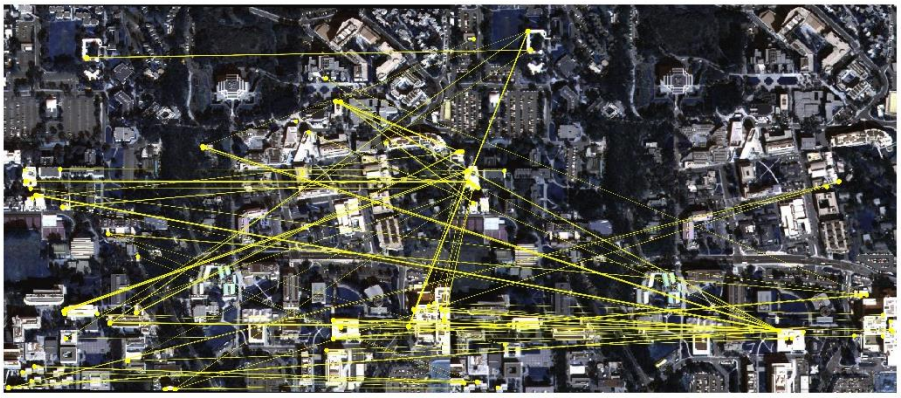

Figure 13. SSD on image pair 2 with $\sigma = 1.5$

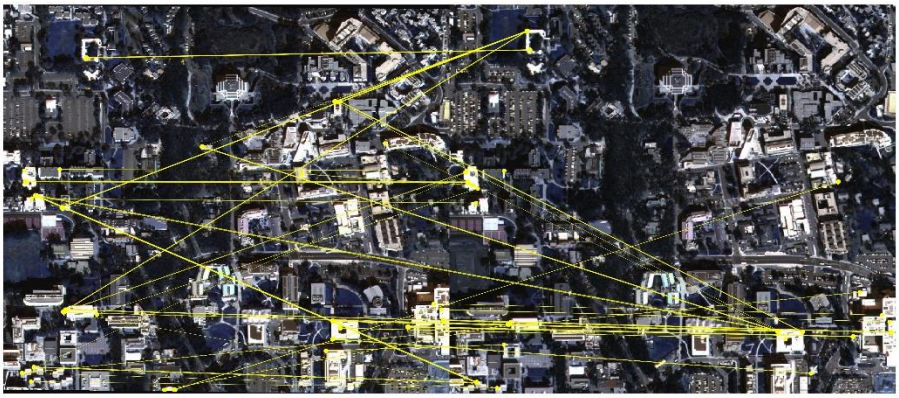

Figure 14. NCC on image pair 2 with $\sigma = 1.5$

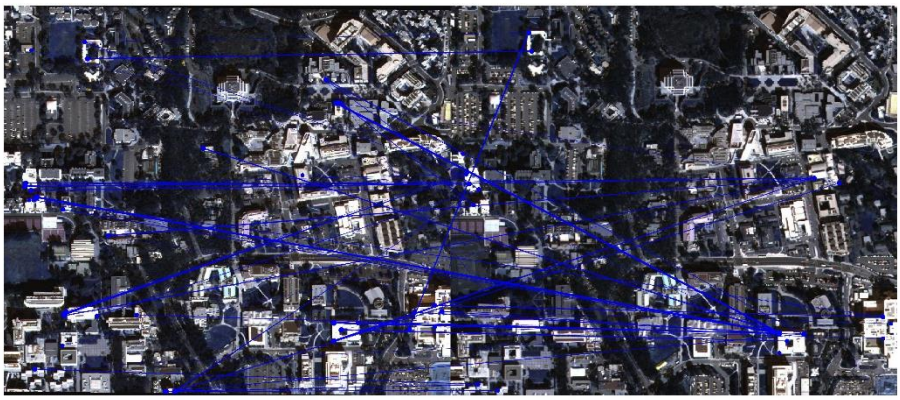

Figure 15. SSD on image pair 2 with $\sigma = 2.0$

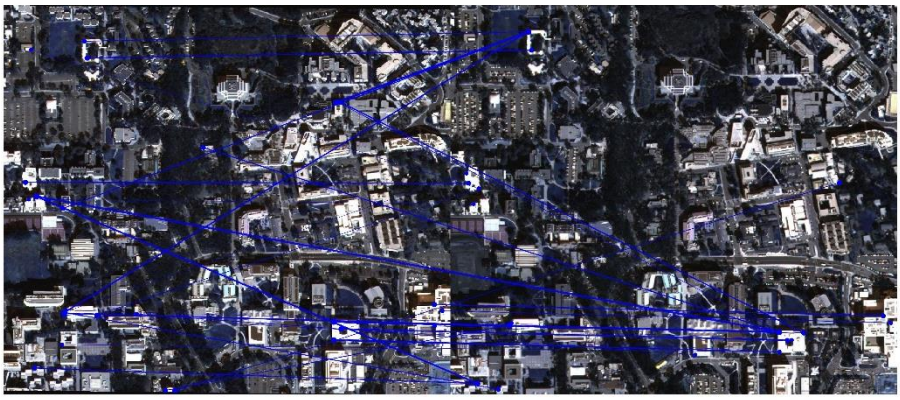

Figure 16. NCC on image pair 2 with $\sigma = 2.0$

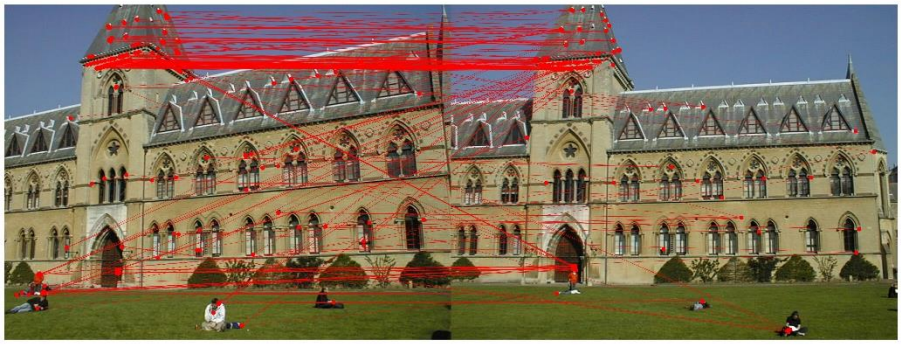

Figure 17. SSD on image pair 3 with $\sigma = 0.5$

Figure 18. NCC on image pair 3 with $\sigma = 0.5$

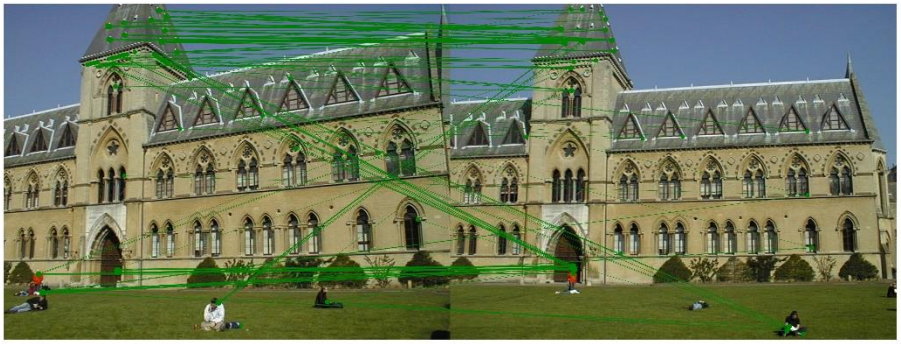

Figure 19. SSD on image pair 3 with $\sigma = 1.0$

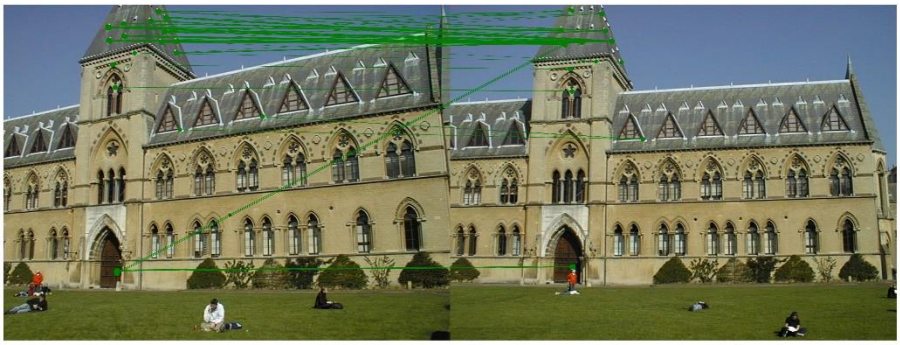

Figure 20. NCC on image pair 3 with $\sigma = 1.0$

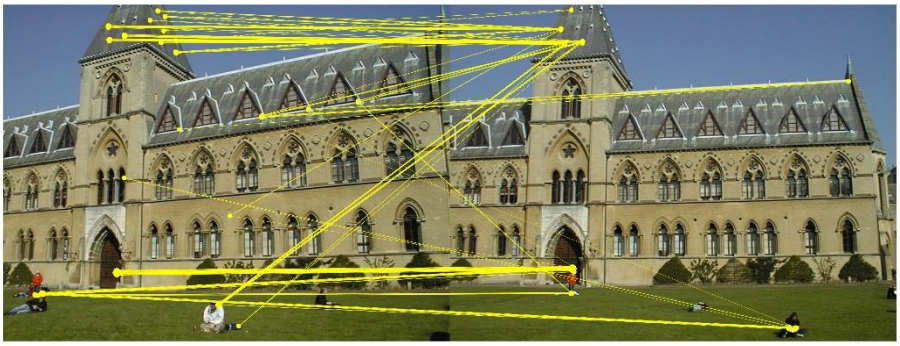

Figure 21. SSD on image pair 3 with $\sigma = 1.5$

Figure 22. NCC on image pair 3 with $\sigma = 1.5$

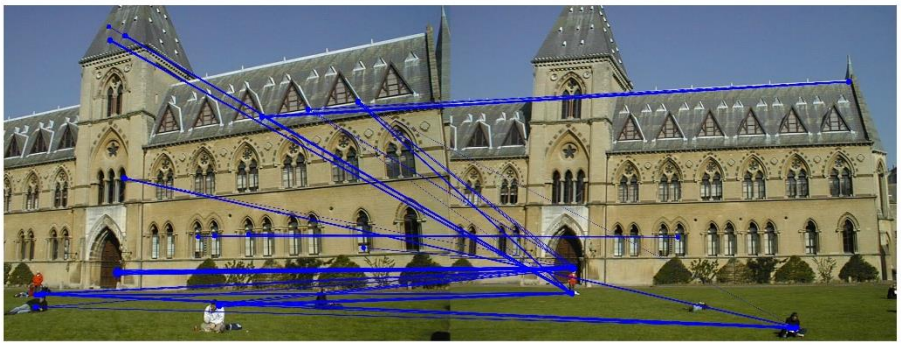

Figure 23. SSD on image pair 3 with $\sigma = 2.0$

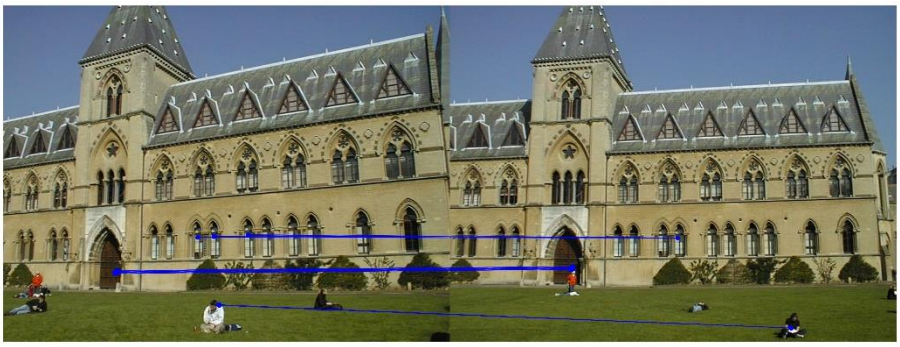

Figure 24. NCC on image pair 3 with $\sigma = 2.0$

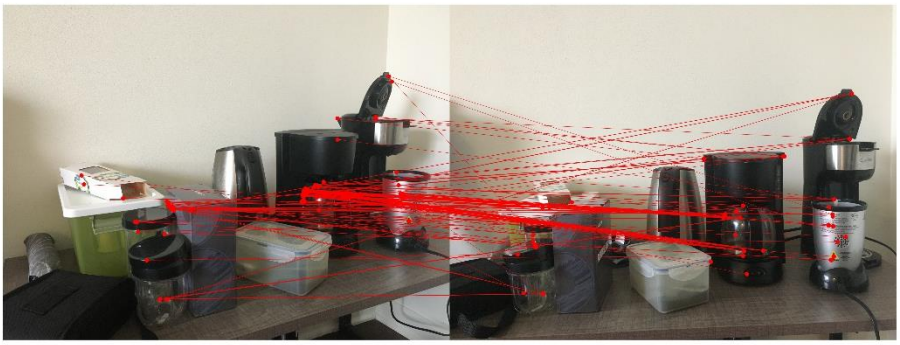

Figure 25. SSD on my image pair 1 with $\sigma = 0.5$

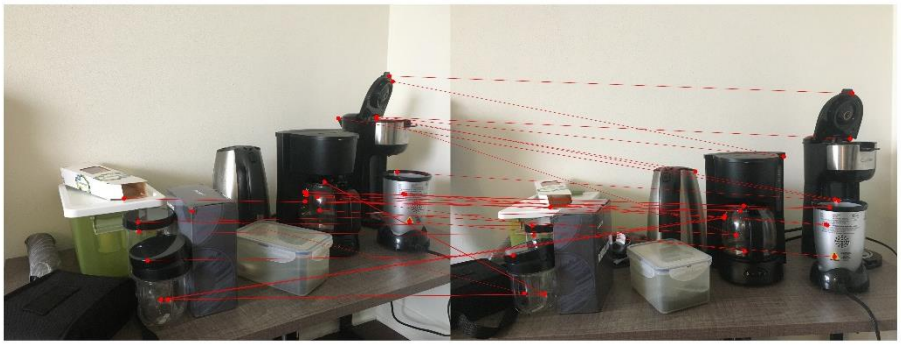

Figure 26. NCC on my image pair 1 with $\sigma = 0.5$

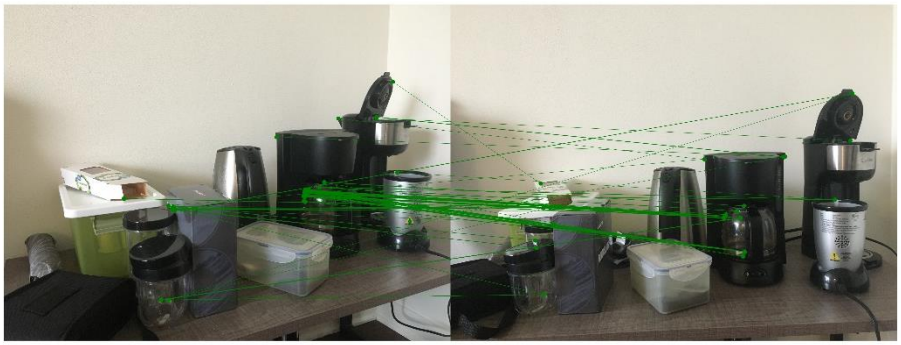

Figure 27. SSD on my image pair 1 with $\sigma = 1.0$

Figure 28. NCC on my image pair 1 with $\sigma = 1.0$

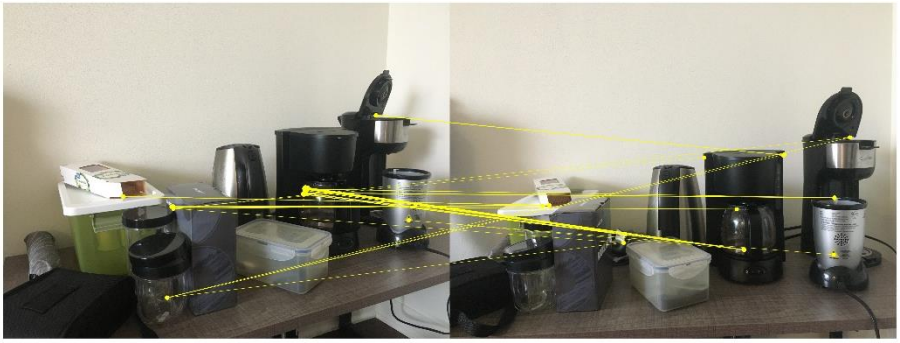

Figure 29. SSD on my image pair 1 with $\sigma = 1.5$

Figure 30. NCC on my image pair 1 with $\sigma = 1.5$

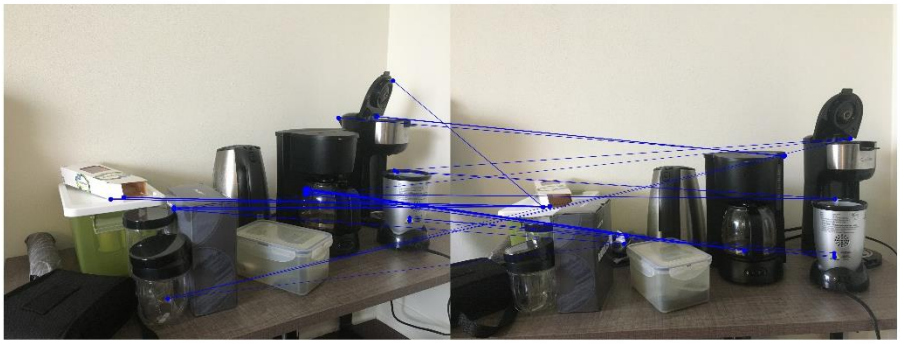

Figure 31. SSD on my image pair 1 with $\sigma = 2.0$

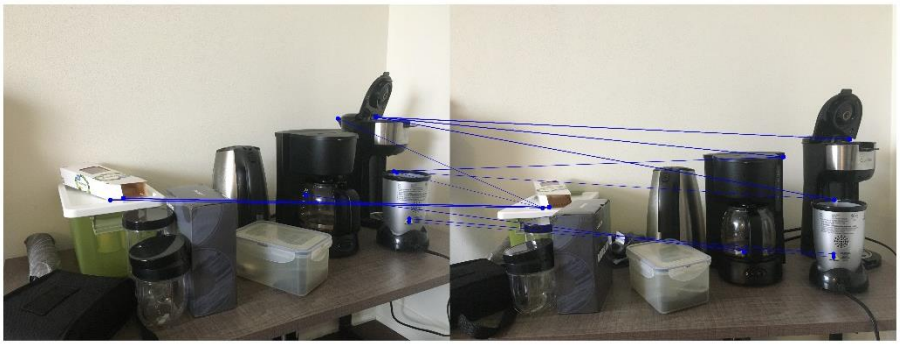

Figure 32. NCC on my image pair 1 with $\sigma = 2.0$

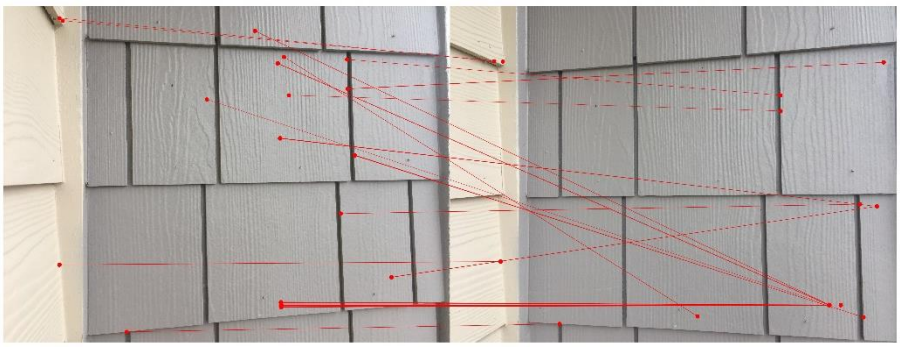

Figure 33. SSD on my image pair 2 with $\sigma = 0.5$

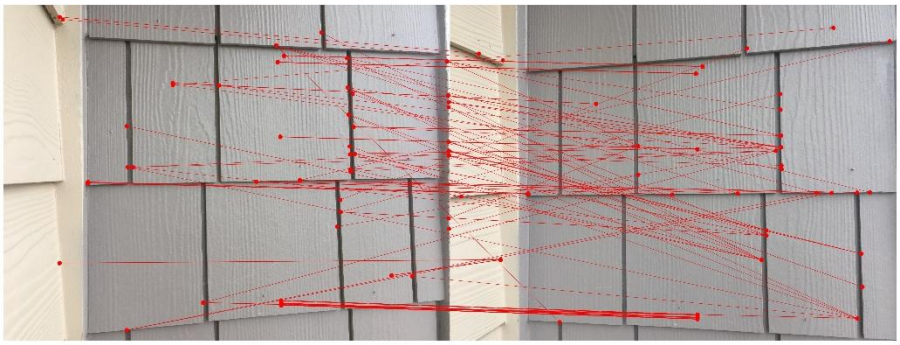

Figure 34. NCC on my image pair 2 with $\sigma = 0.5$

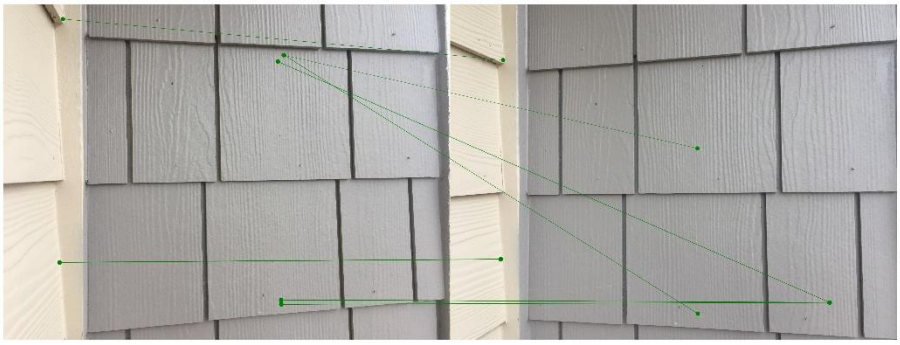

Figure 35. SSD on my image pair 2 with $\sigma = 1.0$

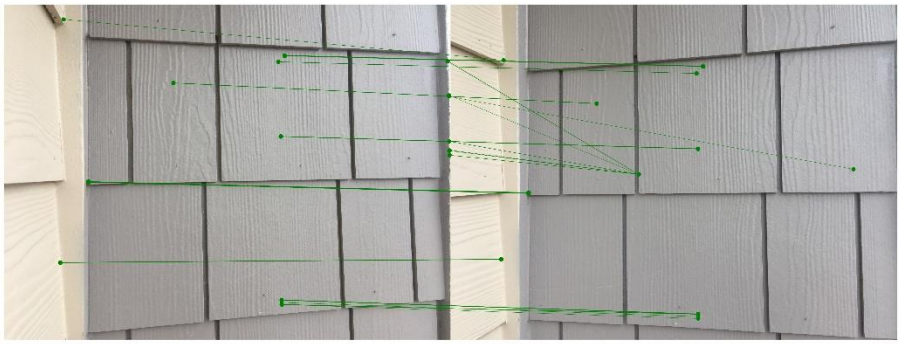

Figure 36. NCC on my image pair 2 with $\sigma = 1.0$

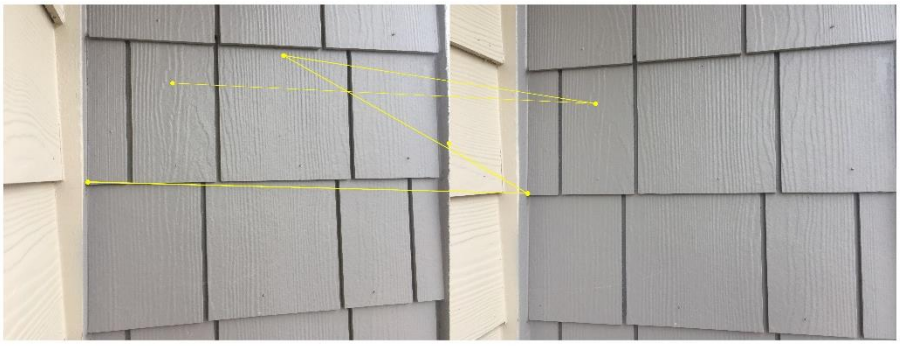

Figure 37. SSD on my image pair 2 with $\sigma = 1.5$

Figure 38. NCC on my image pair 2 with $\sigma = 1.5$

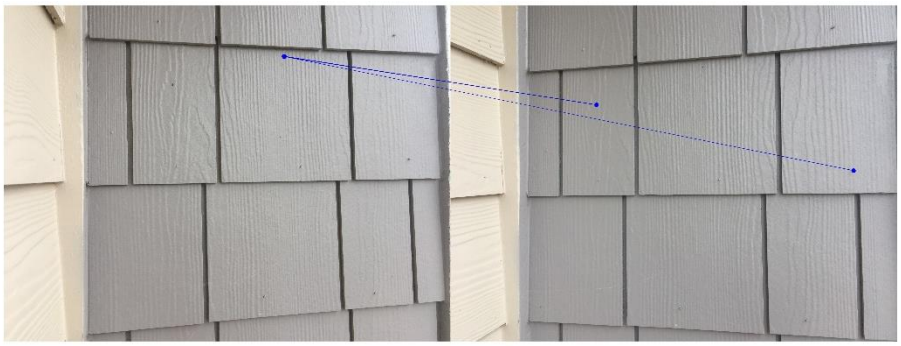

Figure 39. SSD on my image pair 2 with $\sigma = 2.0$

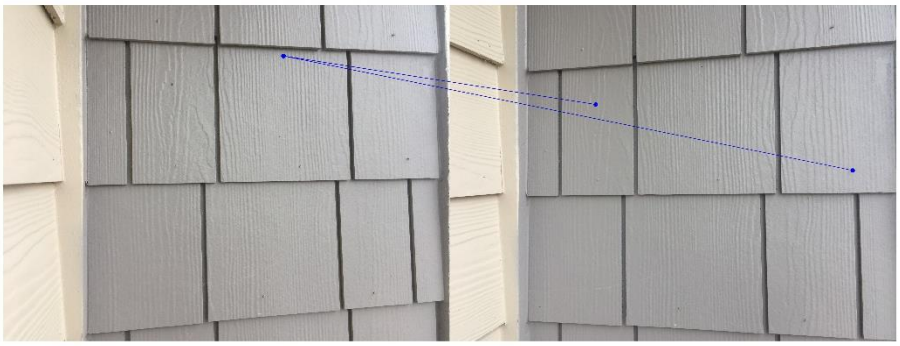

Figure 40. NCC on my image pair 2 with $\sigma = 2.0$

4 Scale Invariant Feature Transform (SIFT)

Step 1: Calculate the local extrema using Difference of Gaussian (DoG) in the (x, y, σ) scale space, then we can get the potential key points.

Step 2: As σ increase from a lower octave to a higher octave, the size of the precision level decreases. To address the more correct location of a local extrema, use Taylor series expansion to find its precise location of the extrema in the image.

Step 3: Not all key points are needed, we need to filter out the key points which are on the edges or of low contrast, thresholds are applied.

Step 4: We will assign orientation to remaining key points, based on local image gradient directions. This step will maintain the invariance to rotation.

Step 5: A descriptor will be created for each key point, a 128-dimensional vector.

5 Experiment Results of Scale Invariant Feature Transform (SIFT)

Figure 41. SIFT on image pair 1

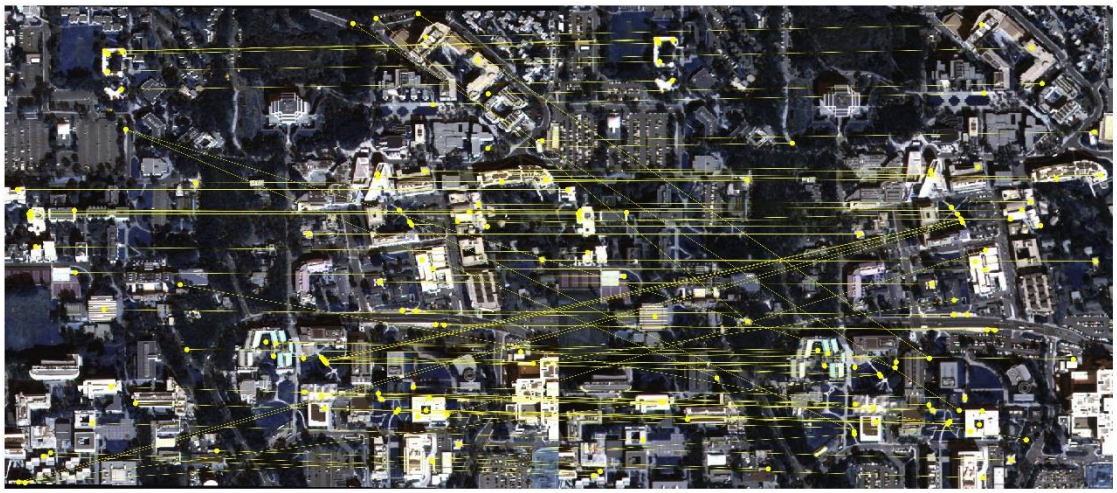

Figure 42. SIFT on image pair 2

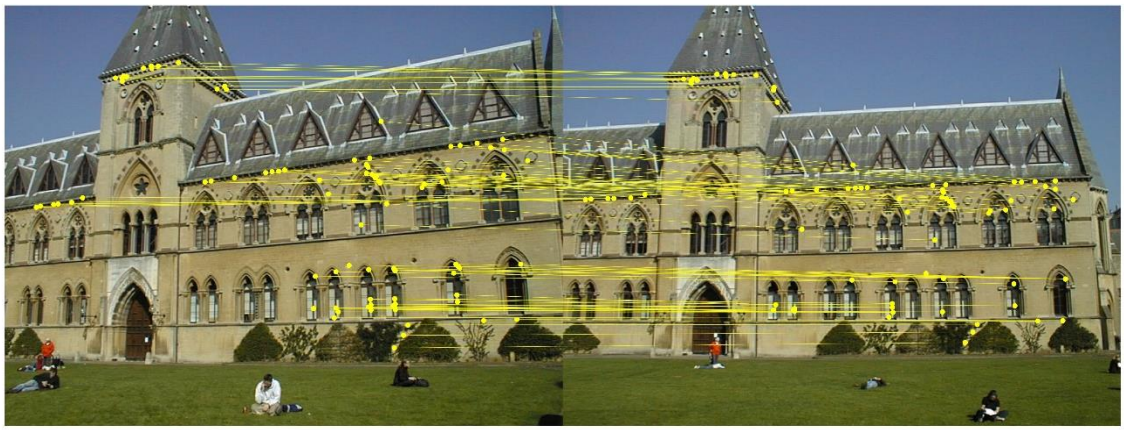

Figure 43. SIFT on image pair 3

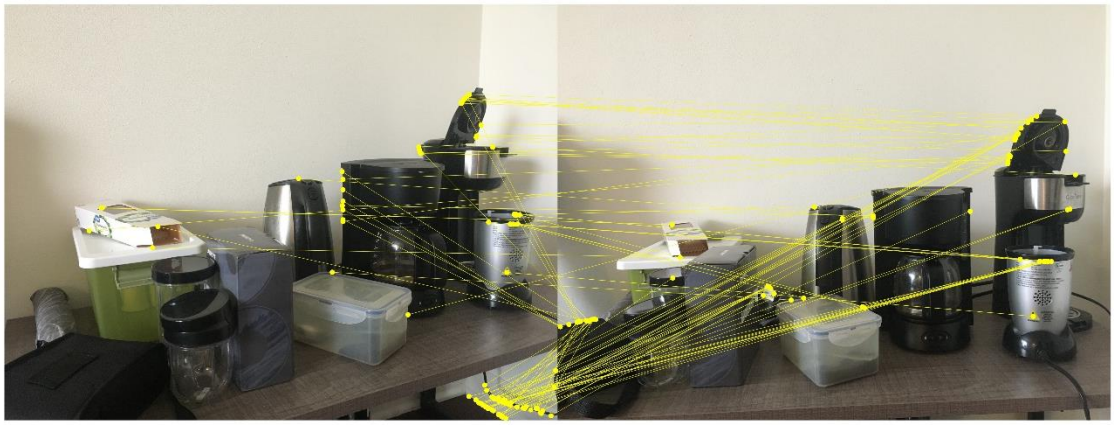

Figure 44. SIFT on my image pair 1

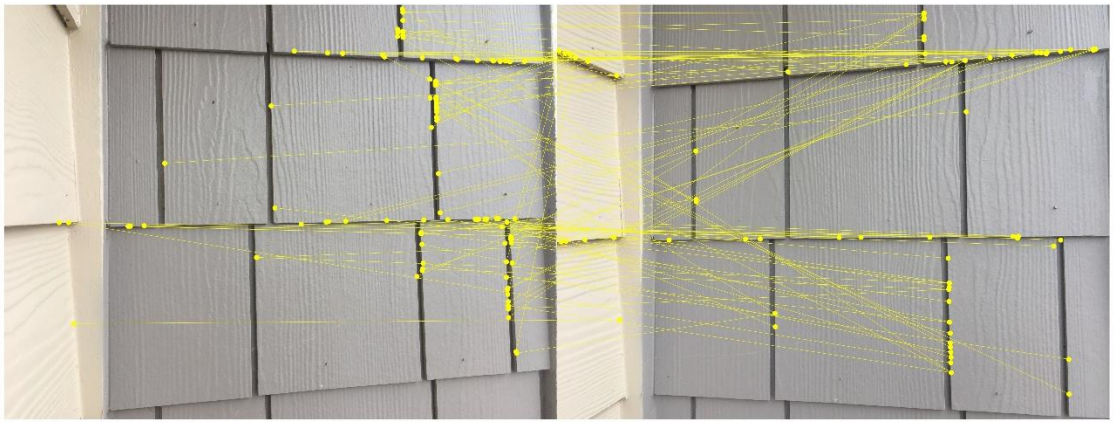

Figure 45. SIFT on my image pair 2

6 Observations from Experiment Results

It can be observed clearly that the Harris corner detector method's precision of finding interest points depends heavily on the scale we choose. Usually the greater the scale is, the less potential interest points will be obtained. Even though the Harris corner detector can find lots of interest points, most of them are prone to be mismatched with pairs, for both two matching metrics. In my experiments, NCC metric is better than the SSD metric, since NCC results fewer mismatching pairs.

SIFT method tends to be more robust than Harris corner detector in the sense of giving more consistent and robust interest points. Since SIFT make use of the properties of scale invariance, orientation and illumination.

7 Experiment Parameters

Harris Corner Detector	Scale (σ)	0.5, 1.0, 1.5, 2.0
	# of interest points	2000
SSD	Ratio of threshold with minimal value	3.0
NCC	Ratio of threshold with maximal value	0.9
SIFT	# of features	2000
	Threshold of # of best matching pairs	100

Table 1. Parameters used in the implementation

8 Code

```
#!/usr/bin/env python
# coding: utf-8



import numpy as np
import cv2
import matplotlib.pyplot as plt
from scipy import signal



def GetHaar (sigma):
    # Get the haar filter in x,y directions
    k = np.int(np.ceil(sigma * 4))
    if k % 2 == 1:
```

```
        k = k + 1

    dx = np.ones((k, k))
    dx[:, :np.int(k/2)] = -1

    dy = np.ones((k, k))
    dy[np.int(k/2):, :] = -1



    return dx, dy

def GetHarrisCorners (img, sigma):
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img_gray = np.double(img_gray) / 255
    height, width = img_gray.shape

    # get the derivative of image from Haar filter
    dx, dy = GetHaar(sigma)

    img_dx = signal.convolve2d(img_gray, dx, mode='same')
    img_dy = signal.convolve2d(img_gray, dy, mode='same')
    # get the second order derivative
    img_dx2 = np.multiply(img_dx, img_dx)
    img_dy2 = np.multiply(img_dy, img_dy)
    img_dxdy = np.multiply(img_dx, img_dy)
    kernel_size = np.int(np.ceil(sigma * 5))
    if kernel_size % 2 == 0:
        kernel_size = kernel_size + 1
    kernel = np.ones((kernel_size, kernel_size))
```

```
# calculate the sum in 5sigma window
img_dx2_sum = signal.correlate2d(img_dx2, kernel, mode="same")
img_dy2_sum = signal.correlate2d(img_dy2, kernel, mode="same")
img_dxdy_sum = signal.correlate2d(img_dxdy, kernel, mode="same")

# get the det and trace of C
C_det = np.multiply(img_dx2_sum, img_dy2_sum) - np.multiply(img_dxdy_sum, img_dxdy_sum)
C_trace = img_dx2_sum + img_dy2_sum
C_trace2 = np.multiply(C_trace, C_trace)

# find the points with pixel value bigger than threshold
response = C_det - 0.05 * C_trace2
#print(response.shape)
response_sorted = sorted(response.flatten())
response_sorted = response_sorted[::-1]
threshold = response_sorted[400-1]

corners = []
for j in range(height):
   for i in range(width):
      if response[j, i] >= threshold:
         corners.append([i, j])



return corners
```

```
def GetValidCorners(corners, img, window_size = 21):
    # remove the interest points lying close to the border of image
    corners_valid = []
    h = int(window_size / 2)
    for pt in corners:
        if h < pt[0] < img.shape[1] - h and h < pt[1] < img.shape[0] - h:
            corners_valid.append(pt)

    #print(len(corners_valid))

    return corners_valid

def GetSSD(patch1, patch2):
    # calculate the SDD value from the formula
    sum = np.sum(np.sum(np.square(patch1 - patch2)))
    return sum

def Correspondence_SSD (img1, corners1, img2, corners2, window_size = 21):
    # get the relevant correspondent pairs with SSD
    value_min = []
    index_min = []
    h = int(window_size / 2)

    corners1 = GetValidCorners(corners1, img1, window_size)
    corners2 = GetValidCorners(corners2, img2, window_size)

    for pt1 in corners1:
        x1 = pt1[0]
```

```
    y1 = pt1[1]
    patch1 = img1[y1 - h: y1 + h, x1 - h: x1 + h]

    SSD_all = []
    for pt2 in corners2:
        x2 = pt2[0]
        y2 = pt2[1]
        patch2 = img2[y2 - h: y2 + h, x2 - h: x2 + h]
        SSD_all.append(GetSSD(patch1, patch2))

    # calculate the current image patch's minimum SSD value
    idx = np.argmin(SSD_all)
    value_min.append(SSD_all[idx])
    index_min.append(idx)

# set the threshold of keeping pairs
threshold = np.min(value_min) * 3
#print(np.min(value_min))
#print(np.max(value_min))
corners1_corr = []
corners2_corr = []

for i in range(len(value_min)):
    if value_min[i] < threshold:
        corners1_corr.append(corners1[i])
        corners2_corr.append(corners2[index_min[i]])

return corners1_corr, corners2_corr
```

```
def GetNCC(patch1, patch2):
    # calculate the NCC value from the formula
    mean1 = np.mean(patch1)
    mean2 = np.mean(patch2)

    up = np.sum(np.sum(np.multiply(patch1 - mean1, patch2 - mean2)))
    down = np.sqrt(np.sum(np.sum(np.square(patch1 - mean1)) * np.sum(np.square(patch2 - mean2))))
    sum = up / down

    return sum



def Correspondence_NCC (img1, corners1, img2, corners2, window_size = 21):
    # get the relevant correspondent pairs with NCC
    value_max = []
    index_max = []
    h = int(window_size / 2)

    corners1 = GetValidCorners(corners1, img1, window_size)
    corners2 = GetValidCorners(corners2, img2, window_size)
    for pt1 in corners1:
        x1 = pt1[0]
        y1 = pt1[1]
        patch1 = img1[y1 - h: y1 + h, x1 - h: x1 + h]

        NCC_all = []
        for pt2 in corners2:
            x2 = pt2[0]
            y2 = pt2[1]
```

```
            patch2 = img2[y2 - h: y2 + h, x2 - h: x2 + h]
            NCC_all.append(GetNCC(patch1, patch2))

        # calculate the current image patch's minimum NCC value
        idx = np.argmax(NCC_all)
        value_max.append(NCC_all[idx])
        index_max.append(idx)

    threshold = np.max(value_max) * 0.9

    corners1_corr = []
    corners2_corr = []

    for i in range(len(value_max)):
        if value_max[i] > threshold:
            corners1_corr.append(corners1[i])
            corners2_corr.append(corners2[index_max[i]])

    return corners1_corr, corners2_corr

def plot_correspondece(img1, l1, img2, l2, color = 'yellow'):
    # plot the correspondence pairs in the image pair
    h, w, c = img1.shape
    # combine image pairs to one image
    img_synthetic = np.zeros((h, 2*w, c), dtype = np.uint8)
    img_synthetic[:, 0:w, :] = img1
    img_synthetic[:, w:2*w, :] = img2

    plt.figure(dpi=1200)
```

```
    plt.imshow(cv2.cvtColor(img_synthetic, cv2.COLOR_BGR2RGB))

    for i in range(len(l1)):
        plt.scatter(l1[i][0], l1[i][1], color = color, s = 0.3)
        plt.scatter(l2[i][0] + w, l2[i][1], color = color, s = 0.3)

        plt.plot([l1[i][0], l2[i][0] + w], [l1[i][1], l2[i][1]], color = color,  linewidth = 0.1)




def getSIFT(img, nfeatures = 2000):
    # get the desired number of interest points and descriptive vectors
    sift = cv2.xfeatures2d.SIFT_create(nfeatures = nfeatures)
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    kps, des = sift.detectAndCompute(img_gray, None)

    return kps, des

def Correspondence_SIFT(img1, kps1, des1, img2, kps2, des2):
    # Keep only the 100 most relavant pairs of interest points
    corr1 = []
    corr2 = []
    dist_min_all = []
    for i in range(len(kps1)):
        pt1 = kps1[i].pt
        pt1 = np.round(pt1)

        # measure the Euclidean distance between two descriptor vector
```

```
        dist_min = np.sqrt(np.sum(np.multiply(des1[i] - des2[0], des1[i] - des2[0])))
        for j in range(len(kps2)):
            pt2 = kps2[j].pt
            pt2 = np.round(pt2)

            dist = np.sqrt(np.sum(np.multiply(des1[i] - des2[j], des1[i] - des2[j])))
            if dist <= dist_min:
                dist_min = dist
                pt1_corr = pt2

        corr1.append(pt1)
        corr2.append(pt1_corr)
        dist_min_all.append(dist_min)

    dist_sorted = sorted(dist_min_all)
    corr1_sorted = []
    corr2_sorted = []

    for i in range(100):
        idx = np.where(dist_min_all == dist_sorted[i])
        idx = idx[0][0]
        corr1_sorted.append(corr1[idx])
        corr2_sorted.append(corr2[idx])
    return corr1_sorted, corr2_sorted


# pair 1
sigmas = [0.5, 1, 1.5, 2]
directory = "/home/xu1363/Documents/ECE 661/hw4/hw4_Task1_Images/pair1/"
```

```
file1 = "1.JPG"
file2 = "2.JPG"
img1 = cv2.imread(directory+file1,cv2.IMREAD_COLOR)
img2 = cv2.imread(directory+file2,cv2.IMREAD_COLOR)
colors = ['red', 'green', 'yellow', 'blue']
if img1.shape == img2.shape:
   print("same size")
else:
   print("different size")
   img2 = cv2.resize(img2, (img1.shape[1], img1.shape[0]), interpolation = cv2.INTER_AREA)

i = 0
for sigma in sigmas:
   color = colors[i]
   i = i + 1
   corners1 = GetHarrisCorners(img1, sigma)
   corners2 = GetHarrisCorners(img2, sigma)

   # SSD metric
   l1, l2 = Correspondence_SSD(img1, corners1, img2, corners2, window_size = 21)
   plot_correspondece(img1, l1, img2, l2, color = color)
   plt.axis('off')
   plt.savefig("/home/xu1363/Documents/ECE
661/hw4/output/pair1/"+"pair1_SSD_sigma"+str(sigma)+".jpeg")

   # NCC metric
   l1, l2 = Correspondence_NCC(img1, corners1, img2, corners2, window_size = 21)
   plot_correspondece(img1, l1, img2, l2, color = color)
   plt.axis('off')
```

```
    plt.savefig("/home/xu1363/Documents/ECE
661/hw4/output/pair1/"+"pair1_NCC_sigma"+str(sigma)+".jpeg")


# SIFT
kps1, des1 = getSIFT(img1)
kps2, des2 = getSIFT(img2)
l1, l2 = Correspondence_SIFT(img1, kps1, des1, img2, kps2, des2)
plot_correspondece(img1, l1, img2, l2)
plt.axis('off')
plt.savefig("/home/xu1363/Documents/ECE 661/hw4/output/pair1/"+"pair1_SIFT.jpeg")




# pair 2
sigmas = [0.5, 1, 1.5, 2]
directory = "/home/xu1363/Documents/ECE 661/hw4/hw4_Task1_Images/pair2/"
file1 = "1.JPG"
file2 = "2.JPG"
img1 = cv2.imread(directory+file1,cv2.IMREAD_COLOR)
img2 = cv2.imread(directory+file2,cv2.IMREAD_COLOR)
colors = ['red', 'green', 'yellow', 'blue']



if img1.shape == img2.shape:
    print("same size")
else:
    print("different size")
    img2 = cv2.resize(img2, (img1.shape[1], img1.shape[0]), interpolation = cv2.INTER_AREA)
```

```
i = 0
for sigma in sigmas:
    color = colors[i]
    i = i + 1
    corners1 = GetHarrisCorners(img1, sigma)
    corners2 = GetHarrisCorners(img2, sigma)

    # SSD metric
    l1, l2 = Correspondence_SSD(img1, corners1, img2, corners2, window_size = 21)
    plot_correspondece(img1, l1, img2, l2, color = color)
    plt.axis('off')
    plt.savefig("/home/xu1363/Documents/ECE 661/hw4/output/pair2/"+"pair2_SSD_sigma"+str(sigma)+".jpeg")

    # NCC metric
    l1, l2 = Correspondence_NCC(img1, corners1, img2, corners2, window_size = 21)
    plot_correspondece(img1, l1, img2, l2, color = color)
    plt.axis('off')
    plt.savefig("/home/xu1363/Documents/ECE 661/hw4/output/pair2/"+"pair2_NCC_sigma"+str(sigma)+".jpeg")

# SIFT
kps1, des1 = getSIFT(img1)
kps2, des2 = getSIFT(img2)
l1, l2 = Correspondence_SIFT(img1, kps1, des1, img2, kps2, des2)
plot_correspondece(img1, l1, img2, l2)
plt.axis('off')
plt.savefig("/home/xu1363/Documents/ECE 661/hw4/output/pair2/"+"pair2_SIFT.jpeg")
```

```
# pair 3
sigmas = [0.5, 1, 1.5, 2]
directory = "/home/xu1363/Documents/ECE 661/hw4/hw4_Task1_Images/pair3/"
file1 = "1.jpg"
file2 = "2.jpg"
img1 = cv2.imread(directory+file1,cv2.IMREAD_COLOR)
img2 = cv2.imread(directory+file2,cv2.IMREAD_COLOR)
colors = ['red', 'green', 'yellow', 'blue']

if img1.shape == img2.shape:
   print("same size")
else:
   print("different size")
   img2 = cv2.resize(img2, (img1.shape[1], img1.shape[0]), interpolation = cv2.INTER_AREA)
i = 0
for sigma in sigmas:
   color = colors[i]
   i = i + 1
   corners1 = GetHarrisCorners(img1, sigma)
   corners2 = GetHarrisCorners(img2, sigma)

   # SSD metric
   l1, l2 = Correspondence_SSD(img1, corners1, img2, corners2, window_size = 21)
   plot_correspondece(img1, l1, img2, l2, color = color)
   plt.axis('off')
   plt.savefig("/home/xu1363/Documents/ECE
661/hw4/output/pair3/"+"pair3_SSD_sigma"+str(sigma)+".jpeg")
```

```
    # NCC metric
    l1, l2 = Correspondence_NCC(img1, corners1, img2, corners2, window_size = 21)
    plot_correspondece(img1, l1, img2, l2, color = color)
    plt.axis('off')
    plt.savefig("/home/xu1363/Documents/ECE
661/hw4/output/pair3/"+"pair3_NCC_sigma"+str(sigma)+".jpeg")

# SIFT
kps1, des1 = getSIFT(img1)
kps2, des2 = getSIFT(img2)
l1, l2 = Correspondence_SIFT(img1, kps1, des1, img2, kps2, des2)
plot_correspondece(img1, l1, img2, l2)
plt.axis('off')
plt.savefig("/home/xu1363/Documents/ECE 661/hw4/output/pair3/"+"pair3_SIFT.jpeg")



# my pair 1
sigmas = [0.5, 1, 1.5, 2]
directory = "/home/xu1363/Documents/ECE 661/hw4/hw4_Task1_Images/mypair1/"
file1 = "1.jpg"
file2 = "2.jpg"
img1 = cv2.imread(directory+file1,cv2.IMREAD_COLOR)
img2 = cv2.imread(directory+file2,cv2.IMREAD_COLOR)
colors = ['red', 'green', 'yellow', 'blue']

if img1.shape == img2.shape:
    print("same size")
```

```
else:
    print("different size")
    img2 = cv2.resize(img2, (img1.shape[1], img1.shape[0]), interpolation = cv2.INTER_AREA)

i = 0
for sigma in sigmas:
    color = colors[i]
    i = i + 1
    corners1 = GetHarrisCorners(img1, sigma)
    corners2 = GetHarrisCorners(img2, sigma)

    # SSD metric
    l1, l2 = Correspondence_SSD(img1, corners1, img2, corners2, window_size = 21)
    plot_correspondece(img1, l1, img2, l2, color = color)
    plt.axis('off')
    plt.savefig("/home/xu1363/Documents/ECE
661/hw4/output/mypair1/"+"mypair1_SSD_sigma"+str(sigma)+".jpeg")

    # NCC metric
    l1, l2 = Correspondence_NCC(img1, corners1, img2, corners2, window_size = 21)
    plot_correspondece(img1, l1, img2, l2, color = color)
    plt.axis('off')
    plt.savefig("/home/xu1363/Documents/ECE
661/hw4/output/mypair1/"+"mypair1_NCC_sigma"+str(sigma)+".jpeg")

# SIFT
kps1, des1 = getSIFT(img1)
kps2, des2 = getSIFT(img2)
l1, l2 = Correspondence_SIFT(img1, kps1, des1, img2, kps2, des2)
```

```
plot_correspondece(img1, l1, img2, l2)
plt.axis('off')
plt.savefig("/home/xu1363/Documents/ECE 661/hw4/output/mypair1/"+"mypair1_SIFT.jpeg")



# my pair 2
sigmas = [0.5, 1, 1.5, 2]
directory = "/home/xu1363/Documents/ECE 661/hw4/hw4_Task1_Images/mypair2/"
file1 = "1.jpg"
file2 = "2.jpg"
img1 = cv2.imread(directory+file1,cv2.IMREAD_COLOR)
img2 = cv2.imread(directory+file2,cv2.IMREAD_COLOR)
colors = ['red', 'green', 'yellow', 'blue']

if img1.shape == img2.shape:
    print("same size")
else:
    print("different size")
    img2 = cv2.resize(img2, (img1.shape[1], img1.shape[0]), interpolation = cv2.INTER_AREA)

i = 0
for sigma in sigmas:
    color = colors[i]
    i = i + 1
    corners1 = GetHarrisCorners(img1, sigma)
    corners2 = GetHarrisCorners(img2, sigma)
    # SSD metric
    l1, l2 = Correspondence_SSD(img1, corners1, img2, corners2, window_size = 21)
```

```
    plot_correspondece(img1, l1, img2, l2, color = color)
    plt.axis('off')
    plt.savefig("/home/xu1363/Documents/ECE 661/hw4/output/mypair2/"+"mypair2_SSD_sigma"+str(sigma)+".jpeg")

    # NCC metric
    l1, l2 = Correspondence_NCC(img1, corners1, img2, corners2, window_size = 21)
    plot_correspondece(img1, l1, img2, l2, color = color)
    plt.axis('off')
    plt.savefig("/home/xu1363/Documents/ECE 661/hw4/output/mypair2/"+"mypair2_NCC_sigma"+str(sigma)+".jpeg")

# SIFT
kps1, des1 = getSIFT(img1)
kps2, des2 = getSIFT(img2)
l1, l2 = Correspondence_SIFT(img1, kps1, des1, img2, kps2, des2)
plot_correspondece(img1, l1, img2, l2)
plt.axis('off')
plt.savefig("/home/xu1363/Documents/ECE 661/hw4/output/mypair2/"+"mypair2_SIFT.jpeg")
```